Общество с ограниченной ответственностью «Платформа Данных Селена» 123022, г. Москва, вн.тер.г. муниципальный округ Пресненский, ул. 2-я Звенигородская, д. 13, стр. 37 ИНН 9703195303, КПП 770301001, ОГРН 1247700725491

Платформа Данных Селена

РУКОВОДСТВО АДМИНИСТРАТОРА ДАННЫХ

Версия программы 3.0

Москва
2026

Аннотация

Настоящее руководство предназначено для администраторов данных и описывает ключевые задачи при работе с Платформой Данных Селена (далее Selena): проектирование таблиц, загрузку и выгрузку данных, работу с озёрами данных и внешними каталогами, а также приёмы ускорения запросов.

Материал ориентирован на специалистов, отвечающих за организацию хранения и обработки данных, и предполагает знание основ SQL. Руководство может использоваться как для последовательного изучения, так и в качестве справочника при решении практических задач.

© ООО «Платформа Данных Селена», 2026

Оглавление

1 Лучшие практики

1.1 Обзор
1.2 Партиционирование
1.3 Кластеризация таблиц
1.4 Bucketing
1.5 Таблицы Primary Key
1.6 Аутентификация и авторизация
1.7 Конфигурация групп ресурсов на основе audit log
1.8 Query Tuning
1.8.1 Введение в оптимизацию запросов
1.8.2 План запроса
1.8.3 Обзор Query Profile
1.8.4 Рецепты настройки запросов
1.8.5 Рецепты настройки схемы
1.8.6 Explain Analyze
1.8.7 Query Hint
1.8.8 Метрики Query Profile

2 Проектирование таблиц

2.1 Каталоги, базы данных и таблицы
2.2 Обзор таблиц
2.3 Обзор типов таблиц
2.3.3 Возможности различных типов таблиц
2.3.4 Primary Key table
2.3.5 Duplicate Key таблица
2.3.6 Aggregate таблица
2.3.7 Таблица с уникальным ключом
2.4 Распределение данных
2.4.5 Expression partitioning (рекомендуется)
2.4.6 List partitioning (Legacy)
2.4.7 Range partitioning (Устаревший)
2.4.8 Временная партиция
2.4.9 Поддержка функций: Распределение данных
2.5 Сжатие данных
2.6 Indexes
2.6.1 Индексы
2.6.2 Prefix индексы
2.6.3 Bitmap индексы
2.6.4 Bloom filter индексы
2.6.5 N-gram bloom filter индекс
2.6.6 Полнотекстовый inverted индекс
2.6.7 Vector индекс
2.7 Гибридное строчно-столбцовое хранилище

3 Загрузка данных

3.1 Обзор загрузки
- 3.1.1 Варианты загрузки
- 3.1.2 Концепции загрузки
- 3.1.3 Рекомендации
- 3.1.4 Поддержка функций: Загрузка и выгрузка данных
- 3.1.5 Устранение неполадок при загрузке данных

3.2 Загрузка данных из объектного хранилища
- 3.2.1 Загрузка данных из AWS S3
- 3.2.2 Загрузка данных из Microsoft Azure Storage
- 3.2.3 Загрузка данных из GCS
- 3.2.4 Загрузка данных из MinIO

3.3 Загрузка данных из локальной файловой системы

3.4 Загрузка данных из HDFS

3.5 Загрузка из Apache Kafka
- 3.5.1 Загрузка данных с помощью Kafka connector
- 3.5.2 Загрузка данных с помощью Routine Load

3.6 Загрузка из Apache Spark
- 3.6.1 Загрузка данных с помощью Spark connector (рекомендуется)
- 3.6.2 Массовая загрузка данных с помощью Spark Load

3.7 Загрузка данных с помощью INSERT

3.8 SQL-транзакции

3.9 Загрузка данных с использованием транзакционного интерфейса Stream Load

3.10 Синхронизация в реальном времени из MySQL

3.11 Непрерывная загрузка данных из Apache Flink®

3.12 Непрерывная загрузка данных из Apache Pulsar

3.13 Изменение данных через загрузку

3.14 Трансформация данных при загрузке

3.15 Загрузка данных с помощью инструментов

3.16 Строгий режим

4 Выгрузка данных

4.1 Экспорт данных с помощью EXPORT

4.2 Чтение данных из Selena с помощью Spark connector

4.3 Чтение данных из Selena с помощью Flink connector

4.4 Взаимодействие с Selena через Arrow Flight SQL

4.5 Выгрузка данных с помощью INSERT INTO FILES

4.6 Поддержка функций: Загрузка и выгрузка данных

5 Data Lakehouse

5.7 Обзор Catalog
- 5.7.1 Обзор
- 5.7.2 Default catalog
- 5.7.3 Hive catalog
- 5.7.4 Iceberg catalog
  - Time Travel с Iceberg Catalog
  - Таблицы метаданных Iceberg

Создание Iceberg REST Catalog для AWS S3 Tables
Настройка безопасности для Iceberg REST Catalog
5.7.5 Hudi catalog
5.7.6 Delta Lake catalog
5.7.7 JDBC catalog
5.7.8 Elasticsearch catalog
5.7.9 Paimon catalog
5.7.10 Unified catalog
5.7.11 MaxCompute catalog
5.7.12 Kudu catalog
5.7.13 Руководство по Iceberg Lakehouse
5.7.15 Запрос внешних данных
5.8 Кэш данных
5.8.1 Data Cache
5.8.2 Прогрев Block cache
5.8.3 Наблюдаемость Data Cache
5.8.4 FAQ по Data Cache
5.9 Внешняя таблица
5.10 Файловая внешняя таблица
5.11 FAQ по Data Lake
5.12 Поддержка функций: аналитика Data Lake
6 Ускорение запросов
6.1 Просмотр выполняющихся запросов
6.2 Сбор статистики для CBO
6.3 SQL Plan Manager
6.4 Query Feedback
6.5 Синхронное материализованное представление
6.6 Asynchronous materialized views
6.6.1 Асинхронные материализованные представления
6.6.2 Ссылки на асинхронные материализованные представления
6.6.3 Use cases
Моделирование данных с помощью материализованных представлений
Перезапись запросов с помощью материализованных представлений
Ускорение запросов к озеру данных с помощью материализованных представлений
Создание разбитого материализованного представления
6.6.4 Устранение неполадок асинхронных материализованных представлений
6.6.5 Поддержка функций: Асинхронные материализованные представления
6.6.6 Понимание выполнения задач материализованного представления
6.7 Colocate Join
6.8 Использование Lateral Join для преобразования столбцов в строки
6.9 Caching
6.9.1 Кэш запросов
6.9.2 Кэш данных (Data Cache)
6.10 Computing distinct value count
6.10.1 Bitmap для Count Distinct
6.10.2 Использование HLL для приблизительного count distinct

6.11 Потоковая агрегация с сортировкой
6.12 Ускорение COUNT(DISTINCT) и Join с помощью AUTO INCREMENT и глобального словаря
6.13 Flat JSON
6.14 Ускорение соединений, сохраняющих кардинальность
6.15 JIT-компиляция для выражений
6.16 Skew Join V2

1 Лучшие практики

1.1 Обзор

Эти лучшие практики написаны опытными инженерами баз данных. Проектирование с учетом эффективности не только улучшает скорость выполнения запросов, но и снижает затраты за счет уменьшения хранилища, использования CPU и затрат на API объектного хранилища (например, S3).

1.1.1 Общий дизайн таблиц

Три руководства, охватывающие:

- Партиционирование
- Кластеризация
- Bucketing

Узнайте о:

- Различиях между партиционированием и bucketing
- Когда использовать партиционирование
- Как выбрать эффективный ключ сортировки
- Выбор между hash и random bucketing

1.1.2 Таблицы Primary Key

Таблица Primary Key использует новый движок хранения, разработанный Selena. Его основное преимущество заключается в поддержке обновлений данных в режиме реального времени при сохранении эффективной производительности для сложных ad-hoc запросов. В аналитике бизнеса в реальном времени принятие решений может выиграть от использования таблиц Primary Key, которые используют самые свежие данные для анализа результатов в реальном времени, смягчая задержку данных в анализе данных.

Узнайте о:

- Выборе типа индекса первичного ключа
- Выборе первичного ключа
- Мониторинге и управлении использованием памяти

© ООО «Платформа Данных Селена», 2026

- Настройке производительности

1.1.3 Настройка запросов

Настройка запросов необходима для достижения высокой производительности и надежности в Selena. Этот раздел объединяет практические руководства, справочные материалы и практические рецепты, которые помогут вам анализировать, диагностировать и оптимизировать производительность запросов на каждом этапе — от написания SQL до интерпретации деталей выполнения.

Эффективная настройка запросов в Selena обычно следует процессу сверху вниз:

1. **Определите проблему**
2. **Соберите и проанализируйте информацию о выполнении**
3. **Найдите первопричину**
4. **Примените стратегии настройки**
5. **Проверьте и повторите**

Независимо от того, являетесь ли вы администратором баз данных, разработчиком или инженером данных, эти ресурсы помогут вам:

- Диагностировать и решать медленные или ресурсоемкие запросы
- Понимать выбор оптимизатора и детали выполнения
- Применять лучшие практики и продвинутые стратегии настройки

Начните с обзора, углубитесь в справочные материалы по мере необходимости и используйте рецепты и советы для решения реальных проблем производительности в Selena.

1.2 Партиционирование

Быстрая аналитика в Selena начинается с макета таблицы, который соответствует вашим шаблонам запросов. Это руководство превращает практический опыт в четкие правила для **партиционирования**, помогая вам:

- **Сканировать меньше данных** с помощью агрессивного отсечения партиций
- **Управлять задачами жизненного цикла** (TTL, удаления GDPR, tiering) с помощью операций только с метаданными

- **Масштабироваться плавно** по мере роста количества tenant'ов, объема данных или окон хранения
- **Контролируемая write amplification** — новые данные попадают в "горячую" партицию; compaction происходит в исторических партициях

Держите этот совет под рукой при моделировании новой таблицы или рефакторинге старой — каждый раздел дает критерии, направленные на конкретную цель, эвристику проектирования и операционные ограничения, чтобы вы избежали дорогостоящего перепартиционирования в будущем.

1.2.1 Партиционирование vs. Bucketing — разные задачи

Понимание различия между партиционированием и bucketing является фундаментальным при проектировании производительных таблиц Selena. Хотя оба помогают управлять большими наборами данных, они служат разным целям:

- **Партиционирование** позволяет Selena пропускать целые блоки данных во время выполнения запроса, используя отсечение партиций, и обеспечивает операции жизненного цикла только с метаданными, такие как удаление старых данных или данных конкретного tenant'a.
- **Bucketing**, с другой стороны, помогает распределять данные по tablet'ам для параллелизации выполнения запросов и балансировки нагрузки, особенно в сочетании с hash-функциями.

Аспект	Партиционирование	Bucketing (Hash/Random)
Основная цель	Грубое отсечение данных и управление жизненным циклом (TTL, архивирование).	Мелкозернистый параллелизм и локальность данных внутри каждой партиции.
Видимость планировщику	Партиции — это объекты каталога; FE может пропускать их через предикаты.	Только предикаты равенства поддерживают отсечение bucket'ов
Операции жизненного цикла	DROP PARTITION — операция только с метаданными, идеальна для удалений GDPR, ежемесячного очищения.	Bucket'ы нельзя удалить; они изменяются только с помощью `ALTER TABLE … MODIFY DISTRIBUTED BY`.
Типичное количество	10^2–10^4 на таблицу (дни, недели, tenant'ы).	10–120 на партицию; Selena `BUCKETS xxx` настраивает это.
Обработка перекоса	Объединяйте или разделяйте партиции; рассмотрите композитную/гибридную схему.	Увеличьте количество bucket'ов, хешируйте по составному ключу, изолируйте "китов" или используйте random bucketing
Красные флаги	>100 k партиций могут привести к значительному потреблению памяти для FE	>200 k tablet'ов на BE; tablet'ы, превышающие 10 GB, могут столкнуться с проблемами

Аспект	Партиционирование	Bucketing (Hash/Random)
		compaction.

1.2.2 Когда следует использовать партиционирование?

Тип таблицы	Партиционирование?	Типичный ключ
Fact / event stream	Да	`date_trunc('day', event_time)`
Огромная dimension (миллиарды строк)	Иногда	Время или дата изменения бизнес-ключа
Маленькая dimension / lookup	Нет	Полагайтесь на hash distribution

1.2.3 Выбор ключа партиционирования

1. **Время в первую очередь по умолчанию** — если 80% запросов включают фильтр по времени, начните с `date_trunc('day', dt)`.
2. **Изоляция tenant'a** — добавьте `tenant_id` в ключ, когда вам нужно управлять данными на основе tenant'a
3. **Выравнивание с периодом хранения** — поместите в ключ столбец, который вы планируете очищать.
4. **Составные ключи**: `PARTITION BY tenant_id, date_trunc('day', dt)` отсекает идеально, но создает `#tenants × #days` партиций. Держите ниже ≈ 100 k в общей сложности, иначе пострадают память FE и compaction BE.

1.2.4 Выбор детализации

Детализация `PARTITION BY date_trunc('day', dt)` должна корректироваться в зависимости от варианта использования. Вы можете использовать "hour", "day" или "month" и т.д. См. `date_trunc`

Детализация	Используйте когда	Плюсы	Минусы
Daily (по умолчанию)	Большинство BI и отчетности	Мало партиций (365/год); простой TTL	Менее точно для запросов "последние 3 часа"
Hourly	> 2 × tablet на день; всплески IoT	Изоляция горячих точек; 24 партиции/день	8 700 партиций/год
Weekly / Monthly	Исторический архив	Крошечные метаданные; легкое объединение	Более грубое отсечение

- **Правило большого пальца**: Держите каждую партицию ≤ 100 GB и ≤ 20 k tablet'ов/партицию с учетом replica.
- **Смешанная детализация**: Начиная с версии 3.4, Selena поддерживает смешанную детализацию путем объединения исторических партиций в более грубую

детализацию.

1.2.5 Примеры рецептов

Click-stream fact (single-tenant)

```
CREATE TABLE click_stream (
  user_id BIGINT,
  event_time DATETIME,
  url STRING,
  ...
)
DUPLICATE KEY(user_id, event_time)
PARTITION BY date_trunc('day', event_time)
DISTRIBUTED BY HASH(user_id) BUCKETS xxx;
```

SaaS метрики (multi-tenant, паттерн A)

Рекомендуется для большинства SaaS рабочих нагрузок. Отсекает по времени, держит строки tenant'a совместно расположенными.

```
CREATE TABLE metrics (
  tenant_id INT,
  dt DATETIME,
  metric_name STRING,
  v DOUBLE
)
PRIMARY KEY(tenant_id, dt, metric_name)
PARTITION BY date_trunc('DAY', dt)
DISTRIBUTED BY HASH(tenant_id) BUCKETS xxx;
```

Whale tenant composite (паттерн B)

Когда необходимы DML/DDL операции для конкретного tenant'a или присутствуют крупномасштабные tenant'ы, будьте осторожны с потенциальным взрывом партиций.

```
CREATE TABLE activity (
  tenant_id INT,
  dt DATETIME,
  id BIGINT,
  ....
)
DUPLICATE KEY(dt, id)
PARTITION BY tenant_id, date_trunc('MONTH', dt)
DISTRIBUTED BY HASH(id) BUCKETS xxx;
```

1.3 Кластеризация таблиц

Продуманный ключ сортировки — это самый эффективный рычаг физического проектирования в Selena. Это руководство объясняет, как работает ключ сортировки под капотом, какие системные преимущества он открывает, и предоставляет конкретный план для выбора эффективного ключа для вашей рабочей нагрузки.

1.3.1 Пример

Предположим, у вас есть телеметрическая система, которая получает миллиарды строк в день, каждая помечена `device_id` и `ts` (timestamp). Определение `ORDER BY (device_id, ts)` для вашей таблицы фактов обеспечивает:

- Точечные запросы по `device_id` выполняются за миллисекунды.
- Дашборды фильтруют последние временные окна для каждого устройства, отсекая большую часть данных.
- Агрегации вроде `GROUP BY device_id` получают выгоду от потоковой агрегации.
- Сжатие улучшается благодаря последовательности близких временных меток для каждого устройства.

Этот простой двухколоночный ключ сортировки `ORDER BY (device_id, ts)` обеспечивает сокращение I/O, экономию CPU и более стабильную производительность запросов для миллиардов строк.

```
CREATE TABLE telemetry (
  device_id VARCHAR,
  ts DATETIME,
  value DOUBLE
)
ENGINE=OLAP
PRIMARY KEY(device_id, ts)
PARTITION BY date_trunc('day', ts)
DISTRIBUTED BY HASH(device_id) BUCKETS 16
ORDER BY (device_id, ts);
```

1.3.2 Преимущества в деталях

1. Массовое устранение I/O — Pruning сегментов и страниц

   Как это работает:

   Каждый сегмент и страница в 64 КБ хранят минимальные/максимальные значения для всех колонок. Если предикат выходит за пределы этого диапазона, Selena пропускает весь блок и никогда не обращается к диску.

   Пример:

   ```
   SELECT count(*)
   FROM events
   WHERE tenant_id = 42
     AND ts BETWEEN '2025-05-01' AND '2025-05-07';
   ```

   С `ORDER BY (tenant_id, ts)` рассматриваются только сегменты, у которых первый ключ равен 42, и в них только те страницы, чье временное окно ts пересекается с этими семью днями. Таблица из 100 млрд строк может сканировать менее 1 млрд строк, превращая минуты в секунды.

2. Точечные поиски за миллисекунды — Sparse Prefix Index

   Как это работает:

   Разреженный префиксный индекс хранит каждое ~1000-е значение ключа сортировки. Бинарный поиск попадает на нужную страницу, затем одно чтение с диска (часто уже

в кэше) возвращает строку.

Пример:

```
SELECT *
FROM orders
WHERE order_id = 982347234;
```

С `ORDER BY (order_id)` поиск требует ≈ 50 сравнений ключей для таблицы из 50 млрд строк — латентность менее 10 мс даже на холодном кэше данных.

3. Более быстрая сортированная агрегация

Как это работает:

Когда ключ сортировки совпадает с GROUP BY, Selena выполняет потоковую агрегацию при сканировании — не требуется сортировка или хеш-таблица.

Этот план сортированной агрегации сканирует строки в порядке ключа сортировки и формирует группы на лету, используя локальность CPU-кэша и пропуская промежуточную материализацию.

Пример:

```
SELECT device_id, COUNT(*)
FROM   telemetry
WHERE  ts BETWEEN '2025-01-01' AND '2025-01-31'
GROUP  BY device_id;
```

Если таблица имеет `ORDER BY (device_id, ts)`, движок группирует строки по мере их поступления — без построения хеш-таблицы или пересортировки. Для высококардинальных ключей, таких как device_id, это может радикально снизить использование CPU и памяти.

Потоковая агрегация с сортированным вводом обычно улучшает пропускную способность в 2–3× по сравнению с хеш-агрегацией для больших мощностей групп.

4. Более высокое сжатие и горячий кэш

Как это работает:

Сортированные данные показывают небольшие дельты или длинные последовательности, ускоряя словарное кодирование, RLE и frame-of-reference кодирование. Компактные страницы последовательно передаются через CPU-кэши.

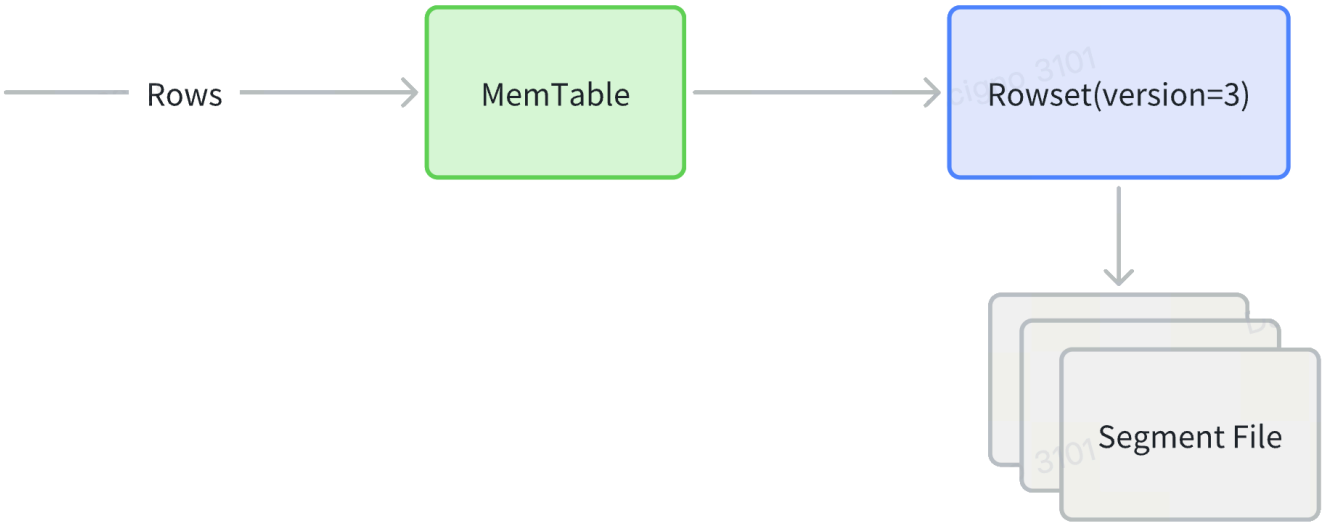
Пример:

Таблица телеметрии, отсортированная по (device_id, ts), достигла в 1.8× лучшего сжатия (LZ4) и на 25% меньшего CPU/сканирование, чем те же данные, загруженные без сортировки.

1.3.3 Как работает ключ сортировки

Влияние ключа сортировки начинается с момента записи строки и сохраняется через каждую оптимизацию при чтении. Этот раздел проходит через жизненный цикл — путь записи ➜ иерархия хранения ➜ внутренности сегмента ➜ путь чтения — чтобы показать, как каждый слой усиливает ценность упорядочивания.

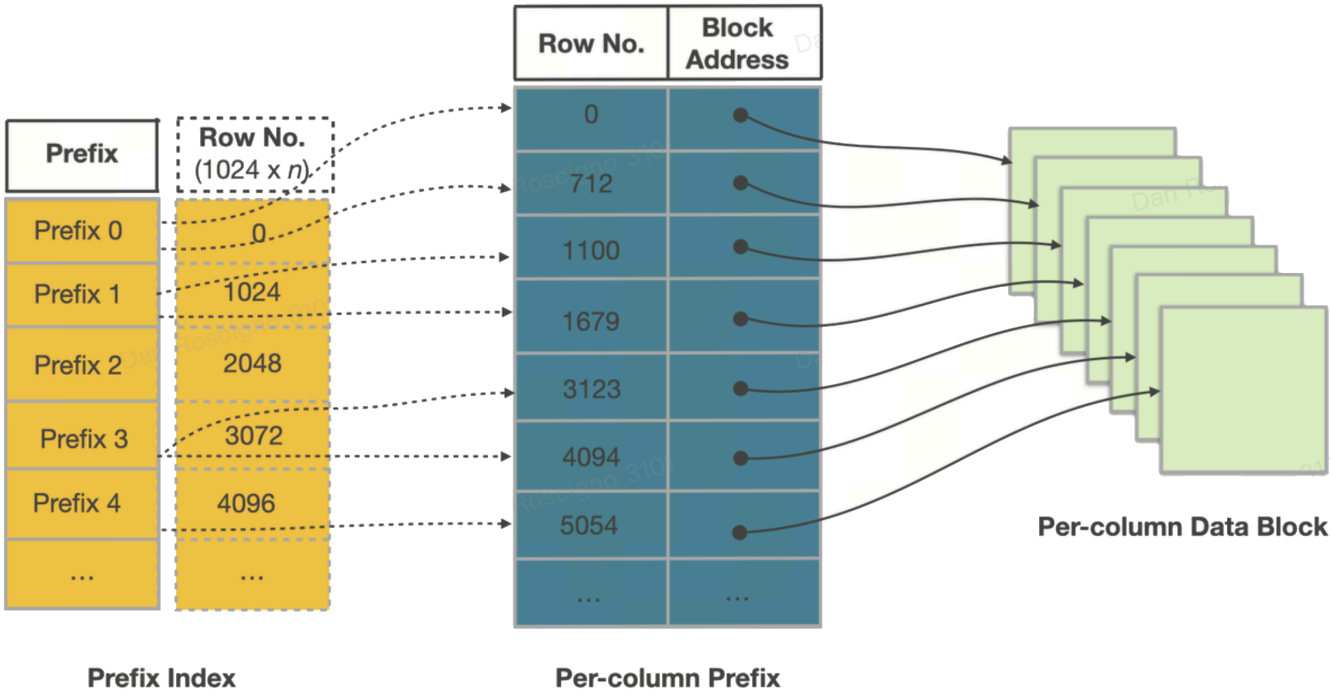
1. Путь записи
    i. Ingest: строки попадают в MemTable, сортируются по объявленному ключу сортировки, затем сбрасываются как новый Rowset, содержащий один или несколько упорядоченных сегментов.
    ii. Compaction: фоновые cumulative/base задачи объединяют множество маленьких Rowset'ов в более крупные, восстанавливая удаления и снижая количество сегментов без пересортировки, поскольку каждый исходный Rowset уже имеет тот же порядок.
    iii. Replication: каждый Tablet (shard, владеющий Rowset'ом) синхронно реплицируется на узлы Back-End, гарантируя, что сортированный порядок согласован между репликами.



2. Иерархия хранения

Объект	Что это	Почему важно для ключа сортировки
Partition	Грубозернистый логический срез таблицы (например, дата или tenant_id).	Позволяет partition pruning на этапе планировщика и изолирует операции жизненного цикла (TTL, массовая загрузка).
Tablet	Hash/random bucket внутри partition, независимо реплицируемый между узлами Back-End.	Единица, строки которой физически упорядочены по ключу сортировки; весь внутрипартиционный pruning начинается здесь.
MemTable	Буфер записи в памяти (~96 МБ), который сортирует по объявленному ключу перед сбросом на диск.	Гарантирует, что каждый сегмент на диске уже упорядочен — внешняя сортировка не требуется позже.
Rowset	Неизменяемый набор одного или нескольких сегментов, созданный сбросом, потоковой загрузкой или циклом compaction.	Дизайн append-only позволяет Selena загружать данные одновременно, пока читатели остаются без блокировок.
Segment	Самодостаточный колоночный файл (~512 МБ) внутри Rowset'a, содержащий страницы данных плюс pruning-индексы.	Zone-map и prefix-индексы на уровне сегмента зависят от порядка, установленного на этапе MemTable.

3. Внутри файла сегмента



Каждый сегмент самоописываемый. Сверху вниз вы найдете:

- Страницы данных колонок: блоки по 64 КБ, закодированные (Dictionary, RLE, Delta) и сжатые (по умолчанию LZ4).
- Ordinal index: отображает порядковый номер строки → смещение страницы, чтобы движок мог перейти непосредственно к странице n.
- Zone-map index: min, max и has_null для каждой страницы и для всего сегмента — первая линия защиты для pruning.
- Short-key (prefix) index: разреженная таблица бинарного поиска первых 36 байт ключа сортировки каждые ~1K строк — позволяет точечные/диапазонные поиски за миллисекунды.
- Footer & magic number: смещения для каждого индекса и контрольная сумма для целостности; позволяет Selena отображать в память только хвост, чтобы обнаружить остальное.

Поскольку страницы уже отсортированы по ключу, эти индексы крошечные, но чрезвычайно эффективные.

4. Путь чтения

    i. Partition pruning (время планировщика): если WHERE ограничивает ключ partition (например, `dt BETWEEN '2025-05-01' AND '2025-05-07'`), оптимизатор открывает только соответствующие каталоги partition.

ii. Tablet pruning (время планировщика): когда фильтр равенства включает колонку хеш-распределения, Selena вычисляет целевые ID tablet'ов и планирует только эти Tablet'ы.
iii. Prefix-index seek: разреженный short-key индекс по ведущим колонкам сортировки направляет точно к нужному сегменту или странице.
iv. Zone-map pruning: метаданные min/max для каждого сегмента и страницы 64 КБ отбрасывают блоки, которые не попадают в окно предиката.
v. Vectorized scan & late materialization: выжившие страницы колонок последовательно передаются через CPU-кэши; материализуются только ссылочные строки и колонки, сохраняя память тесной.

Поскольку данные зафиксированы в порядке ключа при каждом сбросе, каждый слой pruning при чтении усиливается предыдущим, обеспечивая сканирования менее секунды для таблиц с миллиардами строк.

1.3.4 Как выбрать эффективный ключ сортировки

1. Начните с анализа рабочей нагрузки

   Сначала проанализируйте top-N паттернов запросов:

   - Предикаты равенства (`=` / `IN`). Колонки, почти всегда фильтруемые по равенству, являются идеальными ведущими кандидатами.
   - Предикаты диапазона. Временные метки и числовые диапазоны обычно следуют за колонками равенства в ключе сортировки.
   - Ключи агрегации. Если колонка диапазона также появляется в `GROUP BY`, размещение её раньше в ключе (после селективных фильтров) может включить сортированную агрегацию.
   - Ключи join/group-by. Рассмотрите размещение ключей join или группировки рано, если они распространены.

   Измерьте кардинальность колонок: высококардинальные колонки (миллионы различных значений) лучше всего отсекают данные.

2. Эвристики и правила большого пальца

   i. Правило порядка: (высокоселективные колонки равенства) → (основная колонка диапазона) → (помощники кластера).
   ii. Упорядочивание по кардинальности: размещение низкокардинальных колонок перед высококардинальными может улучшить сжатие данных.

iii. Ширина: держите 3-5 колонок. Очень широкие ключи замедляют загрузку и переполняют 36-байтный лимит prefix-индекса.
iv. Строковые колонки: длинная ведущая строковая колонка может занять большую часть или весь 36-байтный лимит в prefix-индексе, не позволяя последующим колонкам в ключе сортировки эффективно индексироваться.

Это снижает мощность pruning prefix-индекса и ухудшает производительность точечных запросов.

3. Координируйте с другими рычагами проектирования

- Partitioning: выберите ключ partition, который грубее ведущей колонки сортировки (например, `PARTITION BY date`, `ORDER BY (tenant_id, ts)`). Таким образом, partition pruning сначала удаляет целые диапазоны дат, а затем sort pruning очищает внутри.
- Bucketing: использование одних и тех же колонок для bucketing и clustering служит разным целям. Bucketing обеспечивает равномерное распределение данных по cluster'у, в то время как сортировка позволяет эффективно устранять I/O.
- Тип таблицы: Primary-Key таблицы по умолчанию используют первичный ключ как ключ сортировки, но они также могут указывать дополнительные колонки для уточнения физического порядка и улучшения pruning. Aggregate и duplicate таблицы должны следовать стратегиям ключа сортировки, управляемым аналитическими предикатами, обсуждаемым выше.

4. Справочные шаблоны

Сценарий	Partition	Ключ сортировки	Обоснование
B2C заказы	date_trunc('day', order_ts)	(user_id, order_ts)	Большинство запросов фильтруют сначала по пользователю, затем по последним временным диапазонам.
IoT телеметрия	date_trunc('day', ts)	(device_id, ts)	Чтения временных рядов с областью действия устройства доминируют.
SaaS Multi-Tenant	tenant_id	(dt, event_id)	Изоляция tenant'а через partition; сортировка кластеризует по дню для дашбордов.
Поиск в измерении	none	(dim_id)	Маленькая таблица, чистые точечные поиски — одной колонки достаточно.

1.3.5 Заключение

Хорошо спроектированный ключ сортировки обменивает небольшие, предсказуемые накладные расходы при загрузке на радикальные улучшения латентности сканирования, эффективности хранения и использования CPU. Основывая свой выбор на реалиях рабочей нагрузки, уважая кардинальность и проверяя с помощью `EXPLAIN`, вы можете поддерживать Selena в работе, даже когда данные и количество пользователей растут в 10× и более.

1.4 Bucketing

Краткое практическое руководство по выбору между Hash Bucketing и Random Bucketing в Selena, включая их механику, компромиссы и рекомендуемые случаи использования.

1.4.1 Сравнительная таблица

Аспект	Hash Bucketing	Random Bucketing
Пример	`DISTRIBUTED BY HASH(id) BUCKETS 16`	`DISTRIBUTED BY RANDOM`
Объявление ключа	Требуется HASH(col1, …)	Нет — строки назначаются по кругу
Начальное количество bucket при пропуске	Выбирается автоматически при CREATE, затем фиксируется	Выбирается автоматически при CREATE; может расти, если установлен bucket_size
Разделение / сжатие tablet	Вручную ALTER … BUCKETS	Автоматическое разделение ⇢ только рост (≥ v1.5.2)
Устойчивость к перекосу	Зависит от кардинальности ключа	Высокая — равномерная по дизайну
Отсечение bucket	☐ (фильтры, JOIN)	☐ (полное сканирование tablet)
Colocate JOIN	☐	☐
Локальная агрегация / bucket-shuffle JOIN	☐	☐
Поддерживаемые типы таблиц	Все	Только таблицы Duplicate Key

1.4.2 Hash Bucketing

Как это работает

Строки назначаются tablet путем хеширования одного или нескольких столбцов. Количество tablet фиксируется после создания, если только не изменено вручную.

Требования

- Необходимо выбрать стабильный, равномерно распределенный ключ с высокой кардинальностью заранее. Кардинальность обычно должна быть в 1000 раз больше количества узлов BE, чтобы предотвратить перекос данных между хеш-bucket.
- Выберите подходящий размер bucket изначально, в идеале в диапазоне от 1 до 10 GB.

Сильные стороны

- Локальность запросов — выборочные фильтры и JOIN затрагивают меньше tablet.
- Colocate JOIN — таблицы фактов/измерений могут использовать общие хеш-ключи для высокоскоростных JOIN.
- Предсказуемая структура — строки с одним и тем же ключом всегда находятся вместе.
- Локальная агрегация и bucket-shuffle JOIN — идентичная хеш-структура между партициями позволяет локальную агрегацию и снижает затраты на перемешивание данных при больших JOIN.

Слабые стороны

- Уязвимость к горячим tablet, если распределение данных искажено.
- Количество tablet статично; масштабирование требует обслуживающего DDL.
- Недостаточное количество tablet может негативно повлиять на загрузку данных, компактирование данных и параллелизм выполнения запросов.
- Чрезмерное использование tablet увеличит объем метаданных.

© ООО «Платформа Данных Селена», 2026

Пример: JOIN измерений и фактов и отсечение tablet

```
-- Таблица фактов, партиционированная и с хеш-bucket по (customer_id)
CREATE TABLE sales (
  sale_id bigint,
  customer_id int,
  sale_date date,
  amount decimal(10,2)
) ENGINE = OLAP
DISTRIBUTED BY HASH(customer_id) BUCKETS 48
PARTITION BY date_trunc('DAY', sale_date)
PROPERTIES ("colocate_with" = "group1");

-- Таблица измерений с хеш-bucket по тому же ключу и количеству bucket,
colocate с таблицей sales
CREATE TABLE customers (
  customer_id int,
  region varchar(32),
  status tinyint
) ENGINE = OLAP
DISTRIBUTED BY HASH(customer_id) BUCKETS 48
PROPERTIES ("colocate_with" = "group1");


-- Selena может выполнить отсечение tablet
SELECT sum(amount)
FROM sales
WHERE customer_id = 123

-- Selena может выполнить локальную агрегацию
SELECT customer_id, sum(amount) AS total_amount
FROM sales
GROUP BY customer_id
ORDER BY total_amount DESC LIMIT 100;

-- Selena может выполнить colocate JOIN
SELECT c.region, sum(s.amount)
FROM sales s JOIN customers c USING (customer_id)
WHERE s.sale_date BETWEEN '2025-01-01' AND '2025-01-31'
GROUP BY c.region;
```

© ООО «Платформа Данных Селена», 2026

Что вы получаете из этого примера?

- **Отсечение tablet**: Предикат customer_id `WHERE customer_id = 123` позволяет отсечение bucket, позволяя запросу обращаться только к одному tablet, что снижает задержку и циклы CPU, особенно для точечных запросов.
- **Локальная агрегация**: когда ключ хеш-распределения является подмножеством ключа агрегации, Selena может обойти фазу агрегации с перемешиванием, снижая общие затраты.
- **Colocate JOIN**: поскольку обе таблицы используют одинаковое количество bucket и ключ, каждый BE может объединить свою локальную пару tablet без сетевого перемешивания.

Когда использовать

- Стабильные схемы с хорошо известными ключами фильтрации/JOIN распределения.
- Рабочие нагрузки хранилища данных, которые выигрывают от отсечения bucket.
- Вам нужна какая-то специфическая оптимизация, такая как colocate JOIN/bucket shuffle JOIN/локальная агрегация.
- Вы используете таблицы Aggregate/Primary Key.

1.4.3 Random Bucketing

Как это работает

Строки назначаются по кругу; ключ не указывается. С `PROPERTIES ("bucket_size"="<bytes>")`, Selena динамически разделяет tablet по мере роста партиций (v1.5.2+).

Сильные стороны

- **Нулевой технический долг** — нет ключей, нет математики bucket.
- **Защита от перекоса записей** — равномерное давление на диски и BE.
- **Эластичный рост** — разделение tablet сохраняет быстрый прием данных по мере роста данных или cluster.

Слабые стороны

- **Нет отсечения bucket** — каждый запрос сканирует все tablet в партиции.
- **Нет colocate JOIN** — структура без ключей предотвращает локальность.
- Ограничено таблицами **Duplicate Key** на сегодняшний день.

Когда использовать

- Таблицы логов/событий или мультитенантных SaaS, где ключи изменяются или искажаются.
- Конвейеры с интенсивной записью, где критична равномерная пропускная способность приема данных.

1.4.4 Операционные рекомендации

- Выберите размер bucket (например, 1 GiB) для random bucketing, чтобы включить автоматическое разделение.
- Для hash bucketing отслеживайте размер tablet; пересегментируйте до того, как tablet превысят 5–10 GiB

1.5 Таблицы Primary Key

Таблица Primary Key использует новый движок хранения, разработанный Selena. Его основное преимущество заключается в поддержке обновлений данных в режиме реального времени при сохранении эффективной производительности для сложных ad-hoc запросов. В аналитике бизнеса в реальном времени принятие решений может выиграть от использования таблиц Primary Key, которые используют самые свежие данные для анализа результатов в реальном времени, что может смягчить задержку данных в анализе данных. Однако первичный ключ — это не бесплатный обед. При неправильном использовании он может привести к ненужной трате ресурсов.

Поэтому в этом разделе мы расскажем вам, как более эффективно использовать модель первичного ключа для достижения желаемых результатов.

1.5.1 Выбор индекса Primary Key

Первичный индекс является наиболее критическим компонентом в таблице Primary Key. Индекс первичного ключа используется для хранения сопоставления между значениями первичного ключа и расположениями строк данных, идентифицируемых значениями первичного ключа.

В настоящее время мы поддерживаем три типа индекса первичного ключа:

1. Полностью в памяти индекс первичного ключа.

```
PROPERTIES (
    "enable_persistent_index" = "false"
);
```

2. Persistent индекс первичного ключа на основе локального диска.

```
PROPERTIES (
    "enable_persistent_index" = "true",
    "persistent_index_type" = "LOCAL"
);
```

3. Cloud native persistent индекс первичного ключа.

```
PROPERTIES (
    "enable_persistent_index" = "true",
    "persistent_index_type" = "CLOUD_NATIVE"
);
```

Мы НЕ рекомендуем использовать индексирование в памяти, так как это может привести к значительной трате ресурсов памяти.

Если вы используете shared-data (elastic) cluster Selena, мы рекомендуем выбрать cloud-native persistent индекс первичного ключа. В отличие от локального дискового persistent индекса первичного ключа, он хранит полные данные индекса в удаленном объектном хранилище, используя локальные диски только в качестве кеша. По сравнению с локальным дисковым persistent индексом первичного ключа, его преимущества включают:

1. Отсутствие зависимости от емкости локального диска.
2. Отсутствие необходимости перестраивать индексы после ребалансировки data shard'ов.

1.5.2 Выбор первичного ключа

Первичный ключ обычно не помогает ускорить запросы. Вы можете указать столбец, отличный от первичного ключа, в качестве ключа сортировки, используя предложение `ORDER BY`, чтобы ускорить запросы. Поэтому при выборе первичного ключа вам нужно учитывать только уникальность во время процессов импорта и обновления данных.

Чем больше первичный ключ, тем больше памяти, I/O и других ресурсов он потребляет. Поэтому обычно рекомендуется избегать выбора слишком большого количества или слишком больших столбцов в качестве первичного ключа. Максимальный размер первичного ключа по умолчанию составляет 128 байт, управляется параметром `primary_key_limit_size` в `be.conf`.

Вы можете увеличить `primary_key_limit_size`, чтобы выбрать больший первичный ключ, но имейте в виду, что это приведет к более высокому потреблению ресурсов.

Сколько места для хранения и памяти займет persistent индекс?

Формула стоимости места для хранения

`(key size + 8 bytes) * row count * 50%`

> **ПРИМЕЧАНИЕ**
>
> 50% — это оценочная эффективность сжатия, фактический эффект сжатия зависит от самих данных.

Формула стоимости памяти

`min(l0_max_mem_usage * tablet cnt, update_memory_limit_percent * BE process memory);`

1.5.3 Использование памяти

Память, используемая таблицей Primary Key, может отслеживаться через mem_tracker:

```
// Просмотр общей статистики памяти
http://be_ip:be_http_port/mem_tracker

// Просмотр статистики памяти таблицы primary key
http://be_ip:be_http_port/mem_tracker?type=update

// Просмотр статистики памяти таблицы primary key с более подробной
информацией
http://be_ip:be_http_port/mem_tracker?type=update&upper_level=4
```

Элемент `update` в `mem_tracker` записывает всю память, используемую таблицей Primary Key, такую как индекс первичного ключа, delete vector и так далее. Вы также можете отслеживать этот элемент `update` через сервис мониторинга метрик. Например, в Grafana вы можете проверить элемент update через (элемент в красной рамке):

> Подробнее о мониторинге и оповещениях с Prometheus и Grafana:
> https://docs.selena.io/docs/administration/management/monitoring/Monitor_and_Alert/

Если вы чувствительны к использованию памяти и хотите снизить потребление памяти во время процесса импорта в таблицу PK, вы можете достичь этого с помощью следующей конфигурации:

```
be.conf

l0_max_mem_usage = (некоторое значение меньше 104857600, по умолчанию
104857600)
skip_pk_preload = true

// Shared-nothing cluster
transaction_apply_worker_count = (некоторое значение меньше количества ядер
cpu, по умолчанию количество ядер cpu)

// Shared-data cluster
transaction_publish_version_worker_count = (некоторое значение меньше
количества ядер cpu, по умолчанию количество ядер cpu)
```

`l0_max_mem_usage` контролирует максимальное использование памяти persistent индекса первичного ключа на tablet. `transaction_apply_worker_count` и `transaction_publish_version_worker_count` оба контролируют максимальное количество потоков, которые могут использоваться для обработки upsert и delete в таблице primary key.

Но вам нужно помнить, что уменьшение `l0_max_mem_usage` может увеличить давление I/O, в то время как уменьшение `transaction_apply_worker_count` или `transaction_publish_version_worker_count` может замедлить прием данных.

1.5.4 Компромисс между ресурсами compaction, свежестью данных и задержкой запросов

По сравнению с таблицами других моделей, таблицы primary key требуют дополнительных операций для поиска по индексу первичного ключа и генерации delete vector во время импорта, обновления и удаления данных, что вносит дополнительные ресурсные затраты. Поэтому вам нужно сделать компромисс между этими тремя факторами:

- Ограничение ресурсов compaction.
- Свежесть данных
- Задержка запросов

Свежесть данных и задержка запросов

Если вы хотите получить лучшую свежесть данных, а также лучшую задержку запросов, это означает, что вы введете высокочастотные записи, а также захотите убедиться, что

они могут быть скомпактированы как можно скорее. Тогда вам потребуется больше ресурсов compaction для обработки этих записей:

```
// shared-data
be.conf
compact_threads = 4

// shared-nothing
be.conf
update_compaction_num_threads_per_disk = 1
update_compaction_per_tablet_min_interval_seconds = 120
```

Вы можете увеличить `compact_threads` и `update_compaction_num_threads_per_disk` или уменьшить `update_compaction_per_tablet_min_interval_seconds`, чтобы ввести больше ресурсов compaction для обработки высокочастотных записей.

Как узнать, могут ли текущие ресурсы compaction и настройки справиться с текущими высокочастотными записями? Вы можете наблюдать это следующими способами:

1. Для shared-data cluster, если compaction не может поспевать за скоростью приема, это может привести к замедлению приема или даже к ошибкам write failure и остановке приема. a. Замедление приема. Вы можете использовать `show proc /transactions/{db_name}/running';` для проверки текущих выполняющихся транзакций, и если появляется какое-либо сообщение о замедлении, например:

```
Partition's compaction score is larger than 100.0, delay commit for xxxms.
You can try to increase compaction concurrency
```

появляется в поле ErrMsg, это означает, что происходит замедление приема. Например:

```
mysql> show proc '/transactions/test_pri_load_c/running';
+---------------+-----------------------------------------------+-----------
-------+-------------------+-------------------+---------------------+----
--------+-------------+------------+-------------------------------------
-----------------------------------------------------------------------
----------+-------------------+------------+-----------+--------+
| TransactionId | Label                                         |
Coordinator      | TransactionStatus | LoadJobSourceType  | PrepareTime
| CommitTime | PublishTime | FinishTime | Reason
| ErrorReplicasCount | ListenerId | TimeoutMs | ErrMsg |
+---------------+-----------------------------------------------+-----------
-------+-------------------+-------------------+---------------------+----
--------+-------------+------------+-------------------------------------
-----------------------------------------------------------------------
----------+-------------------+------------+-----------+--------+
| 1034          | stream_load_d2753fbaa0b343acadd5f13de92d44c1 | FE:
172.26.94.39 | PREPARE           | FRONTEND_STREAMING | 2024-10-24 13:05:01
| NULL       | NULL        | NULL       | Partition's compaction score is
larger than 100.0, delay commit for 6513ms. You can try to increase
compaction concurrency, | 0                  | 11054      | 86400000  |
|
+---------------+-----------------------------------------------+-----------
-------+-------------------+-------------------+---------------------+----
--------+-------------+------------+-------------------------------------
-----------------------------------------------------------------------
----------+-------------------+------------+-----------+--------+
```

b. Остановка приема. Если есть ошибка приема, такая как:

```
Failed to load data into partition xxx, because of too large compaction
score, current/limit: xxx/xxx. You can reduce the loading job concurrency,
or increase compaction concurrency
```

Это означает, что прием остановлен, потому что compaction не может поспевать за текущими высокочастотными записями.

2. Для shared-nothing cluster нет стратегии замедления приема, если compaction не может поспевать за текущими высокочастотными записями. Прием завершится неудачей и вернет сообщение об ошибке:

```
Failed to load data into tablet xxx, because of too many versions,
current/limit: xxx/xxx. You can reduce the loading job concurrency, or
increase loading data batch size. If you are loading data with Routine
Load, you can increase FE configs routine_load_task_consume_second and
max_routine_load_batch_size.
```

Свежесть данных и ограничение ресурсов compaction

Если у вас ограниченные ресурсы compaction, но все еще нужно поддерживать достаточную свежесть данных, это означает, что вам нужно пожертвовать некоторой задержкой запросов.

Вы можете внести эти изменения в конфигурацию для достижения этого:

- Shared-data cluster

```
fe.conf

lake_ingest_slowdown_threshold = xxx (по умолчанию 100, вы можете
увеличить)
lake_compaction_score_upper_bound = xxx (по умолчанию 2000, вы можете
увеличить)
```

Параметр `lake_ingest_slowdown_threshold` контролирует порог для инициирования замедления приема. Когда показатель compaction партиции превышает этот порог, система начнет замедлять прием данных. Аналогично, `lake_compaction_score_upper_bound` определяет порог для инициирования остановки приема.

- Shared-nothing cluster

```
be.conf

tablet_max_versions = xxx (по умолчанию 1000, вы можете увеличить)
```

`tablet_max_versions` определяет порог для инициирования остановки приема.

Увеличивая эти конфигурации, система может вместить больше небольших файлов данных и снизить частоту compaction, но это также повлияет на задержку запросов.

Задержка запросов и ограничение ресурсов compaction

Если вы хотите достичь хорошей задержки запросов с ограниченными ресурсами compaction, вам нужно снизить частоту записи и создавать более крупные пакеты данных для приема.

Для конкретной реализации, пожалуйста, обратитесь к разделам о различных методах приема, которые подробно описывают, как снизить частоту приема и увеличить размер пакета.

1.6 Аутентификация и авторизация

Эта тема направлена на предоставление последовательного руководства по лучшим практикам разработки вашего собственного рабочего процесса аутентификации и авторизации.

Подробные инструкции по каждой операции, упомянутой ниже, см. по ссылкам в разделе См. также.

1.6.1 Реальный корпоративный сценарий

Крупные предприятия часто имеют сложные организационные структуры и большое количество сотрудников, использующих различные платформы и инструменты. С точки зрения IT-управления наличие единой системы идентификации, аутентификации и авторизации приносит значительные преимущества:

- **Упрощенное управление пользователями**: Администраторам больше не нужно вручную создавать или удалять пользователей и назначать разрешения в нескольких системах. Управление жизненным циклом пользователей (например, прием/увольнение) становится бесшовным и удобным для аудита.
- **Повышенная безопасность**: Механизм единого входа (SSO) устраняет необходимость для пользователей управлять несколькими учетными данными, уменьшая поверхность атаки.
- **Контроль доступа на основе ролей**: Разрешения доступа обычно привязаны к роли или отделу пользователя. Хорошо структурированная система идентификации обеспечивает более простые и точные решения по авторизации.

Пример

Предположим, что три новых сотрудника присоединяются к разным отделам SaaS-компании: один специалист по маркетингу и два архитектора решений.

- **Организационно** они принадлежат к разным командам.
- **С точки зрения идентификации** их email-аккаунты служат учетными данными для входа на внутренние платформы.
- **По правам доступа** каждому из трех предоставлен доступ к различным платформам:
  - Специалист по маркетингу может войти в backend Hubspot для просмотра новых лидов.
  - Архитекторы решений могут получить доступ к консоли сервисов и управлять сервисами для назначенных клиентов.

Хотя все трое используют одного и того же провайдера идентификации, их права доступа строго соблюдаются:

- Специалист по маркетингу имеет доступ только к Hubspot.
- Архитекторы решений могут получить доступ к консоли сервисов, но они не могут получить доступ к сервису для пользователей, к которым они не назначены. Они также не могут получить доступ к Hubspot.

1.6.2 Три уровня контроля доступа

Этот пример подчеркивает три ключевых компонента в корпоративном потоке идентификации и доступа:

1. **Аутентификация идентичности** – "Я Петр, проверенный сотрудник SaaS-компании."
2. **Аутентификация доступа** – "Как архитектор решений, я уполномочен войти в консоль сервисов." (Не все проверенные сотрудники должны иметь доступ ко всем сервисам.)
3. **Авторизация действий** – "Как клиент SaaS-компании, я могу просматривать информацию нашего собственного сервиса, но не других клиентов."

В контексте базы данных

Эти уровни контроля доступа также применимы к системе баз данных:

1. **Проверка идентичности**: Подтверждение, что пользователь является действительным сотрудником со своим собственным паролем.

2. **Аутентификация доступа**: Проверка, что пользователь или его группа имеет разрешение войти в конкретный cluster.
3. **Авторизация операций**: Проверка, может ли пользователь выполнить запрос, загрузить данные и т.д.

Как вы можете видеть, аутентификация и авторизация тесно связаны на практике. Запрос аутентификации пользователя часто подразумевает более широкие требования к контролю доступа. Поэтому важно понимать полный поток доступа.

1.6.3 Ключевые концепции

LDAP

Lightweight Directory Access Protocol (LDAP) — это протокол для доступа и поддержки распределенной информации каталогов. Вы можете думать о нем как о глобальной адресной книге вашей организации:

- Каждый пользователь имеет уникальный путь (Distinguished Name, DN).
- LDAP хранит базовую информацию о пользователях, включая пароли.
- LDAP также управляет структурами групп и членством.
- Запросы `ldapsearch` могут извлекать пользователей или группы.

LDAP может использоваться:

- Как источник **аутентификации** (для проверки имен пользователей и паролей).
- Как провайдер **информации о группах** для контроля доступа.

UNIX Groups

Иногда пользователи дублируют группы LDAP локально (на хостовой ОС) по соображениям безопасности или изоляции, избегая прямого взаимодействия с внешними серверами LDAP. Эти локальные UNIX-группы могут использоваться для аутентификации или контроля доступа.

OAuth, OIDC и JWT

ПОДСКАЗКА

Краткое объяснение терминов

- **ID Token**: Подтверждение личности (я — это я).
- **Access Token**: Подтверждение права доступа к определённым ресурсам (я могу делать определённые вещи).
- **OAuth 2.0**: Фреймворк авторизации, предоставляющий access-токены.
- **OIDC**: Слой аутентификации поверх OAuth. Предоставляет ID и Access токены.
- **JWT**: Формат токена. Используется как OAuth, так и OIDC.

Практическое применение:

- **OAuth-вход**: Перенаправляет на внешнюю страницу входа (например, Google), затем обратно в cluster. Требует доступа к браузеру и предварительной настройки redirect URL.
- **JWT-вход**: Пользователь передаёт токен напрямую в cluster, что требует предварительной настройки публичного ключа или endpoint.

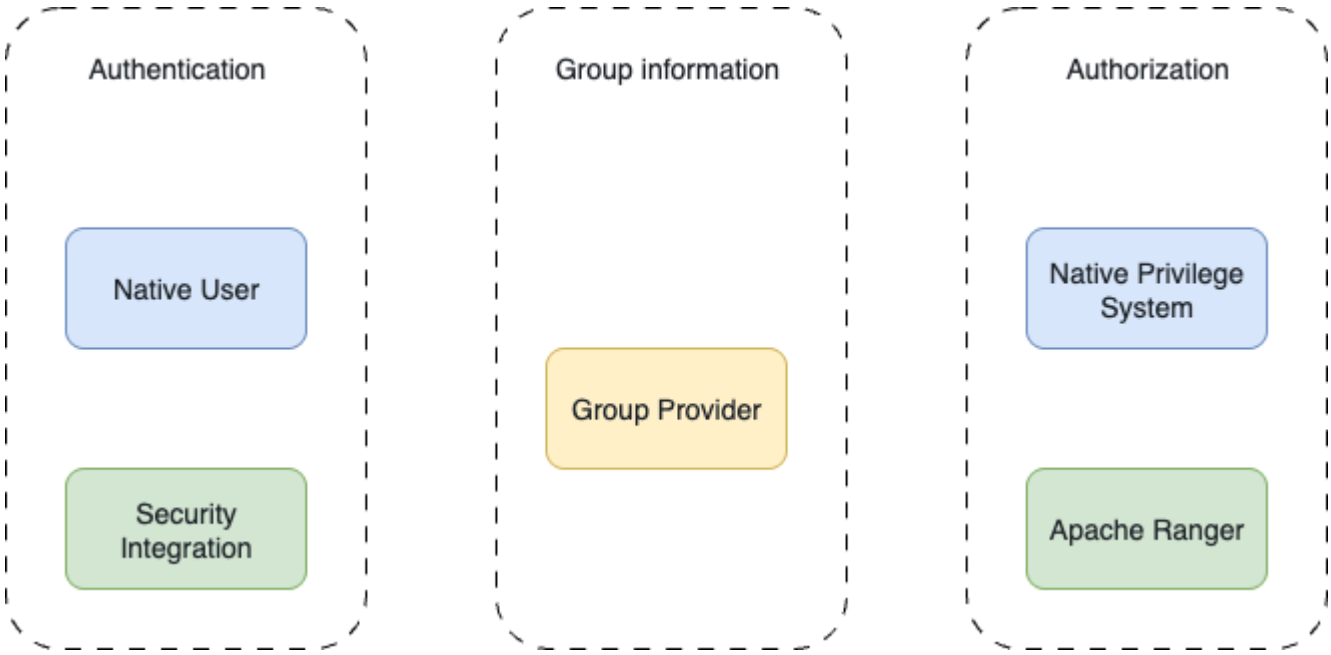
1.6.4 Возможности

Система поддерживает все три уровня контроля доступа:

1. **Аутентификация пользователя** – "Я тот, за кого себя выдаю."
2. **Авторизация входа** – "Мне разрешено получить доступ к этому cluster." (Это зависит от индивидуального или группового членства.)
3. **Авторизация операций** – "Я могу выполнить этот запрос или загрузить этот набор данных." (Авторизация может быть основана на идентичности или групповой принадлежности.)

Начиная с версии 3.5, Selena предоставляет модульную, компонуемую модель для поддержки различных комбинаций компонентов управления идентификацией и доступом.

Соответствие функций



С точки зрения функциональности:

1. **Провайдер аутентификации** – Поддерживаемые протоколы: Native user, LDAP, OIDC и OAuth 2.0.
2. **Провайдер групп** – Поддерживаемые источники: LDAP, операционная система и файловая конфигурация.
3. **Система авторизации** – Поддерживаемые системы: Native RBAC & IBAC и Apache Ranger.

Аутентификация

Сравнение поддерживаемых режимов аутентификации:

Метод	CREATE USER (Нативный пользователь)	CREATE SECURITY INTEGRATION (Временный пользователь на основе сессии)
Описание	Вручную создает пользователей в cluster. Вы можете связать их с внешними системами аутентификации. Пользователь существует явно в cluster.	Определяет интеграцию внешней аутентификации. Cluster не хранит никакой информации о пользователях. Вы можете опционально объединить это с провайдером групп для определения разрешенных пользователей.
Процесс входа	Пользователи должны быть предварительно созданы в cluster. Во время входа пользователь аутентифицируется через Selena или через настроенную внешнюю систему аутентификации (например, LDAP). Только предварительно созданные пользователи могут войти.	При входе Selena аутентифицирует пользователя с использованием внешних систем идентификации. В случае успеха создается временный "временный пользователь" с областью действия сессии внутренне. Этот

Метод	CREATE USER (Нативный пользователь)	CREATE SECURITY INTEGRATION (Временный пользователь на основе сессии)
		пользователь удаляется после окончания сессии.
Процесс авторизации	Поскольку пользователи существуют в cluster, разрешения могут быть назначены заранее с использованием либо нативной системы авторизации, либо Apache Ranger.	Хотя пользователи не сохраняются, вы можете предопределить сопоставления ролей с группами. Когда пользователь входит, система назначает роли на основе их группы, включая RBAC. Apache Ranger также может использоваться параллельно.
Плюсы и минусы, Случаи использования	• **Плюсы**: Полная гибкость—поддерживает как нативные, так и внешние системы авторизации. • **Минусы**: Требует ручных усилий для создания пользователей, что может быть обременительным. • **Рекомендуется для**: Небольших пользовательских баз или случаев, когда cluster обрабатывает контроль доступа.	• **Плюсы**: Легко настраивается; требуется только конфигурация внешней аутентификации и определения разрешенных групп. • **Рекомендуется для**: Идеально для больших пользовательских баз с сопоставлениями роль-группа.

Эти режимы аутентификации могут сосуществовать. Когда пользователь пытается войти:

1. Cluster сначала проверяет, существует ли пользователь как нативный пользователь, и пытается аутентифицировать соответственно.
2. Если пользователь не найден, cluster продолжает по `authentication_chain`, как определено в конфигурации.

Этот гибридный режим обеспечивает как гибкость, так и контроль, подходящий для различных организационных требований.

Вариант 1: Создание нативного пользователя с внешней системой аутентификации

Например, вы можете использовать следующий синтаксис для создания нативного пользователя с LDAP:

```
CREATE USER <username> IDENTIFIED WITH authentication_ldap_simple AS
'uid=tom,ou=company,dc=example,dc=com';
```

Затем вы можете `GRANT` привилегии или роли пользователю или делегировать авторизацию внешним системам, таким как Apache Ranger.

Вариант 2: Использование интеграции безопасности с внешней системой аутентификации

Вы также можете создать интеграцию безопасности, чтобы разрешить доступ вашей внешней службы аутентификации к cluster.

```
CREATE SECURITY INTEGRATION <security_integration_name>
PROPERTIES (
    "type" = "authentication_ldap_simple",
    "authentication_ldap_simple_server_host" = "",
    "authentication_ldap_simple_server_port" = "",
    "authentication_ldap_simple_bind_base_dn" = "",
    "authentication_ldap_simple_user_search_attr" = ""
    "authentication_ldap_simple_bind_root_dn" = "",
    "authentication_ldap_simple_bind_root_pwd" = "",
    "authentication_ldap_simple_ssl_conn_allow_insecure" = "{true |
false}",
    "authentication_ldap_simple_ssl_conn_trust_store_path" = "",
    "authentication_ldap_simple_ssl_conn_trust_store_pwd" = "",
    "comment" = ""
);
```

После этого вам нужно настроить параметр FE `authentication_chain` и включить интеграцию безопасности для вашего cluster.

```
ADMIN SET FRONTEND CONFIG (
    "authentication_chain" = "<security_integration_name>[... ,]"
);
```

Провайдер групп (необязательно, но рекомендуется)

Информация о группах в cluster **отделена** как от систем аутентификации, так и от систем авторизации. Она служит общим слоем, который может быть независимо настроен и затем использован как для контроля входа, так и для контроля доступа.

Как используются группы

- **Этап аутентификации**

При использовании вместе с интеграцией безопасности членство в группе может определять область того, кому разрешено входить. Только пользователи, прошедшие аутентификацию и принадлежащие к указанной группе, получат доступ к cluster.

- **Этап авторизации**

  Членство в группе автоматически учитывается во время авторизации. Если привилегии предоставлены группе, все пользователи в этой группе унаследуют разрешения при проверках доступа.

Примечания по конфигурации

- При настройке провайдера групп вы должны указать:
  - Группы, используемые для определения **кто может войти** (область входа)
  - Группы, используемые для определения **кто может получить доступ к конкретным ресурсам** (авторизация)
- **Важно**: Идентификатор пользователя (например, имя пользователя или ID), возвращаемый провайдером групп, **должен совпадать** с идентификатором, используемым во время аутентификации и авторизации. Несогласованные идентификаторы приведут к сбоям разрешений или входа.

Пример

Следующий пример основан на LDAP.

1. Создайте провайдера групп.

```
-- LDAP Group Provider
CREATE GROUP PROVIDER <group_provider_name>
PROPERTIES (
    "type" = "ldap",
    ldap_info,
    ldap_search_group_arg,
    ldap_search_attr,
    [ldap_cache_attr]
)

ldap_info ::=
    "ldap_conn_url" = "",
    "ldap_bind_root_dn" = "",
    "ldap_bind_root_pwd" = "",
    "ldap_bind_base_dn" = "",
    ["ldap_conn_timeout" = "",]
    ["ldap_conn_read_timeout" = ""]

ldap_search_group_arg ::=
    { "ldap_group_dn" = ""
    | "ldap_group_filter" = "" },
    "ldap_group_identifier_attr" = ""

ldap_search_user_arg ::=
    "ldap_group_member_attr" = "",
    "ldap_user_search_attr" = ""

ldap_cache_arg ::=
    "ldap_cache_refresh_interval" = ""
```

2. Интегрируйте провайдера групп с интеграцией безопасности.

```
ALTER SECURITY INTEGRATION <security_integration_name> SET
(
    "group_provider" = "",
    "permitted_groups" = ""
)
```

3. Интегрируйте провайдера групп с системой авторизации. Вы можете использовать либо нативную авторизацию, либо Apache Ranger.

    - Нативная авторизация:

Роли могут быть назначены группам. При входе пользователи автоматически получают роли на основе членства в группе.

```
GRANT role TO EXTERNAL GROUP <group_name>
```

- Apache Ranger:

  Как только пользователь входит, Selena передает информацию о группах в Ranger для оценки политики.

Авторизация

Selena поддерживает **как внутренние, так и внешние механизмы авторизации**, которые могут использоваться независимо или в комбинации:

- **Внутренняя авторизация**

  Selena предоставляет встроенную систему **RBAC (Role-Based Access Control — контроль доступа на основе ролей)** и **IBAC (Identity-Based Access Control — контроль доступа на основе идентичности)**.

  - **RBAC**: Назначает роли пользователям или группам и предоставляет привилегии этим ролям.
  - **IBAC**: Предоставляет привилегии непосредственно пользователям.
- **Внешняя авторизация**

  Selena интегрируется с **Apache Ranger** для поддержки централизованного и унифицированного управления авторизацией.

Apache Ranger может использоваться либо как интегральное решение само по себе, либо вместе с нативной системой авторизации Selena.

- **Полная авторизация Ranger** Как внутренние таблицы, так и внешние таблицы (например, Hive) авторизуются через Ranger.
  - Разрешения для внутренних таблиц используют плагин Selena для Ranger.
  - Разрешения для внешних таблиц могут управляться либо через плагин Selena, либо через другие плагины внешних сервисов (например, плагин Hive).
- **Гибридная авторизация**

- **Внутренние таблицы**: Авторизуются нативной системой Selena (RBAC/IBAC).
- **Внешние таблицы**: Авторизуются через Ranger. Разрешения для внешних таблиц по-прежнему могут управляться либо с использованием плагина Selena, либо через соответствующий внешний сервис (например, Hive, HDFS).

Эта гибкость позволяет организациям постепенно переходить к централизованной авторизации или поддерживать гибридную модель, которая соответствует их текущей инфраструктуре и политикам безопасности.

1.6.5 Комбинированные решения

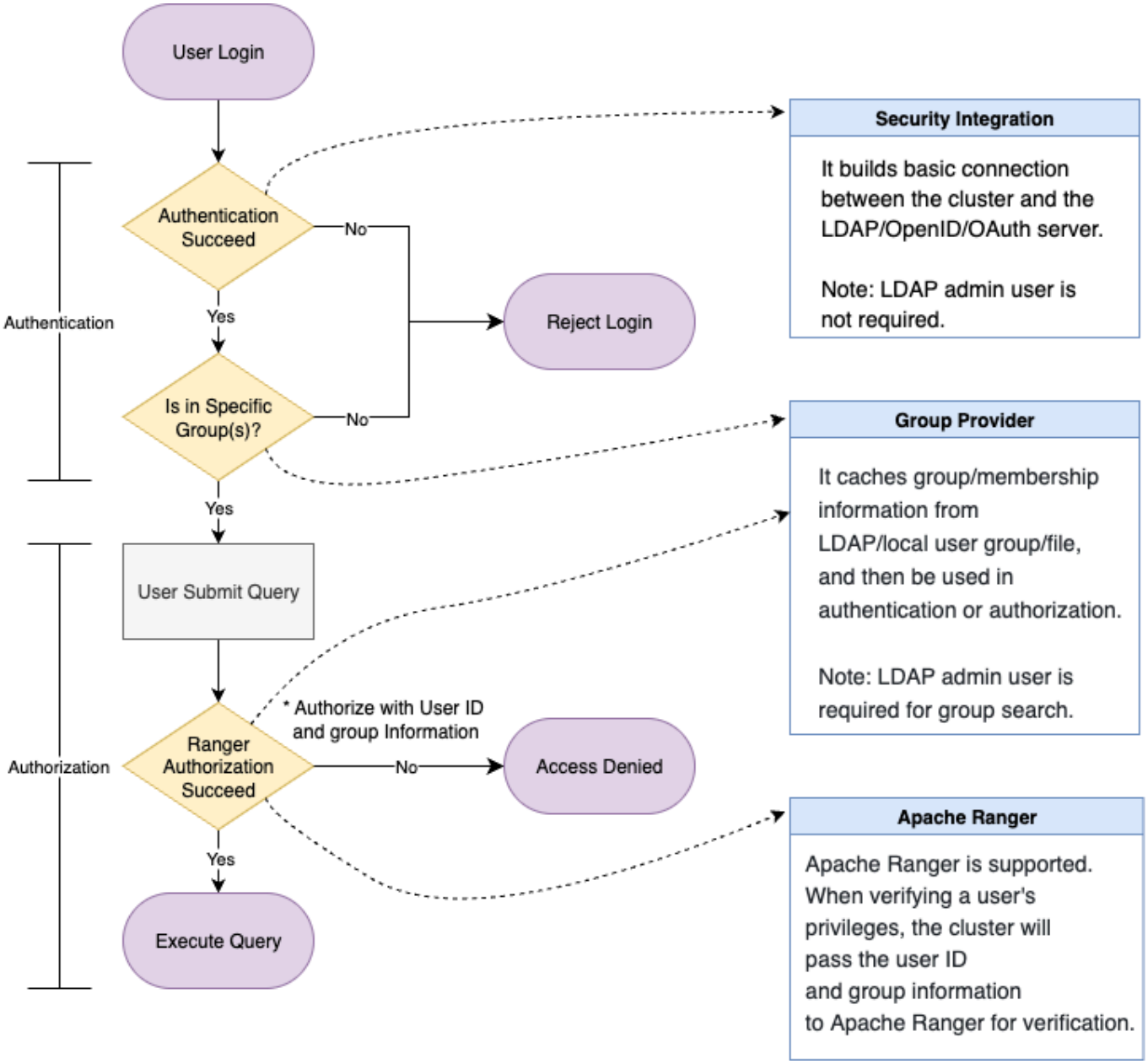
Вы можете выбрать решение в зависимости от того, как вы хотите завершить свой рабочий процесс аутентификации и авторизации.

Решение 1: Внешняя аутентификация + Внешняя авторизация

Вы можете полностью использовать внешние системы аутентификации и авторизации для контроля входа и разрешений доступа к cluster. Общий процесс следующий:

1. Используйте **интеграцию безопасности** для установления соединения с внешней системой аутентификации.
2. Настройте необходимую информацию о группах для аутентификации и авторизации в **провайдере групп**.
3. Определите группу(ы), которым разрешено входить в cluster, в **интеграции безопасности**. Пользователи, принадлежащие к этим группам, получат доступ для входа.
4. Создайте **сервис Selena** в **Apache Ranger** для управления контролем доступа как для внутренних, так и для внешних таблиц. Для внешних таблиц вы также можете повторно использовать существующие сервисы для авторизации.
5. Когда пользователь отправляет запрос, система отправит идентификатор пользователя вместе с его членством в группах (как настроено в провайдере групп) в Ranger для авторизации.
6. Если проверка авторизации пройдена, система продолжит выполнение запроса.

> **ПРИМЕЧАНИЕ**
>
> Вы должны убедиться, что идентификаторы пользователей и имена групп остаются согласованными во всех интегрированных системах на протяжении всего этого процесса.



Решение 2: Внешняя аутентификация (нативный пользователь) + Внутренняя авторизация

Если вы предпочитаете использовать **встроенную систему авторизации**, но все еще полагаетесь на **внешнюю аутентификацию**, вы можете следовать этому подходу:

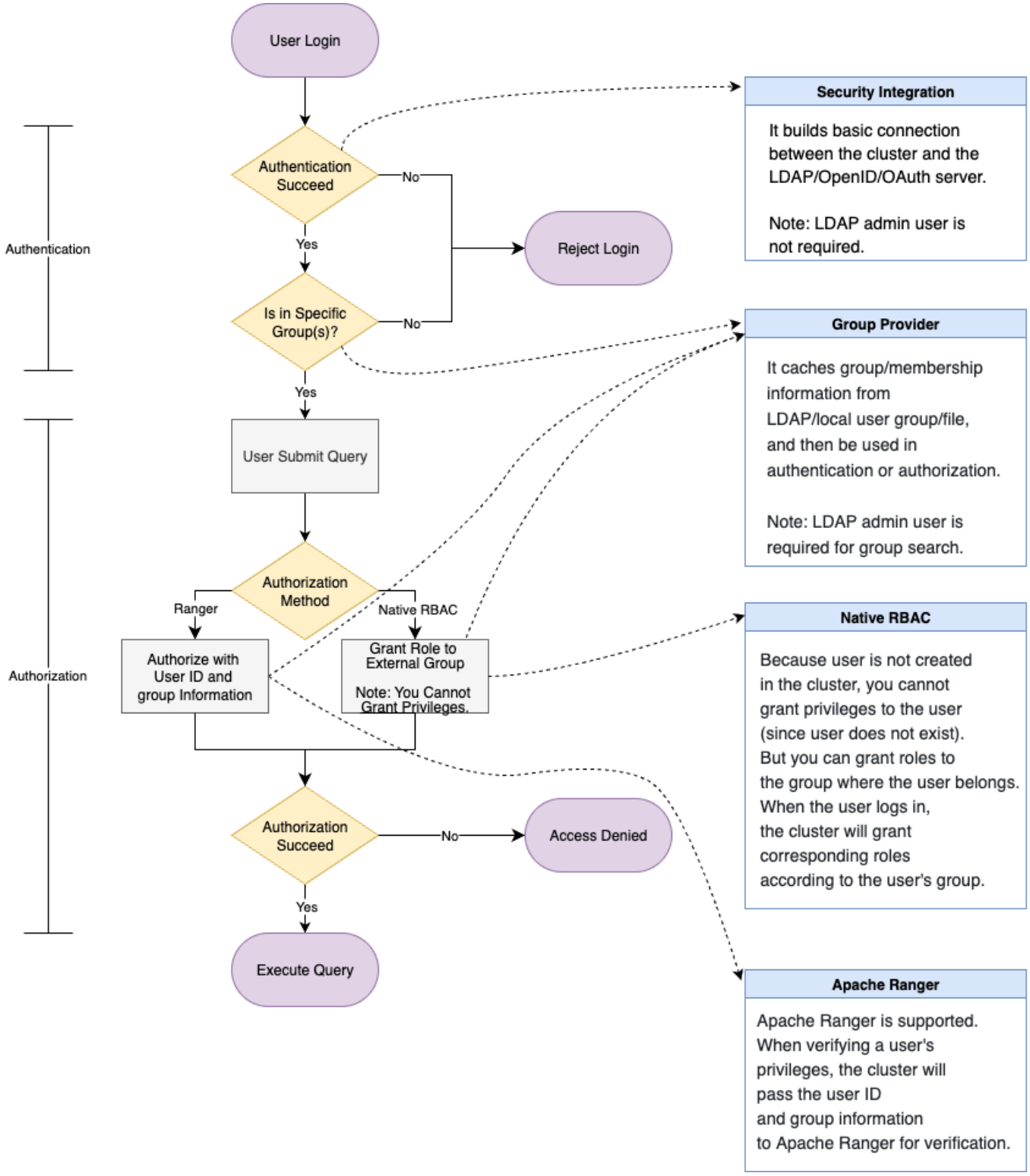
1. **Вручную создайте пользователей** и укажите метод внешней аутентификации для каждого пользователя.
2. После создания пользователя используйте стандартные операторы `GRANT` для назначения ролей или привилегий.
3. После аутентификации пользователь будет авторизован на основе нативной системы разрешений cluster.

> **ПОДСКАЗКА**
>
> Хотя вручную созданные пользователи все еще могут быть интегрированы с **провайдером групп** и **Ranger**, этот подход более сложен и менее автоматизирован по сравнению с использованием **интеграции безопасности**. Поэтому это **не рекомендуемая лучшая практика**.

Решение 3: Внешняя аутентификация (внешняя идентичность) + Внутренняя авторизация

Если вы предпочитаете использовать **встроенную систему авторизации Selena**, но все еще полагаетесь на **внешнюю аутентификацию**, вы можете следовать этому подходу:

1. Используйте **интеграцию безопасности** для установления соединения с внешней системой аутентификации.
2. Настройте необходимую информацию о группах для аутентификации и авторизации в **провайдере групп**.
3. Определите группу(ы), которым разрешено входить в cluster Selena, в **интеграции безопасности**. Пользователи, принадлежащие к этим группам, получат доступ для входа.
4. **Создайте необходимые роли** в Selena и **предоставьте их внешним группам**.
5. Когда пользователь пытается войти, он должен как пройти аутентификацию, так и принадлежать к авторизованной группе. После успешного входа Selena автоматически назначит соответствующие роли на основе членства в группе.
6. Во время выполнения запроса Selena будет применять **внутреннюю авторизацию на основе RBAC** как обычно.
7. Кроме того, вы можете объединить **Ranger** с этим решением. Например, используйте **нативный RBAC Selena** для авторизации внутренних таблиц и используйте **Ranger** для авторизации внешних таблиц. При выполнении авторизации через Ranger, Selena все равно передаст **идентификатор пользователя и соответствующую информацию о группах** в Ranger для контроля доступа.



1.6.6 Смотрите также

- **Аутентификация**
  - Нативная аутентификация
  - Интеграция безопасности
  - LDAP-аутентификация
  - OAuth 2.0 аутентификация

  - Аутентификация JSON Web Token
- **Провайдер групп**
- **Авторизация**
  - Нативная авторизация
  - Плагин Apache Ranger

1.7 Конфигурация групп ресурсов на основе audit log

В Selena **группы ресурсов** (Resource Groups) предоставляют эффективный механизм изоляции ресурсов путем распределения лимитов CPU, памяти и параллелизма на основе классификаторов, таких как идентификатор пользователя и тип запроса. Эта функция необходима для эффективного использования ресурсов в мультитенантных средах.

Традиционная конфигурация групп ресурсов часто основывается на эмпирических оценках. Анализируя исторические данные запросов из таблицы audit log `selena_audit_db__.selena_audit_tbl__`, администраторы могут вместо этого использовать **подход на основе данных** для настройки групп ресурсов. Ключевые метрики, такие как время CPU, потребление памяти и параллелизм запросов, дают объективное представление о фактических характеристиках нагрузки.

Этот подход помогает:

- Предотвратить задержки запросов, вызванные конкуренцией за ресурсы
- Защитить cluster от истощения ресурсов
- Улучшить общую стабильность и предсказуемость

Эта тема содержит пошаговое руководство о том, как вывести подходящие параметры групп ресурсов на основе паттернов нагрузки, наблюдаемых из audit log.

> **ПРИМЕЧАНИЕ**
>
> Это руководство основано на анализе с использованием плагина AuditLoader, который позволяет выполнять запросы к журналам аудита с помощью SQL-операторов непосредственно в вашем cluster. Подробные инструкции по установке плагина см. в разделе AuditLoader.

1.7.1 Распределение ресурсов CPU

Цель

Определить потребление CPU на пользователя и пропорционально распределить ресурсы CPU, используя `cpu_weight` или `exclusive_cpu_cores`.

Анализ

Следующий SQL агрегирует общее время CPU на пользователя (`cpuCostNs`) за последние 30 дней, преобразует его в секунды и вычисляет процент от общего использования CPU.

```
SELECT
    user,
    SUM(cpuCostNs) / 1e9 AS total_cpu_seconds,                    -- Запрос общего времени CPU.
    (
        SUM(cpuCostNs) /
        (
            SELECT SUM(cpuCostNs)
            FROM selena_audit_db__.selena_audit_tbl__
            WHERE state IN ('EOF','OK')
              AND timestamp >= DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)
        )
    ) * 100 AS cpu_usage_percentage                               -- Вычисление процента от общего использования CPU на пользователя.
FROM selena_audit_db__.selena_audit_tbl__
WHERE state IN ('EOF','OK')                                       -- Включить только завершенные запросы.
  AND timestamp >= DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)      -- Запрос данных за последние 30 дней.
GROUP BY user
ORDER BY total_cpu_seconds DESC
LIMIT 20;                                                         -- Список топ-20 пользователей с наибольшим потреблением ресурсов CPU.
```

Лучшие практики

Предположим фиксированное количество ядер CPU на BE (например, 64 ядра). Если пользователь составляет 16% (`cpu_usage_percentage`) от общего времени CPU, выделение приблизительно `64 × 16% ≈ 11 ядер` является разумным.

Вы можете настроить лимиты CPU для группы ресурсов следующим образом:

- `exclusive_cpu_cores`:
  - Его значение не должно превышать общее количество ядер на одном BE.
  - Сумма `exclusive_cpu_cores` всех групп ресурсов не должна превышать общее количество ядер на одном BE.
- `cpu_weight`:
  - Применяется только к группам ресурсов с **мягкой изоляцией**.
  - Определяет относительную долю CPU среди конкурирующих запросов на оставшихся ядрах.
  - **Не** отображается напрямую на фиксированное количество ядер CPU.

1.7.2 Управление памятью

Цель

Определить пользователей с интенсивным использованием памяти и установить соответствующие лимиты памяти и circuit breaker.

Анализ

Следующий SQL вычисляет максимальное использование памяти на пользователя (`memCostBytes`) для одного запроса за последние 30 дней.

```
SELECT
    user,
    MAX(memCostBytes) / (1024 * 1024) AS max_mem_mb            --
Максимальное использование памяти (в MB) на запрос.
FROM selena_audit_db__.selena_audit_tbl__
WHERE state IN ('EOF','OK')                                    -- Включить
только завершенные запросы.
  AND timestamp >= DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)   -- Запрос
данных за последние 30 дней.
GROUP BY user
ORDER BY max_mem_mb DESC
LIMIT 20;                                                      -- Список
топ-20 пользователей с наибольшим потреблением памяти.
```

Лучшие практики

`max_mem_mb` представляет **общее использование памяти на всех BE**. Вы можете вычислить приблизительное использование памяти на один BE как: `max_mem_mb / количество_BE`.

Вы можете настроить лимиты памяти для группы ресурсов следующим образом:

- `big_query_mem_limit`:
  - Защищает cluster от аномально больших запросов.
  - Вы можете установить его на относительно высокий порог, чтобы избежать ложного прерывания запросов.
- `mem_limit`:
  - В большинстве случаев устанавливайте его на высокое значение (например, `0.9`).

1.7.3 Контроль параллелизма

Цель

Определить пиковый параллелизм запросов на пользователя и установить соответствующие значения `concurrency_limit`.

Анализ

Следующий SQL анализирует поминутный параллелизм запросов за последние 30 дней и извлекает максимальный наблюдаемый параллелизм на пользователя.

```
WITH UserConcurrency AS (
    SELECT
        user,
        DATE_FORMAT(timestamp, '%Y-%m-%d %H:%i') AS minute_bucket,
        COUNT(*) AS query_concurrency
    FROM selena_audit_db__.selena_audit_tbl__
    WHERE state IN ('EOF', 'OK')                              -- Включить
только завершенные запросы.
      AND timestamp >= DATE_SUB(NOW(), INTERVAL 30 DAY)       -- Запрос
данных за последние 30 дней.
      AND LOWER(stmt) LIKE '%select%'                         -- Включить
только SELECT-запросы.
    GROUP BY user, minute_bucket
    HAVING query_concurrency > 1                              -- Исключить
сценарии, где параллелизм менее одного запроса в минуту.
)
SELECT
    user,
    minute_bucket,
    query_concurrency / 60.0 AS query_concurrency_per_second  -- Запрос
посекундного параллелизма.
FROM (
    SELECT
        user,
        minute_bucket,
        query_concurrency,
        ROW_NUMBER() OVER (
            PARTITION BY user
            ORDER BY query_concurrency DESC
        ) AS rn
    FROM UserConcurrency
) ranked
WHERE rn = 1                                                  -- Сохранить
наивысшую запись для каждого пользователя.
ORDER BY query_concurrency_per_second DESC
LIMIT 50;                                                     -- Список
топ-50 пользователей с наибольшим параллелизмом.
```

Лучшие практики

Вышеуказанный анализ выполняется с **поминутной детализацией**. Фактический посекундный параллелизм может быть выше.

Вы можете настроить лимиты параллелизма для группы ресурсов следующим образом:

- `concurrency_limit`
  - Установите его приблизительно на **1.5× от наблюдаемого пика**, чтобы обеспечить запас.
  - Для пользователей с экстремальными всплесками параллелизма вы можете дополнительно включить **очереди запросов** (Query Queues), чтобы сгладить пиковую нагрузку и защитить стабильность cluster.

1.7.4 Изоляция ресурсов для асинхронных материализованных представлений

Цель

Предотвратить влияние операций обновления асинхронных материализованных представлений на интерактивные запросы.

Анализ

Следующий SQL идентифицирует операции обновления материализованных представлений с интенсивным использованием памяти, обычно характеризуемые операторами `INSERT OVERWRITE`.

```
SELECT
    user,
    MAX(memCostBytes) / (1024 * 1024) AS max_mem_mb             --
Максимальное использование памяти (в MB) на запрос.
FROM selena_audit_db__.selena_audit_tbl__
WHERE state IN ('EOF','OK')                                     -- Включить
только завершенные запросы.
  AND timestamp >= DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)    -- Запрос
данных за последние 30 дней.
  AND LOWER(stmt) LIKE '%insert overwrite%'                     -- Включить
только операции обновления материализованных представлений.
GROUP BY user
ORDER BY max_mem_mb DESC
LIMIT 20;                                                       -- Список
топ-20 пользователей с наибольшим потреблением памяти.
```

Лучшие практики

Selena предоставляет системную группу ресурсов (`default_mv_wg`) для задач обновления материализованных представлений по умолчанию. Однако настройка выделенной группы

ресурсов для задач обновления материализованных представлений строго рекомендуется для обеспечения строгой изоляции и предотвращения деградации производительности запросов переднего плана из-за операций обновления материализованных представлений.

Инструкции по настройке лимитов групп ресурсов см. в разделах Лучшие практики для распределения ресурсов CPU и Лучшие практики для управления памятью.

Следующий пример только предоставляет руководство по созданию и назначению выделенной группы ресурсов для задач обновления материализованных представлений.

1. Создайте выделенную группу ресурсов для обновления материализованных представлений:

```
CREATE RESOURCE GROUP rg_mv
TO (
    user = 'mv_user',
    query_type IN ('insert', 'select')
)
WITH (
    'cpu_weight' = '32',
    'mem_limit' = '0.9',
    'concurrency_limit' = '10',
    'spill_mem_limit_threshold' = '0.5'
);
```

2. Назначьте группу ресурсов материализованному представлению.

   - Назначение при создании материализованного представления:

```
CREATE MATERIALIZED VIEW mv_example
REFRESH ASYNC
PROPERTIES (
    'resource_group' = 'rg_mv'
)
AS
SELECT * FROM example_table;
```

   - Назначение существующему материализованному представлению:

```
ALTER MATERIALIZED VIEW mv_example SET ("resource_group" = "rg_mv");
```

1.7.5 См. также

- Группы ресурсов
- Очереди запросов

1.8 Query Tuning

1.8.1 Введение в оптимизацию запросов

Оптимизация запросов необходима для достижения высокой производительности и надежности в Selena. Этот раздел объединяет практические руководства, справочные материалы и практические рецепты, которые помогут вам анализировать, диагностировать и оптимизировать производительность запросов на каждом этапе — от написания SQL до интерпретации деталей выполнения.

Эффективная оптимизация запросов в Selena обычно следует процессу сверху вниз:

1. **Определите проблему**
    - Обнаружьте медленные запросы, высокое потребление ресурсов или неожиданные результаты.
    - В Selena используйте встроенные инструменты мониторинга, историю запросов и журналы аудита для быстрого выявления проблемных запросов или необычных паттернов.
    - См.: **Рецепты оптимизации запросов** для диагностики по симптомам и **Обзор профиля запросов** для доступа к истории запросов и профилям.
2. **Соберите и проанализируйте информацию о выполнении**
    - Получите план запроса с помощью `EXPLAIN` или `EXPLAIN ANALYZE`.
    - Включите и просмотрите Query Profile для сбора подробных метрик выполнения.
    - См.: **Обзор плана запроса** для понимания планов запросов, **Explain Analyze и текстовый анализ профиля** для пошагового анализа и **Обзор профиля запросов** для включения и интерпретации профилей.
3. **Определите основную причину**

- Определите, какой этап или оператор потребляет больше всего времени или ресурсов.
- Проверьте на наличие распространенных проблем: неоптимальный порядок соединения, отсутствие индексов, проблемы с распределением данных или неэффективные SQL-паттерны.
- См.: **Метрики профиля запросов** для глоссария метрик и операторов, и **Рецепты оптимизации запросов** для анализа основных причин.

4. **Примените стратегии оптимизации**

   - Переписывание SQL: переписать или оптимизировать SQL-запрос (например, добавить фильтры, избегать SELECT *).
   - Оптимизация схемы: добавить индексы, изменить типы таблиц, секционирование, кластеризацию.
   - Оптимизация плана запроса: использовать hint или переменные для управления оптимизатором при необходимости.
   - Оптимизация выполнения: настроить переменные сессии для конкретных рабочих нагрузок.
   - См.: **Рецепты оптимизации схемы** для оптимизации на уровне схемы, **Query Hint** для hint оптимизатора и **Рецепты оптимизации запросов** для оптимизации плана и выполнения.

5. **Проверьте и повторите**

   - Перезапустите запрос и сравните производительность до и после изменений.
   - Просмотрите новый план запроса и профиль, чтобы убедиться в улучшениях.
   - При необходимости повторите процесс для дальнейшей оптимизации.

Независимо от того, являетесь ли вы администратором баз данных, разработчиком или инженером по работе с данными, эти ресурсы помогут вам:

- Диагностировать и решать проблемы с медленными или ресурсоемкими запросами
- Понимать выбор оптимизатора и детали выполнения
- Применять лучшие практики и продвинутые стратегии оптимизации

Начните с обзора, при необходимости изучите справочные материалы и используйте рецепты и советы для решения реальных проблем производительности в Selena.

1.8.2 План запроса

Оптимизация производительности запросов — распространенная задача в аналитических системах. Медленные запросы могут ухудшить пользовательский опыт и общую производительность cluster. В Selena понимание и интерпретация планов запросов и профилей запросов является основой для диагностики и улучшения медленных запросов. Эти инструменты помогают вам:

- Определить узкие места и дорогостоящие операции
- Обнаружить неоптимальные стратегии соединения или отсутствующие индексы
- Понять, как данные фильтруются, агрегируются и перемещаются
- Устранять неполадки и оптимизировать использование ресурсов

План запроса — это подробная дорожная карта, созданная Selena FE, которая описывает, как будет выполнен ваш SQL-оператор. Он разбивает запрос на серию операций — таких как сканирование, соединение, агрегация и сортировка — и определяет наиболее эффективный способ их выполнения.

Selena предоставляет несколько способов просмотра плана запроса:

1. **Оператор EXPLAIN**: Используйте `EXPLAIN` для отображения логического или физического плана выполнения запроса. Вы можете добавить опции для управления выводом:

   - `EXPLAIN LOGICAL <query>`: Показывает упрощенный план.
   - `EXPLAIN <query>`: Показывает базовый физический план
   - `EXPLAIN VERBOSE <query>`: Показывает физический план с подробной информацией.
   - `EXPLAIN COSTS <query>`: Включает оценочные затраты для каждой операции, что используется для диагностики проблем со статистикой
2. **EXPLAIN ANALYZE**: Используйте `EXPLAIN ANALYZE <query>` для выполнения запроса и отображения фактического плана выполнения вместе с реальной статистикой времени выполнения. См. документацию по Explain Anlayze для подробностей.

   Пример:

   ```
   EXPLAIN ANALYZE SELECT * FROM sales_orders WHERE amount > 1000;
   ```

3. **Query Profile**: После выполнения запроса вы можете просмотреть его подробный профиль выполнения, который включает время, использование ресурсов и статистику на уровне операторов. См. документацию по Query Profile для получения информации о том, как получить доступ и интерпретировать эту информацию.

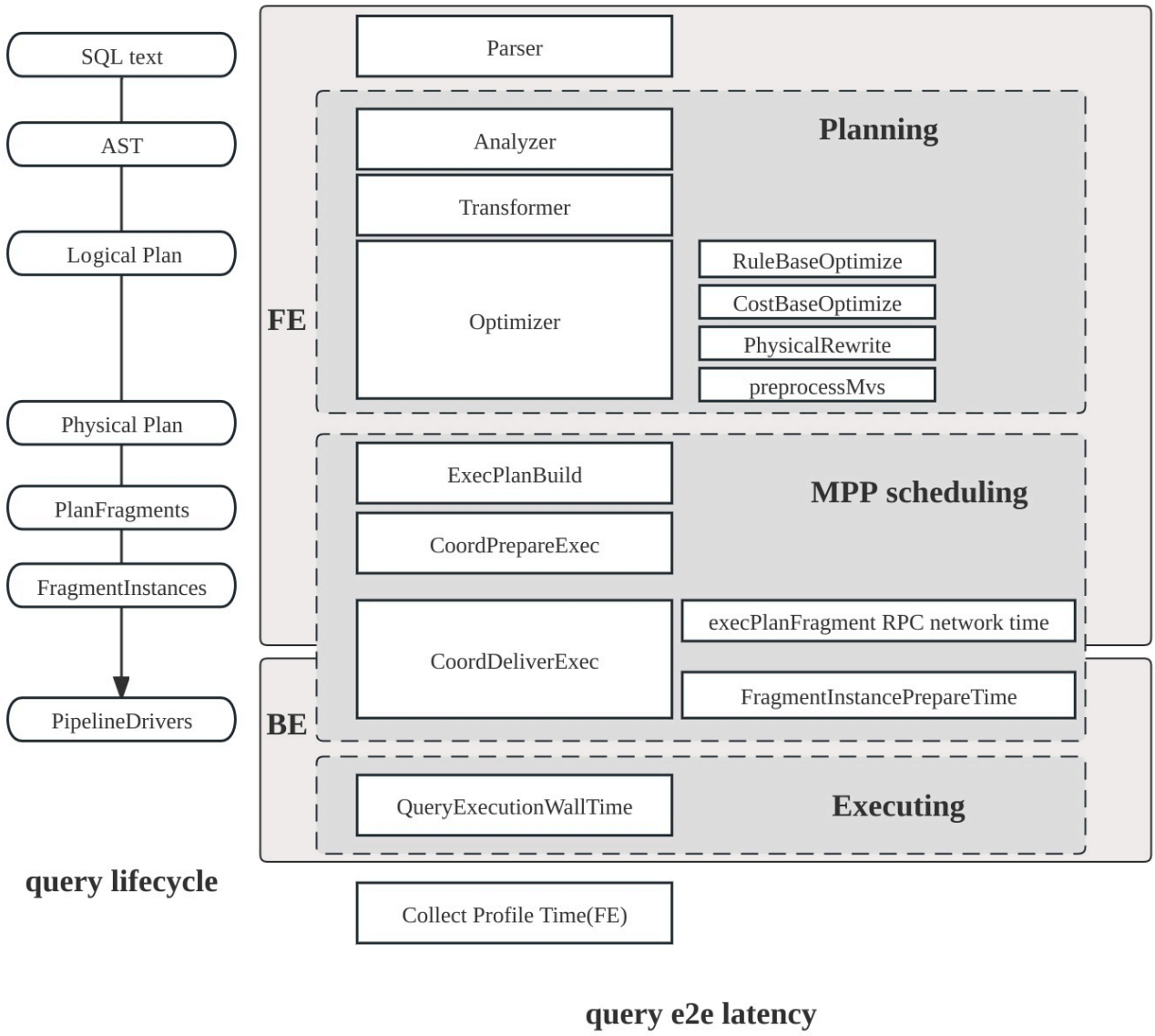
    - **SQL-команды**: `SHOW PROFILELIST` и `ANALYZE PROFILE FOR <query_id>`: можно использовать для получения профиля выполнения для конкретного запроса.
    - **HTTP-сервис FE**: Доступ к профилям запросов через веб-интерфейс Selena FE, перейдя в раздел **Query** или **Profile**, где вы можете искать и просматривать детали выполнения запросов.
    - **Управляемая версия**: В облачных или управляемых развертываниях используйте предоставленную веб-консоль или панель мониторинга для просмотра планов запросов и профилей, часто с расширенными возможностями визуализации и фильтрации.

Обычно план запроса используется для диагностики проблем, связанных с тем, как запрос планируется и оптимизируется, в то время как профиль запроса помогает выявить проблемы производительности во время выполнения запроса. В следующих разделах мы рассмотрим ключевые концепции выполнения запросов и пройдем через конкретный пример анализа плана запроса.

Поток выполнения запроса

Жизненный цикл запроса в Selena состоит из трех основных фаз:

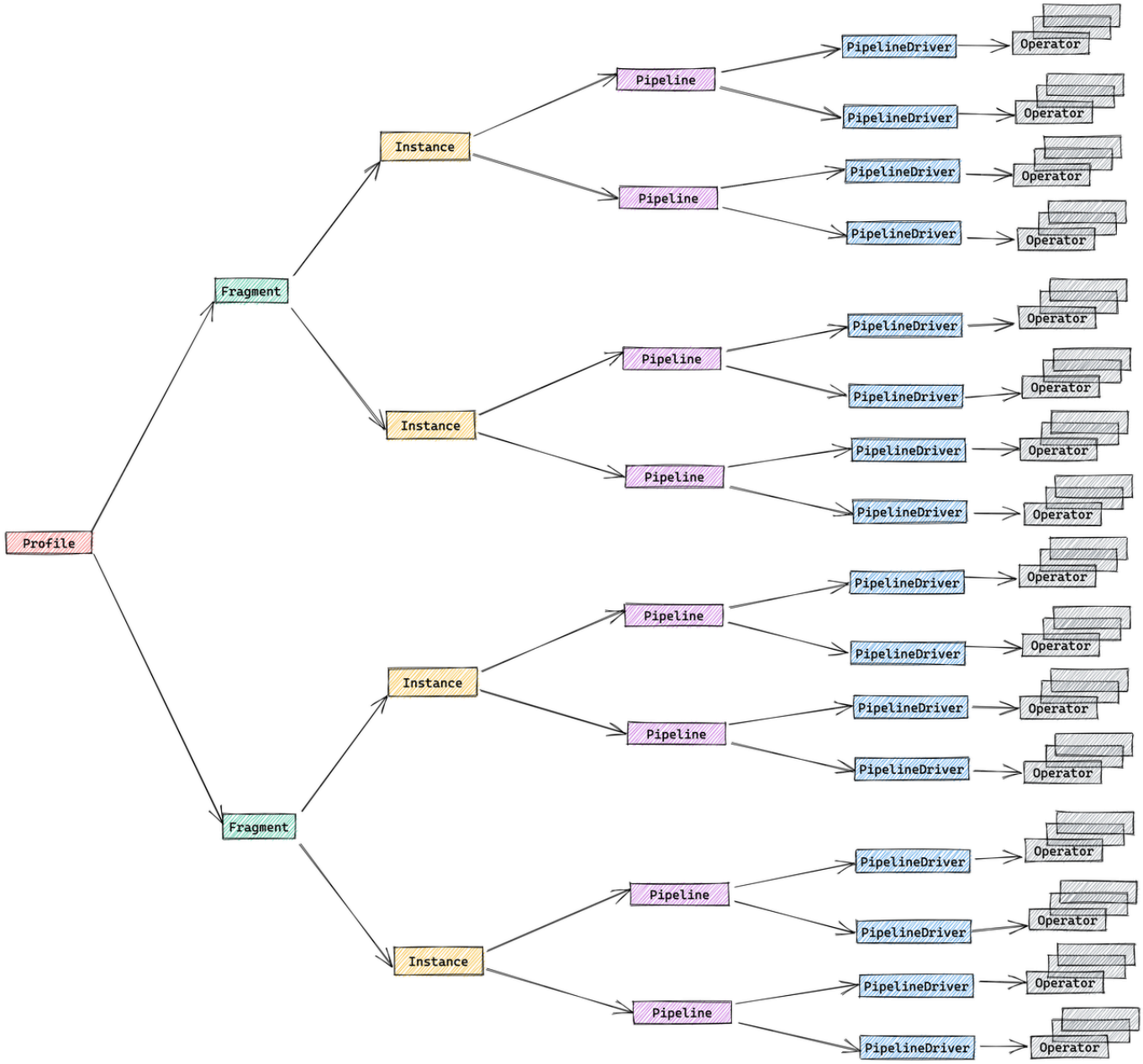
1. **Планирование**: Запрос проходит через парсинг, анализ и оптимизацию, завершаясь генерацией плана запроса.
2. **Планирование выполнения**: Планировщик и координатор распределяют план по всем участвующим backend-узлам.
3. **Выполнение**: План выполняется с использованием pipeline-движка выполнения.



Структура плана

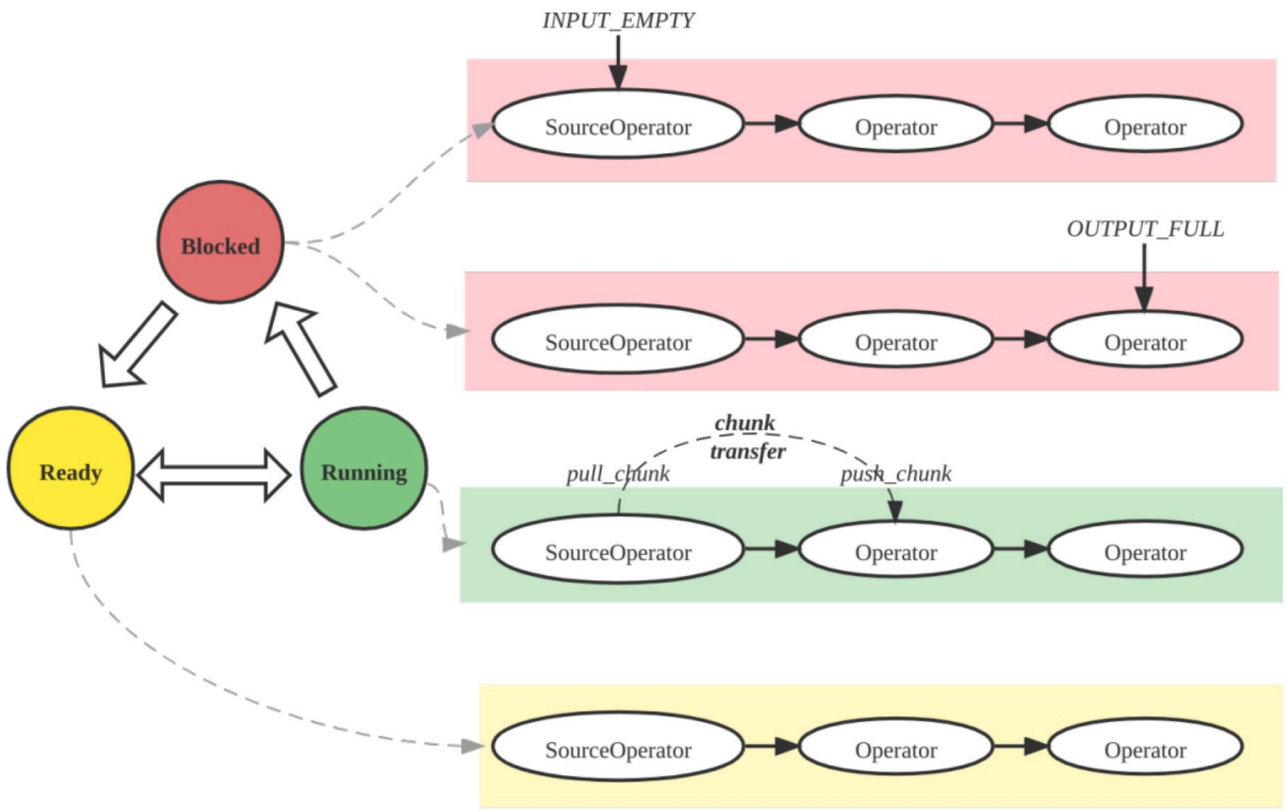
План Selena имеет иерархическую структуру:

- **Fragment**: Срез работы верхнего уровня; каждый fragment порождает множество **FragmentInstances**, которые выполняются на разных backend-узлах.
- **Pipeline**: Внутри instance pipeline связывает операторы вместе; несколько **PipelineDrivers** выполняют один и тот же pipeline параллельно на отдельных ядрах CPU.
- **Operator**: Атомарный шаг — сканирование, соединение, агрегация — который фактически обрабатывает данные.



Pipeline-движок выполнения

Pipeline Engine выполняет план запроса параллельно и эффективно, обрабатывая сложные планы и большие объемы данных для высокой производительности и масштабируемости.



Стратегия объединения метрик

По умолчанию Selena объединяет уровни FragmentInstance и PipelineDriver для уменьшения объема профиля, в результате чего получается упрощенная трехуровневая структура:

- Fragment
- Pipeline
- Operator

Вы можете управлять этим поведением объединения через переменную сессии `pipeline_profile_level`.

Пример

Как читать план запроса и профиль

1. **Понимание структуры**: Планы запросов разделены на фрагменты, каждый из которых представляет этап выполнения. Читайте снизу вверх: сначала узлы сканирования, затем соединения, агрегации и, наконец, результат.

2. **Общий анализ**:
   - Проверьте общее время выполнения, использование памяти и соотношение CPU/wall time.
   - Найдите медленные операторы, отсортировав по времени оператора.
   - Убедитесь, что фильтры проталкиваются вниз, где это возможно.
   - Ищите перекос данных (неравномерное время операторов или количество строк).
   - Следите за высокой памятью или переполнением на диск; при необходимости отрегулируйте порядок соединения или используйте rollup-представления.
   - Используйте материализованные представления и query hint (`BROADCAST`, `SHUFFLE`, `COLOCATE`) для оптимизации по мере необходимости.
3. **Операции сканирования**: Ищите `OlapScanNode` или аналогичные. Обратите внимание, какие таблицы сканируются, какие фильтры применяются и используются ли предварительная агрегация или материализованные представления.

4. **Операции соединения**: Определите типы соединений (`HASH JOIN`, `BROADCAST`, `SHUFFLE`, `COLOCATE`, `BUCKET SHUFFLE`). Метод соединения влияет на производительность:
   - **Broadcast**: Маленькая таблица отправляется на все узлы; хорошо для маленьких таблиц.
   - **Shuffle**: Строки секционируются и перемешиваются; хорошо для больших таблиц.
   - **Colocate**: Таблицы секционированы одинаково; позволяет выполнять локальные соединения.
   - **Bucket Shuffle**: Перемешивается только одна таблица для сокращения сетевых затрат.
5. **Агрегация и сортировка**: Ищите `AGGREGATE`, `TOP-N` или `ORDER BY`. Эти операции могут быть дорогостоящими при больших данных или данных с высокой мощностью.

6. **Перемещение данных**: Узлы `EXCHANGE` показывают передачу данных между фрагментами или узлами. Слишком большое перемещение данных может снизить производительность.

7. **Проталкивание предикатов**: Фильтры, примененные рано (при сканировании), сокращают нижестоящие данные. Проверьте `PREDICATES` или `PushdownPredicates`, чтобы увидеть, какие фильтры проталкиваются вниз.

Пример плана запроса

ПОДСКАЗКА

Это запрос 96 из бенчмарка TPC-DS.

```
explain logical
select  count(*)
from store_sales
    ,household_demographics
    ,time_dim
    , store
where ss_sold_time_sk = time_dim.t_time_sk
    and ss_hdemo_sk = household_demographics.hd_demo_sk
    and ss_store_sk = s_store_sk
    and time_dim.t_hour = 8
    and time_dim.t_minute >= 30
    and household_demographics.hd_dep_count = 5
    and store.s_store_name = 'ese'
order by count(*) limit 100;
```

Вывод представляет собой иерархический план, показывающий, как Selena будет выполнять запрос. План структурирован как дерево операторов, читаемое снизу вверх. Логический план показывает последовательность операций с оценкой затрат:

```
- Output => [69:count]
    - TOP-100(FINAL)[69: count ASC NULLS FIRST]
            Estimates: {row: 1, cpu: 8.00, memory: 8.00, network: 8.00,
cost: 68669801.20}
        - TOP-100(PARTIAL)[69: count ASC NULLS FIRST]
                Estimates: {row: 1, cpu: 8.00, memory: 8.00, network: 8.00,
cost: 68669769.20}
            - AGGREGATE(GLOBAL) []
                    Estimates: {row: 1, cpu: 8.00, memory: 8.00, network:
0.00, cost: 68669737.20}
                    69:count := count(69:count)
                - EXCHANGE(GATHER)
                        Estimates: {row: 1, cpu: 8.00, memory: 0.00,
network: 8.00, cost: 68669717.20}
                    - AGGREGATE(LOCAL) []
                            Estimates: {row: 1, cpu: 3141.35, memory: 0.80,
network: 0.00, cost: 68669701.20}
                            69:count := count()
                        - HASH/INNER JOIN [9:ss_store_sk = 40:s_store_sk]
=> [71:auto_fill_col]
                                Estimates: {row: 3490, cpu: 111184.52,
memory: 8.80, network: 0.00, cost: 68668128.93}
                                71:auto_fill_col := 1
                            - HASH/INNER JOIN [7:ss_hdemo_sk =
25:hd_demo_sk] => [9:ss_store_sk]
                                    Estimates: {row: 19940, cpu:
1841177.20, memory: 2880.00, network: 0.00, cost: 68612474.92}
                                - HASH/INNER JOIN [4:ss_sold_time_sk =
30:t_time_sk] => [7:ss_hdemo_sk, 9:ss_store_sk]
                                        Estimates: {row: 199876, cpu:
69221191.15, memory: 7077.97, network: 0.00, cost: 67671726.32}
                                    - SCAN [store_sales] =>
[4:ss_sold_time_sk, 7:ss_hdemo_sk, 9:ss_store_sk]
                                            Estimates: {row: 5501341, cpu:
66016092.00, memory: 0.00, network: 0.00, cost: 33008046.00}
                                            partitionRatio: 1/1,
tabletRatio: 192/192
                                            predicate: 7:ss_hdemo_sk IS NOT
NULL
                                    - EXCHANGE(BROADCAST)
                                            Estimates: {row: 1769, cpu:
7077.97, memory: 7077.97, network: 7077.97, cost: 38928.81}
                                        - SCAN [time_dim] => [30:t_time_sk]
                                                Estimates: {row: 1769, cpu:
21233.90, memory: 0.00, network: 0.00, cost: 10616.95}
```

```
                                                    partitionRatio: 1/1,
tabletRatio: 5/5
                                                    predicate: 33:t_hour = 8
AND 34:t_minute >= 30
                                - EXCHANGE(BROADCAST)
                                        Estimates: {row: 720, cpu: 2880.00,
memory: 2880.00, network: 2880.00, cost: 14400.00}
                                    - SCAN [household_demographics] =>
[25:hd_demo_sk]
                                            Estimates: {row: 720, cpu:
5760.00, memory: 0.00, network: 0.00, cost: 2880.00}
                                            partitionRatio: 1/1,
tabletRatio: 1/1
                                            predicate: 28:hd_dep_count = 5
                            - EXCHANGE(BROADCAST)
                                    Estimates: {row: 2, cpu: 8.80, memory:
8.80, network: 8.80, cost: 44.15}
                                - SCAN [store] => [40:s_store_sk]
                                        Estimates: {row: 2, cpu: 17.90,
memory: 0.00, network: 0.00, cost: 8.95}
                                        partitionRatio: 1/1, tabletRatio:
1/1
                                        predicate: 45:s_store_name = 'ese'
```

Чтение плана снизу вверх

План запроса следует читать снизу (листовые узлы) вверх к верху (корневой узел), следуя потоку данных:

1. **Операции сканирования (нижний уровень)**: Операторы `SCAN` внизу читают данные из базовых таблиц:
   - `SCAN [store_sales]` читает основную таблицу фактов с предикатом `ss_hdemo_sk IS NOT NULL`
   - `SCAN [time_dim]` читает таблицу измерения времени с предикатами `t_hour = 8 AND t_minute >= 30`
   - `SCAN [household_demographics]` читает таблицу демографии с предикатом `hd_dep_count = 5`
   - `SCAN [store]` читает таблицу магазинов с предикатом `s_store_name = 'ese'`

   Каждая операция сканирования показывает:

- **Estimates**: Оценка количества строк, CPU, памяти, сети и затрат
- **Соотношения секций и tablet**: Сколько секций/tablet сканируется (например, `partitionRatio: 1/1, tabletRatio: 192/192`)
- **Predicates**: Условия запроса, которые проталкиваются до уровня сканирования, сокращая объем считываемых данных

2. **Обмен данными (Broadcast)**: Операции `EXCHANGE(BROADCAST)` распределяют меньшие таблицы измерений на все узлы, обрабатывающие большую таблицу фактов. Это эффективно, когда таблицы измерений малы по сравнению с таблицей фактов, как видно с `time_dim`, `household_demographics` и `store`, которые транслируются.

3. **Операции соединения (средний уровень)**: Данные поступают вверх через операции `HASH/INNER JOIN`:

   - Сначала `store_sales` соединяется с `time_dim` по `ss_sold_time_sk = t_time_sk`
   - Затем результат соединяется с `household_demographics` по `ss_hdemo_sk = hd_demo_sk`
   - Наконец, результат соединяется с `store` по `ss_store_sk = s_store_sk`

   Каждое соединение показывает условие соединения и оценки для результирующего количества строк и использования ресурсов.

4. **Агрегация (верхний уровень)**:

   - `AGGREGATE(LOCAL)` выполняет локальную агрегацию на каждом узле, вычисляя `count()`
   - `EXCHANGE(GATHER)` собирает результаты со всех узлов
   - `AGGREGATE(GLOBAL)` объединяет локальные результаты в окончательный подсчет

5. **Финальные операции (верхний уровень)**:

   - Операции `TOP-100(PARTIAL)` и `TOP-100(FINAL)` обрабатывают предложение `ORDER BY count(*) LIMIT 100`, выбирая верхние 100 результатов после упорядочивания

Логический план предоставляет оценки затрат для каждой операции, помогая вам понять, где запрос тратит большую часть своих ресурсов. Фактический физический план выполнения (из `EXPLAIN` или `EXPLAIN VERBOSE`) включает дополнительные детали о том, как операции распределяются по узлам и выполняются параллельно.

1.8.3 Обзор Query Profile

Введение

Query Profile записывает информацию о выполнении на всех рабочих узлах, участвующих в запросе, помогая быстро идентифицировать узкие места, влияющие на производительность запросов. Это мощный инструмент для диагностики и настройки производительности запросов в Selena.

> Начиная с версии v1.5.2, Selena поддерживает предоставление Query Profile для загрузки данных с помощью INSERT INTO FILES() и Broker Load. Подробности о задействованных метриках см. в разделе OlapTableSink Operator.

Как включить Query Profile

Включение Query Profile

Вы можете включить Query Profile, установив переменную `enable_profile` в значение `true`:

```
SET enable_profile = true;
SET GLOBAL enable_profile = true;
```

Query Profile для медленных запросов

Не рекомендуется включать Query Profile глобально в production на длительные периоды, так как это может создать дополнительную нагрузку на систему. Чтобы захватывать и анализировать только медленные запросы, установите переменную `big_query_profile_threshold` на значение времени больше `0s`. Например, установка значения `30s` означает, что только запросы, превышающие 30 секунд, будут запускать Query Profile.

```
-- 30 секунд
SET global big_query_profile_threshold = '30s';

-- 500 миллисекунд
SET global big_query_profile_threshold = '500ms';

-- 60 минут
SET global big_query_profile_threshold = '60m';
```

Runtime Query Profile

Для долго выполняющихся запросов может быть сложно определить прогресс или обнаружить проблемы до завершения. Функция Runtime Query Profile (v1.5.2+) собирает и отправляет данные Query Profile через фиксированные интервалы во время выполнения, обеспечивая информацию в реальном времени о прогрессе запроса и узких местах.

Когда Query Profile включен, Runtime Query Profile автоматически активируется с интервалом отчетности по умолчанию в 10 секунд. Настройте интервал с помощью `runtime_profile_report_interval`:

```
SET runtime_profile_report_interval = 30;
```

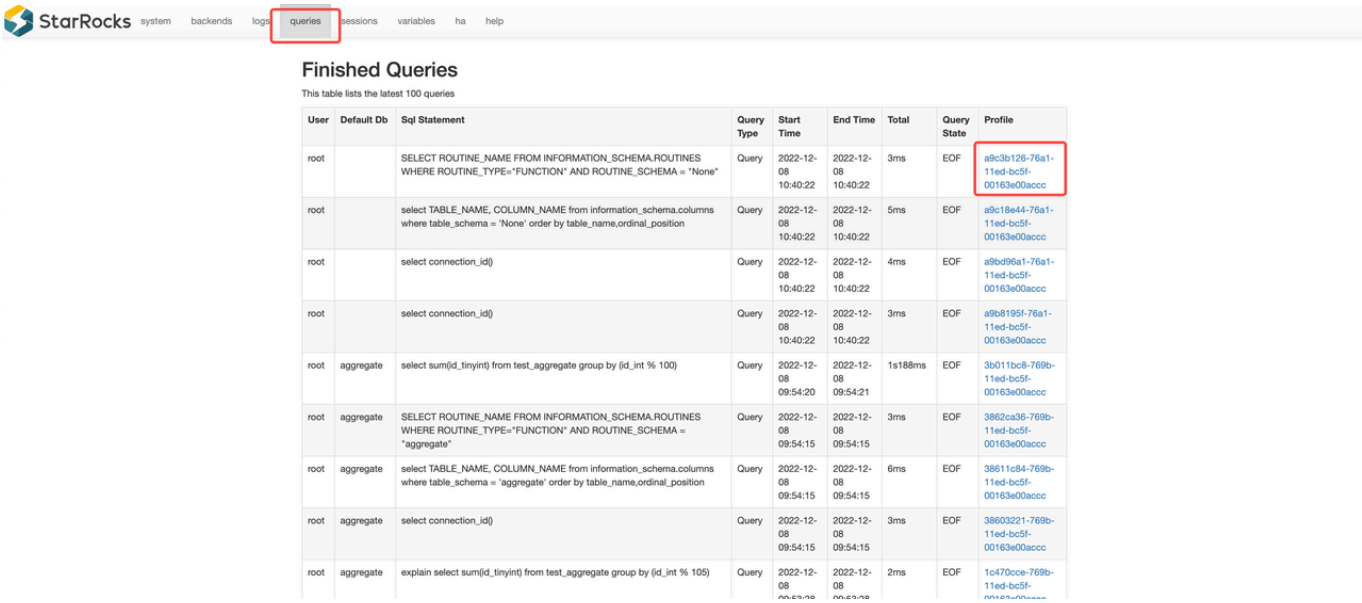
Конфигурации

Параметр конфигурации	Тип	Допустимые значения	По умолчанию	Описание
enable_profile	Session Var	true/false	false	Включить Query Profile
pipeline_profile_level	Session Var	1/2	1	1: объединить метрики; 2: сохранить исходную структуру (отключает инструменты визуализации)
runtime_profile_report_interval	Session Var	Положительное число	10	Интервал отчетности Runtime Query Profile (секунды)
big_query_profile_threshold	Session Var	String	0s	Включить Query Profile для запросов, превышающих эту продолжительность (например, '30s', '500ms', '60m')
enable_statistics_collect_profile	FE Dynamic	true/false	false	Включить Query Profile для запросов, связанных со сбором статистики

Как получить Query Profile

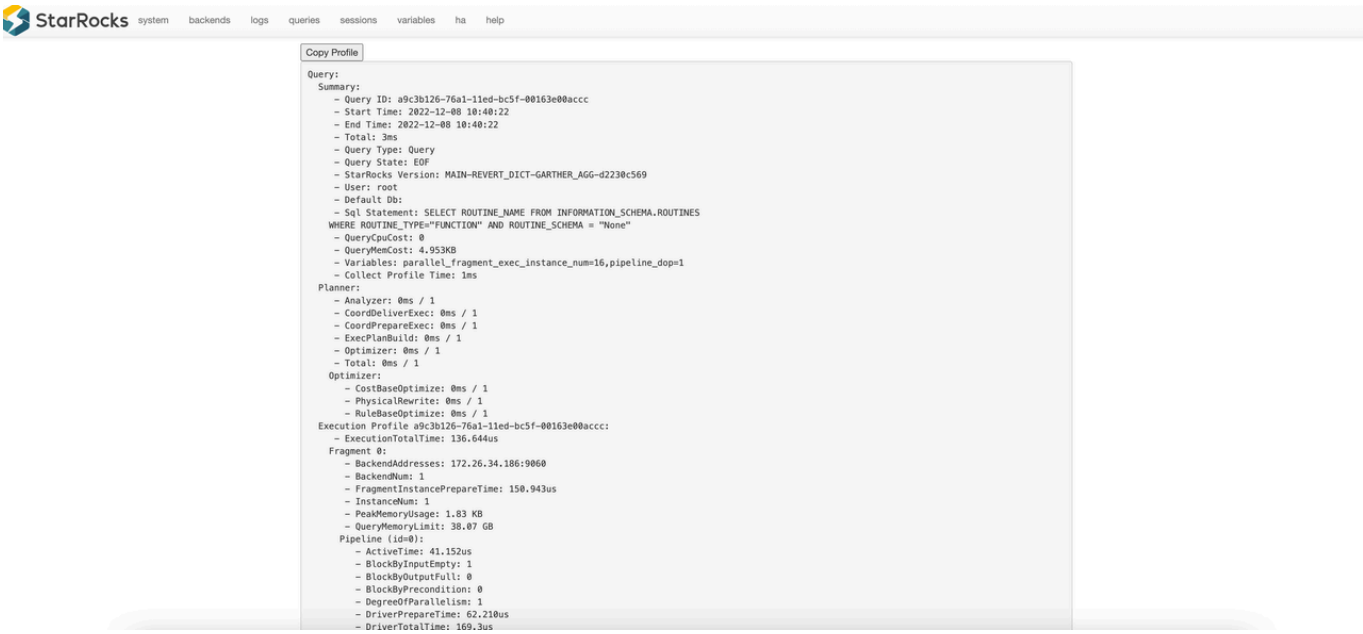
Через веб-интерфейс

1. Откройте в браузере `http://<fe_ip>:<fe_http_port>`.
2. Нажмите **queries** в верхней навигации.
3. В списке **Finished Queries** выберите запрос, который хотите проанализировать, и нажмите на ссылку в столбце **Profile**.



User	Default Db	Sql Statement	Query Type	Start Time	End Time	Total	Query State	Profile
root		SELECT ROUTINE_NAME FROM INFORMATION_SCHEMA.ROUTINES WHERE ROUTINE_TYPE="FUNCTION" AND ROUTINE_SCHEMA = "None"	Query	2022-12-08 10:40:22	2022-12-08 10:40:22	3ms	EOF	a9c3b126-76a1-11ed-bc5f-00163e00accc
root		select TABLE_NAME, COLUMN_NAME from information_schema.columns where table_schema = 'None' order by table_name,ordinal_position	Query	2022-12-08 10:40:22	2022-12-08 10:40:22	5ms	EOF	a9c18e44-76a1-11ed-bc5f-00163e00accc
root		select connection_id()	Query	2022-12-08 10:40:22	2022-12-08 10:40:22	4ms	EOF	a9bd96a1-76a1-11ed-bc5f-00163e00accc
root		select connection_id()	Query	2022-12-08 10:40:22	2022-12-08 10:40:22	3ms	EOF	a9b8195f-76a1-11ed-bc5f-00163e00accc
root	aggregate	select sum(id_tinyint) from test_aggregate group by (id_int % 100)	Query	2022-12-08 09:54:20	2022-12-08 09:54:21	1s188ms	EOF	3b011bc8-769b-11ed-bc5f-00163e00accc
root	aggregate	SELECT ROUTINE_NAME FROM INFORMATION_SCHEMA.ROUTINES WHERE ROUTINE_TYPE="FUNCTION" AND ROUTINE_SCHEMA = "aggregate"	Query	2022-12-08 09:54:15	2022-12-08 09:54:15	3ms	EOF	3862ca36-769b-11ed-bc5f-00163e00accc
root	aggregate	select TABLE_NAME, COLUMN_NAME from information_schema.columns where table_schema = 'aggregate' order by table_name,ordinal_position	Query	2022-12-08 09:54:15	2022-12-08 09:54:15	6ms	EOF	38611c84-769b-11ed-bc5f-00163e00accc
root	aggregate	select connection_id()	Query	2022-12-08 09:54:15	2022-12-08 09:54:15	3ms	EOF	38603221-769b-11ed-bc5f-00163e00accc
root	aggregate	explain select sum(id_tinyint) from test_aggregate group by (id_int % 105)	Query	2022-12-08	2022-12-08	2ms	EOF	1c470cce-769b-11ed-bc5f-

Вы будете перенаправлены на детальную страницу выбранного Query Profile.



Через SQL-функцию (`get_query_profile`)

Пример рабочего процесса:

- `last_query_id()`: Возвращает ID последнего выполненного запроса в вашей сессии. Полезно для быстрого получения profile вашего последнего запроса.
- `show profilelist;`: Выводит список последних запросов вместе с их ID и статусом. Используйте это, чтобы найти `query_id`, необходимый для анализа profile.
- `get_query_profile('<query_id>')`: Возвращает детальный profile выполнения для указанного запроса. Используйте это для анализа того, как запрос был выполнен и

где было потрачено время или ресурсы.

```
-- Включите функцию profiling.
set enable_profile = true;
-- Выполните простой запрос.
select 1;
-- Получите query_id запроса.
select last_query_id();
+--------------------------------------+
| last_query_id()                      |
+--------------------------------------+
| bd3335ce-8dde-11ee-92e4-3269eb8da7d1 |
+--------------------------------------+
-- Получите список profiles
show profilelist;
-- Получите query profile.
select get_query_profile('502f3c04-8f5c-11ee-a41f-b22a2c00f66b')\G
```

В управляемой версии

В окружениях Selena Managed (Enterprise) вы можете удобно получить доступ к query profiles непосредственно из истории запросов в веб-консоли. Управляемый UI предоставляет интуитивное визуальное представление profile выполнения каждого запроса, упрощая анализ производительности и идентификацию узких мест без ручных SQL-команд.

Интерпретация Query Profile

Explain Analyze

Большинству пользователей может быть сложно анализировать необработанный текст напрямую. Selena предоставляет метод текстового анализа Query Profile с визуализацией для более интуитивного понимания.

Управляемая версия

В Selena Enterprise Edition (EE) управляемая версия предоставляет встроенный инструмент визуализации для query profiles. Этот инструмент предлагает интерактивный графический интерфейс, который значительно упрощает интерпретацию сложных деталей выполнения запросов по сравнению с необработанным текстовым выводом.

Ключевые возможности инструмента визуализации включают:

- **Разбивка на уровне операторов:** Просмотр плана выполнения в виде дерева или графа, с четким отображением метрик каждого оператора (время, строки, память).
- **Подсветка узких мест:** Быстрая идентификация медленных или ресурсоемких операторов с помощью цветовых индикаторов.
- **Возможность детализации:** Нажмите на любой оператор, чтобы увидеть детальную статистику, включая входные/выходные строки, время CPU, использование памяти и многое другое.

Как использовать:

1. Откройте веб-консоль Selena Managed.
2. Перейдите в раздел **Query** или **Query History**.
3. Выберите запрос и нажмите кнопку **Profile** или **Visualize**.
4. Изучите визуализированный profile для анализа производительности и определения возможностей оптимизации.

Этот инструмент визуализации эксклюзивен для Managed/Enterprise Edition и разработан для ускорения устранения неполадок и настройки производительности для сложных рабочих нагрузок.

1.8.4 Рецепты настройки запросов

> Прагматичное руководство: **симптом → коренная причина → проверенные решения**. Используйте его, когда вы открыли profile и обнаружили тревожную метрику, но все еще нужно ответить на вопрос "*что теперь?*".

1 · Рабочий процесс быстрой диагностики

1. **Просмотрите обзор выполнения** Если `QueryPeakMemoryUsagePerNode > 80 %` или `QuerySpillBytes > 1 GB`, сразу переходите к рецептам по памяти и spillage.

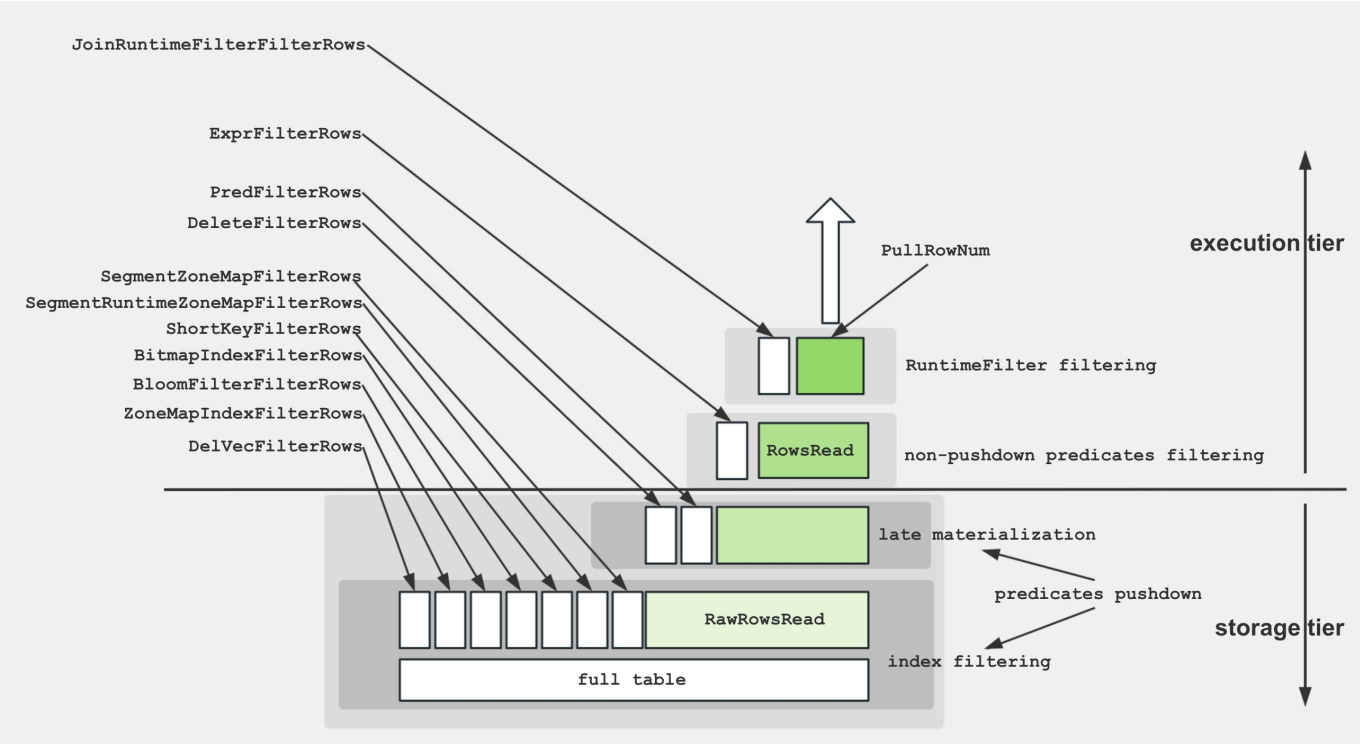
2. **Найдите самый медленный Pipeline / Operator ⟶** В *Query Profile UI* нажмите **Sort by OperatorTotalTime %**. Самый горячий оператор подскажет, какой блок рецептов читать дальше (Scan, Join, Aggregate, …).

3. **Подтвердите подтип узкого места** Каждый рецепт начинается со своего *характерного* паттерна метрик. Сверьте их перед применением исправлений.

2 · Рецепты по операторам

2.1 OLAP / Connector Scan [метрики]

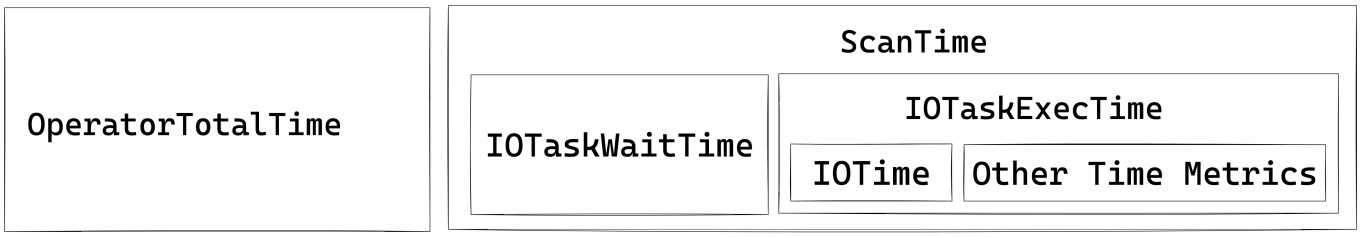
Для облегчения понимания различных метрик внутри Scan Operator следующая диаграмма демонстрирует взаимосвязи между этими метриками и структурами хранения.



Для извлечения данных с диска и применения предикатов механизм хранения использует несколько техник:

1. **Хранение данных**: Закодированные и сжатые данные хранятся на диске в сегментах, сопровождаемые различными индексами.
2. **Фильтрация по индексам**: Движок использует индексы, такие как BitmapIndex, BloomfilterIndex, ZonemapIndex, ShortKeyIndex и NGramIndex, чтобы пропускать ненужные данные.
3. **Предикаты с Pushdown**: Простые предикаты, такие как `a > 1`, проталкиваются вниз для оценки на конкретных столбцах.
4. **Поздняя материализация**: С диска извлекаются только необходимые столбцы и отфильтрованные строки.
5. **Предикаты без Pushdown**: Оцениваются предикаты, которые не могут быть протолкнуты вниз.
6. **Выражение проекции**: Вычисляются выражения, такие как `SELECT a + 1`.

Scan Operator использует дополнительный пул потоков для выполнения задач IO. Поэтому взаимосвязь между метриками времени для этого узла показана ниже:



Распространенные узкие места производительности

Холодное или медленное хранилище – Когда `BytesRead`, `ScanTime` или `IOTaskExecTime` доминируют и дисковый I/O находится около 80-100 %, сканирование сталкивается с холодным или недостаточно выделенным хранилищем. Переместите горячие данные на NVMe/SSD и включите Data Cache. Настройте его через параметры BE `datacache_*` (или устаревшие `block_cache_*`) и включите использование во время сканирования через параметр сессии `enable_scan_datacache`.

Отсутствие push-down фильтров – Если `PushdownPredicates` остается близким к 0, а `ExprFilterRows` высокий, предикаты не достигают уровня хранилища. Перепишите их как простые сравнения (избегайте `%LIKE%` и широких цепочек `OR`) или добавьте индексы zonemap/Bloom или материализованные представления, чтобы они могли быть протолкнуты вниз.

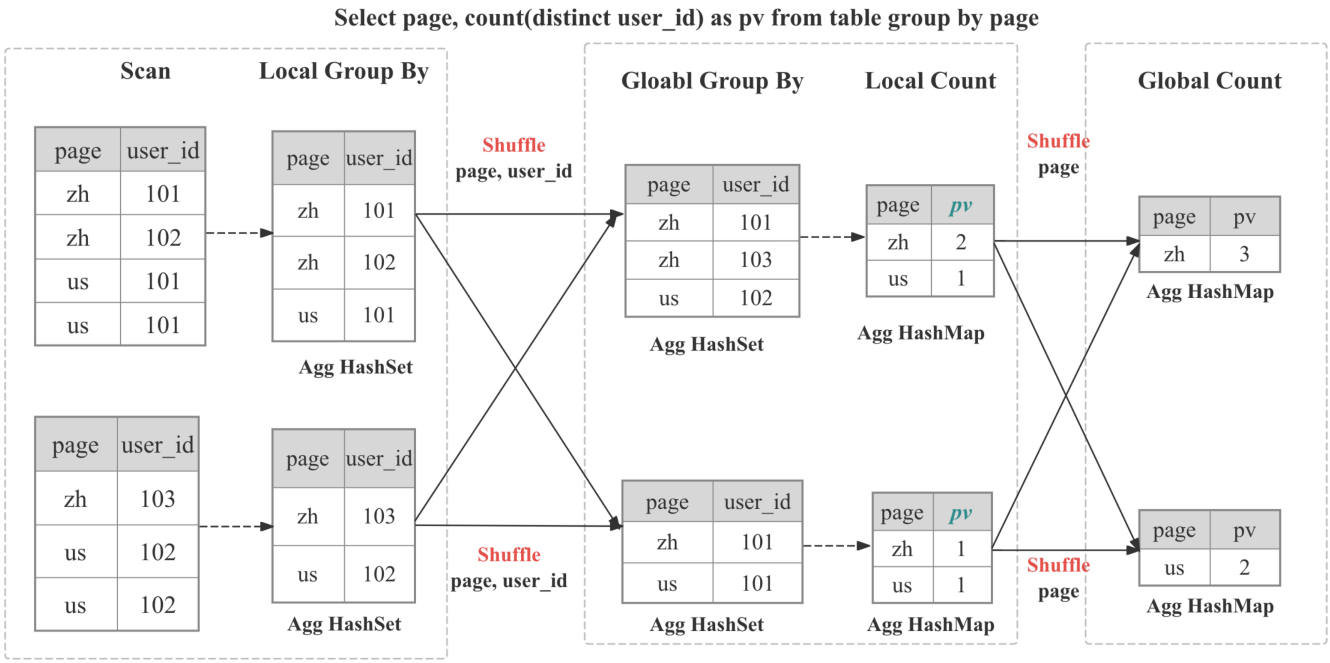
Нехватка потоков в пуле – Высокий `IOTaskWaitTime` вместе с низким `PeakIOTasks` сигнализирует о насыщенной конкурентности I/O. Включите и настройте размер Data Cache (параметры BE `datacache_*` и параметр сессии `enable_scan_datacache`), переместите горячие данные на более быстрое хранилище (NVMe/SSD)

Перекос данных по tablet – Большой разрыв между максимальным и минимальным `OperatorTotalTime` означает, что некоторые tablet выполняют гораздо больше работы, чем другие. Перебакетируйте по ключу с большей кардинальностью или увеличьте количество bucket для распределения нагрузки.

Фрагментация rowset/segment – Растущие `RowsetsReadCount`/`SegmentsReadCount` плюс длинный `SegmentInitTime` указывают на множество крошечных rowset. Запустите ручную компакцию и пакетируйте небольшие загрузки, чтобы сегменты объединялись заранее.

Накопленные мягкие удаления – Большой `DeleteFilterRows` подразумевает интенсивное использование мягкого удаления. Запустите компакцию BE для очистки мягких удалений.

2.2 Aggregate [метрики]



Aggregate Operator отвечает за выполнение функций агрегации, `GROUP BY` и `DISTINCT`.

Множественные формы алгоритма агрегации

Форма	Когда планировщик выбирает её	Внутренняя структура данных	Особенности / предостережения
Hash aggregation	ключи помещаются в память; кардинальность не экстремальная	Компактная хеш-таблица с SIMD-зондированием	путь по умолчанию, отлично подходит для умеренного количества ключей
Sorted aggregation	входные данные уже упорядочены по ключам GROUP BY	Простое сравнение строк + текущее состояние	нулевая стоимость хеш-таблицы, часто в 2-3× быстрее при перекосах с тяжелым зондированием
Spillable aggregation (3.2+)	хеш-таблица превышает лимит памяти	Гибридный хеш/слияние с дисковыми partitions	предотвращает OOM, сохраняет параллелизм pipeline

Многоступенчатая распределенная агрегация

В Selena агрегация реализована распределенным образом, что может быть многоступенчатым в зависимости от паттерна запроса и решения оптимизатора.

```
┌──────────┐        ┌──────────┐        ┌──────────────┐        ┌──────────────┐
| Stage 0 | local  | Stage 1  | shard/ | Stage 2      | gather/| Stage 3      |
final
| Partial |──►     | Update   | hash   | Merge        | shard  | Finalize     |
output
└──────────┘        └──────────┘        └──────────────┘        └──────────────┘
```

Стадии	Когда используется	Что происходит
Одноступенчатая	`DISTRIBUTED BY` является подмножеством `GROUP BY`, партиции совмещены	Частичные агрегаты сразу становятся финальным результатом.
Двухступенчатая (локальная + глобальная)	Типичная распределенная `GROUP BY`	Stage 0 внутри каждого BE адаптивно сворачивает дубликаты; Stage 1 перемешивает данные на основе `GROUP BY`, затем выполняет глобальную агрегацию
Трехступенчатая (локальная + shuffle + финальная)	Тяжелая `DISTINCT` и высококардинальная `GROUP BY`	Stage 0 как выше; Stage 1 перемешивает по `GROUP BY`, затем агрегирует по `GROUP BY` и `DISTINCT`; Stage 2 объединяет частичное состояние как `GROUP BY`
Четырехступенчатая (локальная + частичная + промежуточная + финальная)	Тяжелая `DISTINCT` и низкокардинальная `GROUP BY`	Вводит дополнительную стадию для перемешивания по `GROUP BY` и `DISTINCT`, чтобы избежать узкого места в одной точке

Распространенные узкие места производительности

Высококардинальная GROUP BY – Когда `HashTableSize` или `HashTableMemoryUsage` раздуваются до лимита памяти, ключ группировки слишком широкий или слишком различный. Включите сортированную потоковую агрегацию (`enable_streaming_preaggregation = true`), создайте материализованное представление roll-up или приведите широкие строковые ключи к `INT`.

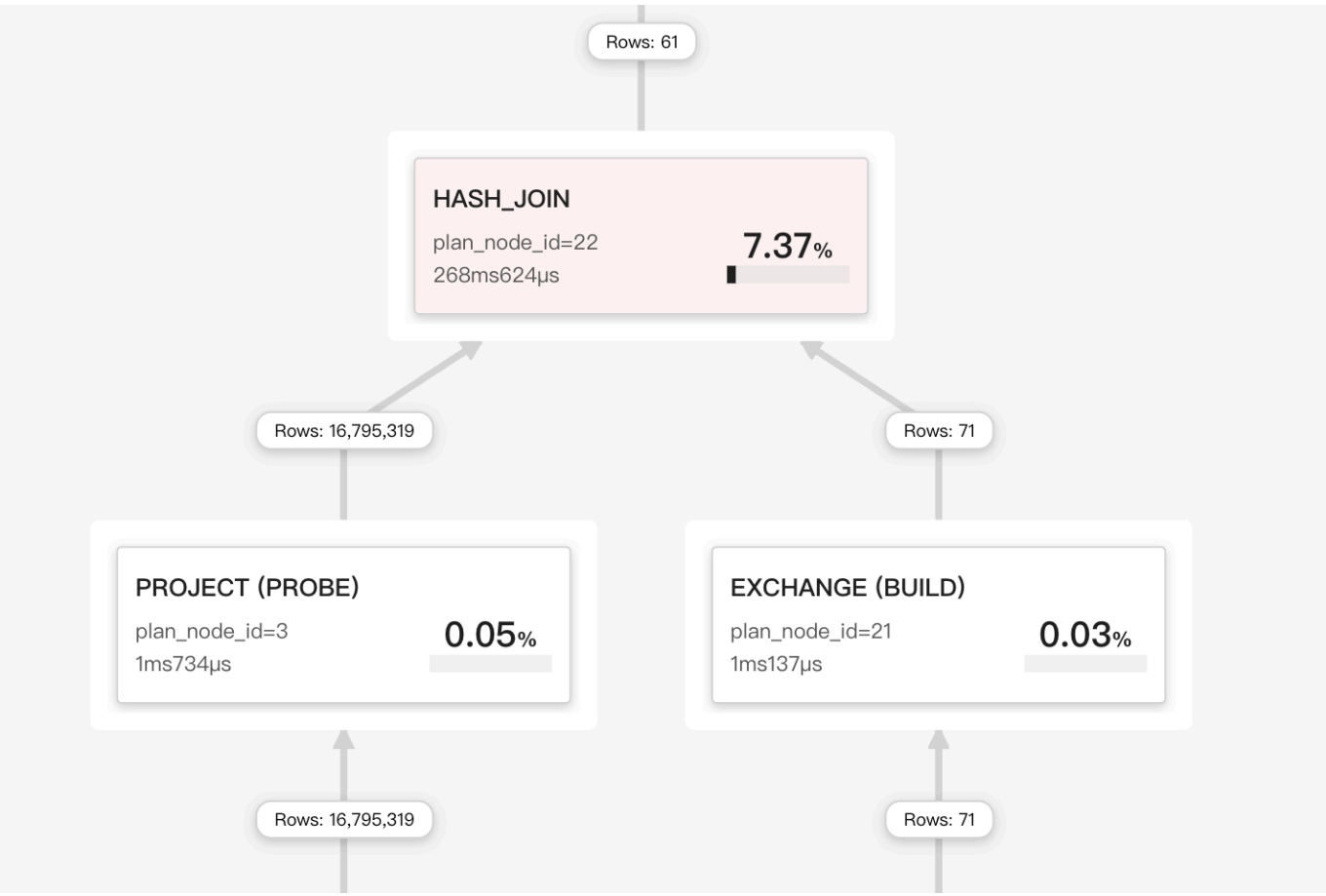
Перекос Shuffle – Большие различия в `HashTableSize` или `InputRowCount` между фрагментами выявляют несбалансированное перемешивание. Добавьте столбец salt к ключу или используйте подсказку `DISTINCT [skew]`, чтобы строки распределялись равномерно.

Функции агрегации с тяжелым состоянием – Если `AggregateFunctions` доминирует во времени выполнения, а функции включают `HLL_`, `BITMAP_` или `COUNT(DISTINCT)`, перемещаются огромные объекты состояния. Предварительно вычислите HLL/bitmap-скетчи во время загрузки или переключитесь на приблизительные варианты.

Деградированная частичная агрегация – Огромный `InputRowCount` с умеренным `AggComputeTime` плюс массивный `BytesSent` в upstream EXCHANGE означает, что предварительная агрегация была обойдена. Принудительно включите её с помощью `SET streaming_preaggregation_mode = "force_preaggregation"`.

Дорогие выражения ключей – Когда `ExprComputeTime` соперничает с `AggComputeTime`, ключи GROUP BY вычисляются построчно. Материализуйте эти выражения в подзапросе или продвиньте их в сгенерированные столбцы.

2.3 Join [метрики]



Join Operator отвечает за реализацию явных соединений или неявных соединений.

Во время выполнения оператор join разделяется на фазы Build (построение хеш-таблицы) и Probe, которые выполняются параллельно внутри движка pipeline. Векторные чанки (до 4096 строк) пакетно хешируются с SIMD; потребленные ключи генерируют runtime-фильтры — Bloom или IN фильтры — проталкиваемые обратно к upstream-сканированиям для раннего сокращения входных данных probe.

Стратегии Join

Selena опирается на векторизованное, дружественное к pipeline ядро hash-join, которое может быть подключено к четырем физическим стратегиям, которые оптимизатор на основе стоимости взвешивает во время планирования:

Стратегия	Когда оптимизатор выбирает её	Что делает её быстрой
Colocate Join	Обе таблицы принадлежат к одной и той же colocation-группе (идентичные bucket-ключи, количество bucket и размещение replica).	Нет сетевого перемешивания: каждый BE соединяет только свои локальные bucket.
Bucket-Shuffle Join	Одна из таблиц join имеет тот же bucket-ключ, что и ключ join	Необходимо перемешать только одну таблицу join, что может снизить сетевые затраты
Broadcast Join	Сторона Build очень мала (пороги строк/байтов или явная подсказка).	Небольшая таблица реплицируется на каждый узел probe; избегает перемешивания большой таблицы.
Shuffle (Hash) Join	Общий случай, ключи не совпадают.	Хеш-партиционирование каждой строки по ключу join, чтобы probe-запросы были сбалансированы по BE.

Распространенные узкие места производительности

Слишком большая сторона build – Скачки в `BuildHashTableTime` и `HashTableMemoryUsage` показывают, что сторона build переросла память. Поменяйте местами таблицы probe/build, предварительно отфильтруйте таблицу build или включите hash spilling.

Probe, неэффективный для кеша – Когда `SearchHashTableTime` доминирует, сторона probe не эффективна для кеша. Отсортируйте строки probe по ключам join и включите runtime-фильтры.

Перекос Shuffle – Если `ProbeRows` одного фрагмента затмевает все остальные, данные перекошены. Переключитесь на ключ с более высокой кардинальностью или добавьте salt, такой как `key || mod(id, 16)`.

Случайный broadcast – Тип Join **BROADCAST** с огромным `BytesSent` означает, что таблица, которую вы считали маленькой, не является таковой. Снизьте `broadcast_row_limit` или принудительно используйте shuffle с подсказкой `SHUFFLE`.

Отсутствующие runtime-фильтры – Крошечный `JoinRuntimeFilterEvaluate` вместе с полнотабличными сканированиями предполагает, что runtime-фильтры никогда не

распространялись. Перепишите join как чистое равенство и убедитесь, что типы столбцов совпадают.

Откат к non-equi – Когда тип оператора `CROSS` или `NESTLOOP`, неравенство или функция предотвращает hash join. Добавьте истинный предикат равенства или предварительно отфильтруйте большую таблицу.

2.4 Exchange (сеть) [метрики]

Слишком большое shuffle или broadcast – Если `NetworkTime` превышает 30 % и `BytesSent` большой, запрос отправляет слишком много данных. Пересмотрите стратегию join и уменьшите объем shuffle/broadcast (например, принудительно используйте shuffle вместо broadcast или предварительно отфильтруйте upstream).

Очередь приемника – Высокий `WaitTime` в приемнике с очередями отправителя, которые остаются заполненными, указывает, что приемник не может справиться. Увеличьте пул потоков приемника (`brpc_num_threads`) и подтвердите настройки пропускной способности NIC и QoS.

Включите сжатие exchange – Когда пропускная способность сети является узким местом, сжимайте полезную нагрузку exchange. Установите `SET transmission_compression_type = 'zstd';` и опционально увеличьте `SET transmission_encode_level = 7;`, чтобы включить адаптивное кодирование столбцов. Ожидайте более высокое использование CPU в обмен на уменьшенное количество байтов по сети.

2.5 Sort / Merge / Window

Для облегчения понимания различных метрик Merge можно представить в виде следующего механизма состояний:

```
                ┌──────────── PENDING ◀──────────┐
                │                                │
                │                                │
                ├────────────────◀───────────────┤
                │                                │
                ▼                                │
INIT ──▶ PREPARE ──▶ SPLIT_CHUNK ──▶ FETCH_CHUNK ──▶ FINISHED
                ▲
                |
                | one traverse from leaf to root
                |
                ▼
            PROCESS
```

Sort spilling – Когда `MaxBufferedBytes` поднимается выше примерно 2 GB или `SpillBytes` не равен нулю, фаза сортировки сбрасывается на диск. Добавьте `LIMIT`, предварительно агрегируйте upstream или поднимите `sort_spill_threshold`, если у машины достаточно памяти.

Нехватка Merge – Высокий `PendingStageTime` говорит вам, что merge ждет upstream-чанков. Сначала оптимизируйте оператор-производитель или увеличьте буферы pipeline.

Широкие партиции window – Огромный `PeakBufferedRows` внутри оператора window указывает на очень широкие партиции или ORDER BY, лишенный ограничений frame. Партиционируйте более детально, добавьте границы `RANGE BETWEEN` или материализуйте промежуточные агрегаты.

3 · Шпаргалка по памяти и spillage

Порог	Что отслеживать	Практическое действие
80 % памяти BE	`QueryPeakMemoryUsagePerNode`	Снизьте `exec_mem_limit` сессии или добавьте RAM в BE
Обнаружен spill (`SpillBytes` > 0)	`QuerySpillBytes`, `SpillBlocks` для каждого оператора	Увеличьте лимит памяти; обновитесь до SR 3.2+ для гибридного hash/merge spill

4 · Шаблон для вашего post-mortem

```
1. Симптом
   – Медленная стадия: Aggregate (OperatorTotalTime 68 %)
   – Тревожный сигнал: HashTableMemoryUsage 9 GB (> exec_mem_limit)
2. Коренная причина
   – GROUP BY высококардинального UUID
3. Примененное исправление
   – Добавлена сортированная потоковая агрегация + roll-up MV
4. Результат
   – Время выполнения запроса ↓ с 95 s → 8 s; пик памяти 0.7 GB```
```

1.8.5 Рецепты настройки схемы

Этот документ предоставляет практические советы и рекомендации по оптимизации производительности запросов в Selena через эффективный дизайн схемы и выбор базовых типов таблиц. Понимая, как различные типы таблиц, ключи и стратегии распределения влияют на выполнение запросов, вы можете значительно улучшить как скорость, так и эффективность использования ресурсов. Используйте эти рекомендации для принятия обоснованных решений при проектировании схем, выборе типов таблиц и настройке вашей среды Selena для высокопроизводительной аналитики.

Выбор типа таблицы

Selena поддерживает четыре типа таблиц: Duplicate Key table, Aggregate table, Unique Key table и Primary Key table. Все они сортируются по KEY.

- `AGGREGATE KEY`: Когда записи с одинаковым AGGREGATE KEY загружаются в Selena, старые и новые записи агрегируются. В настоящее время таблицы Aggregate поддерживают следующие функции агрегации: SUM, MIN, MAX и REPLACE. Таблицы Aggregate поддерживают предварительную агрегацию данных, облегчая бизнес-операторы и многомерные анализы.
- `DUPLICATE KEY`: Вам нужно указать только ключ сортировки для таблицы DUPLICATE KEY. Записи с одинаковым DUPLICATE KEY существуют одновременно. Это подходит для анализов, не включающих предварительную агрегацию данных.
- `UNIQUE KEY`: Когда записи с одинаковым UNIQUE KEY загружаются в Selena, новая запись перезаписывает старую. Таблица UNIQUE KEY похожа на таблицу Aggregate с функцией REPLACE. Обе подходят для анализов с постоянными обновлениями.

- `PRIMARY KEY`: Таблицы Primary Key гарантируют уникальность записей и позволяют выполнять обновления в реальном времени.

```sql
CREATE TABLE site_visit
(
    siteid      INT,
    city        SMALLINT,
    username    VARCHAR(32),
    pv BIGINT   SUM DEFAULT '0'
)
AGGREGATE KEY(siteid, city, username)
DISTRIBUTED BY HASH(siteid);


CREATE TABLE session_data
(
    visitorid   SMALLINT,
    sessionid   BIGINT,
    visittime   DATETIME,
    city        CHAR(20),
    province    CHAR(20),
    ip          varchar(32),
    browser      CHAR(20),
    url         VARCHAR(1024)
)
DUPLICATE KEY(visitorid, sessionid)
DISTRIBUTED BY HASH(sessionid, visitorid);

CREATE TABLE sales_order
(
    orderid     BIGINT,
    status      TINYINT,
    username    VARCHAR(32),
    amount      BIGINT DEFAULT '0'
)
UNIQUE KEY(orderid)
DISTRIBUTED BY HASH(orderid);

CREATE TABLE sales_order
(
    orderid     BIGINT,
    status      TINYINT,
    username    VARCHAR(32),
    amount      BIGINT DEFAULT '0'
)
PRIMARY KEY(orderid)
DISTRIBUTED BY HASH(orderid);
```

Colocate Table

Для ускорения запросов таблицы с одинаковым распределением могут использовать общий столбец для bucketing. В этом случае данные могут быть соединены локально без передачи по cluster во время операции `join`.

```
CREATE TABLE colocate_table
(
    visitorid   SMALLINT,
    sessionid   BIGINT,
    visittime   DATETIME,
    city        CHAR(20),
    province    CHAR(20),
    ip          varchar(32),
    browser      CHAR(20),
    url         VARCHAR(1024)
)
DUPLICATE KEY(visitorid, sessionid)
DISTRIBUTED BY HASH(sessionid, visitorid)
PROPERTIES(
    "colocate_with" = "group1"
);
```

Для получения дополнительной информации о colocate join и управлении replica см. Colocate join

Плоские таблицы и схема звезды

Selena поддерживает схему звезды, которая более гибка в моделировании, чем плоские таблицы. Вы можете создать представление для замены плоских таблиц во время моделирования, а затем запрашивать данные из нескольких таблиц для ускорения запросов.

Плоские таблицы имеют следующие недостатки:

- Дорогие обновления измерений, потому что плоская таблица обычно содержит огромное количество измерений. Каждый раз, когда измерение обновляется, вся таблица должна быть обновлена. Ситуация усугубляется с увеличением частоты обновлений.
- Высокая стоимость обслуживания, потому что плоские таблицы требуют дополнительных рабочих нагрузок разработки, пространства хранения и операций

backfilling данных.
- Высокая стоимость загрузки данных, потому что плоская таблица имеет много полей, а таблица Aggregate может содержать еще больше ключевых полей. Во время загрузки данных необходимо отсортировать больше полей, что продлевает загрузку данных.

Если у вас высокие требования к конкурентности запросов или низкой задержке, вы все еще можете использовать плоские таблицы.

Partition и bucket

Selena поддерживает два уровня партиционирования: первый уровень — RANGE partition, а второй уровень — HASH bucket.

- RANGE partition: RANGE partition используется для разделения данных на различные интервалы (можно понимать как разделение исходной таблицы на несколько подтаблиц). Большинство пользователей выбирают установку партиций по времени, что имеет следующие преимущества:
    - Легче различать горячие и холодные данные
    - Возможность использовать многоуровневое хранилище Selena (SSD + SATA)
    - Быстрее удалять данные по partition
- HASH bucket: Разделяет данные на разные bucket в соответствии с хеш-значением.
    - Рекомендуется использовать столбец с высокой степенью различения для bucketing, чтобы избежать перекоса данных.
    - Для облегчения восстановления данных рекомендуется поддерживать размер сжатых данных в каждом bucket между 100 MB и 1 GB. Мы рекомендуем настроить соответствующее количество bucket при создании таблицы или добавлении partition.
    - Случайный bucketing не рекомендуется. Вы должны явно указать столбец HASH bucketing при создании таблицы.

Разреженный индекс и bloomfilter index

Selena хранит данные упорядоченным образом и строит разреженные индексы с детализацией в 1024 строки.

Selena выбирает префикс фиксированной длины (в настоящее время 36 байтов) в схеме в качестве разреженного индекса.

При создании таблицы рекомендуется размещать общие поля фильтрации в начале объявления схемы. Поля с наибольшей дифференциацией и частотой запросов должны быть размещены первыми.

Поле VARCHAR должно быть размещено в конце разреженного индекса, потому что индекс обрезается с поля VARCHAR. Если поле VARCHAR появляется первым, индекс может быть меньше 36 байтов.

Используйте приведенную выше таблицу `site_visit` в качестве примера. Таблица имеет четыре столбца: `siteid, city, username, pv`. Ключ сортировки содержит три столбца `siteid, city, username`, которые занимают 4, 2 и 32 байта соответственно. Таким образом, префиксный индекс (разреженный индекс) может быть первыми 30 байтами `siteid + city + username`.

Помимо разреженных индексов Selena также предоставляет bloomfilter-индексы, которые эффективны для фильтрации столбцов с высокой дифференциацией. Если вы хотите разместить поля VARCHAR перед другими полями, вы можете создать bloomfilter-индексы.

Инвертированный индекс

Selena использует технологию Bitmap Indexing для поддержки инвертированных индексов, которые могут применяться ко всем столбцам таблицы Duplicate Key и ключевому столбцу таблицы Aggregate и таблицы Unique Key. Bitmap Index подходит для столбцов с небольшим диапазоном значений, таких как пол, город и провинция. По мере расширения диапазона bitmap-индекс расширяется параллельно.

Материализованное представление (rollup)

Rollup по сути является материализованным индексом исходной таблицы (базовой таблицы). При создании rollup можно выбрать только некоторые столбцы базовой таблицы в качестве схемы, и порядок полей в схеме может отличаться от порядка в базовой таблице. Ниже приведены некоторые случаи использования rollup:

- Агрегация данных в базовой таблице невысока, потому что базовая таблица имеет поля с высокой дифференциацией. В этом случае вы можете рассмотреть выбор

некоторых столбцов для создания rollup. Используйте приведенную выше таблицу `site_visit` в качестве примера:

```
site_visit(siteid, city, username, pv)
```

`siteid` может привести к плохой агрегации данных. Если вам нужно часто вычислять PV по городам, вы можете создать rollup только с `city` и `pv`.

```
ALTER TABLE site_visit ADD ROLLUP rollup_city(city, pv);
```

- Префиксный индекс в базовой таблице не может быть использован, потому что способ построения базовой таблицы не может охватить все паттерны запросов. В этом случае вы можете рассмотреть создание rollup для настройки порядка столбцов. Используйте приведенную выше таблицу `session_data` в качестве примера:

```
session_data(visitorid, sessionid, visittime, city, province, ip,
browser, url)
```

Если есть случаи, когда вам нужно анализировать посещения по `browser` и `province` в дополнение к `visitorid`, вы можете создать отдельный rollup:

```
ALTER TABLE session_data
ADD ROLLUP rollup_browser(browser,province,ip,url)
DUPLICATE KEY(browser,province);
```

Изменение схемы

Существует три способа изменения схем в Selena: сортированное изменение схемы, прямое изменение схемы и связанное изменение схемы.

- Сортированное изменение схемы: Изменение сортировки столбца и переупорядочивание данных. Например, удаление столбца в сортированной схеме приводит к переупорядочиванию данных.

`ALTER TABLE site_visit DROP COLUMN city;`

- Прямое изменение схемы: Преобразование данных вместо их переупорядочивания, например, изменение типа столбца или добавление столбца в разреженный индекс.

  `ALTER TABLE site_visit MODIFY COLUMN username varchar(64);`

- Связанное изменение схемы: Завершение изменений без преобразования данных, например, добавление столбцов.

  `ALTER TABLE site_visit ADD COLUMN click bigint SUM default '0';`

  Рекомендуется выбрать подходящую схему при создании таблиц для ускорения изменений схемы.

1.8.6 Explain Analyze

Этот документ объясняет, как получить и проанализировать текстовые Query Profile в Selena. Это поможет вам понять производительность запросов и найти способы оптимизации ваших SQL-запросов.

Анализ Profile существующих запросов с помощью ANALYZE PROFILE

Чтобы проанализировать текстовый Profile существующего (исторического или выполняющегося) запроса в вашем cluster, сначала необходимо использовать оператор SHOW PROFILELIST для получения сводки по запросу. Эта команда перечисляет все запросы, которые успешно завершились, завершились с ошибкой, а также те, которые все еще выполняются (более 10 секунд и еще не завершены). С помощью этого оператора вы можете получить соответствующий Query ID для последующего анализа. Синтаксис следующий:

```
SHOW PROFILELIST [LIMIT <num>];
```

Примеры:

```
SHOW PROFILELIST;
SHOW PROFILELIST LIMIT 5;
```

Вывод:

```
+--------------------------------------+---------------------+-------+-----
-----+-------------------------------------------------------------------
-------------------------------------------------------------+
| QueryId                              | StartTime           | Time  |
State    | Statement
|
+--------------------------------------+---------------------+-------+-----
-----+-------------------------------------------------------------------
-------------------------------------------------------------+
| a40456b2-8428-11ee-8d02-6a32f8c68848 | 2023-11-16 10:34:18 | 21ms  |
Finished | SELECT ROUTINE_NAME FROM INFORMATION_SCHEMA.ROUTINES\n    WHERE
ROUTINE_TYPE="FUNCTION" AND ROUTINE_SCHEMA = "None"               |
| a3fc4060-8428-11ee-8d02-6a32f8c68848 | 2023-11-16 10:34:17 | 39ms  |
Finished | select TABLE_NAME, COLUMN_NAME from information_schema.columns\n
where table_schema = 'Non ... |
| a3f7d38d-8428-11ee-8d02-6a32f8c68848 | 2023-11-16 10:34:17 | 15ms  |
Finished | select connection_id()
|
| a3efbd3b-8428-11ee-8d02-6a32f8c68848 | 2023-11-16 10:34:17 | 16ms  |
Finished | select connection_id()
|
| a26ec286-8428-11ee-8d02-6a32f8c68848 | 2023-11-16 10:34:15 | 269ms |
Error    | EXPLAIN ANALYZE  SELECT c_nation, s_nation, year(lo_orderdate)
AS year , SUM(lo_revenue) AS revenue FROM lineorder_flat WHERE ...  |
+--------------------------------------+---------------------+-------+-----
-----+-------------------------------------------------------------------
-------------------------------------------------------------+
```

Как только у вас есть Query ID, вы можете приступить к анализу Query Profile, используя оператор ANALYZE PROFILE. Синтаксис следующий:

```
ANALYZE PROFILE FROM '<Query_ID>' [, <Node_ID> [, ...] ]
```

- `Query_ID`: ID, соответствующий запросу, полученный из оператора `SHOW PROFILELIST`.
- `Node_ID`: ID узла Profile. Для узлов, чьи ID указаны, Selena возвращает детальную информацию о метриках для этих узлов. Для узлов, чьи ID не указаны, Selena возвращает только сводную информацию.

Profile включает следующие разделы:

- Summary: Сводная информация Profile.
  - QueryID
  - Информация о версии
  - Статус запроса, включая `Finished`, `Error` и `Running`.
  - Общее время выполнения запроса.
  - Использование памяти
  - Топ-10 узлов с самым высоким использованием CPU.
  - Топ-10 узлов с самым высоким использованием памяти.
  - Переменные сессии, которые имеют значение, отличное от значения по умолчанию.
- Fragments: Отображает метрики для каждого узла в каждом Fragment.
  - Время, использование памяти, информация об оценке стоимости и количество выходных строк для каждого узла.
  - Узлы с процентом использования времени, превышающим 30%, выделяются красным цветом.
  - Узлы с процентом использования времени, превышающим 15% и менее 30%, выделяются розовым цветом.

Пример 1: Запрос Query Profile без указания ID узла.

```
MySQL root@172.26.81.178:tpch> analyze profile from '35fc9ef0-8442-11ee-844d-00163e0f59de';
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Explain String                                                                                                                                                                                                                |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Summary                                                                                                                                                                                                                       |
|     QueryId: 35fc9ef0-8442-11ee-844d-00163e0f59de                                                                                                                                                                             |
|     Version: profile-bba710c                                                                                                                                                                                                  |
|     State: Finished                                                                                                                                                                                                           |
|     TotalTime: 26ms                                                                                                                                                                                                           |
|         ExecutionTime: 11.747ms [Scan: 4.489ms (38.21%), Network: 1.874ms (15.95%), ResultDeliverTime: 0ns (0.00%), ScheduleTime: 912.743us (7.77%)]                                                                          |
|         CollectProfileTime: 6ms                                                                                                                                                                                               |
|         FrontendProfileMergeTime: 5.049ms                                                                                                                                                                                     |
|     QueryPeakMemoryUsage: 710.030 KB, QueryAllocatedMemoryUsage: 115.099 MB                                                                                                                                                   |
|     Top Most Time-consuming Nodes:                                                                                                                                                                                            |
|         1. OLAP_SCAN (id=0) : 5.745ms (68.87%)                                                                                                                                                                                |
|         2. EXCHANGE (id=3) [GATHER]: 1.986ms (23.81%)                                                                                                                                                                         |
|         3. PROJECT (id=1) : 343.661us (4.12%)                                                                                                                                                                                 |
|         4. AGGREGATION (id=2) [serialize, update]: 172.182us (2.06%)                                                                                                                                                          |
|         5. AGGREGATION (id=4) [finalize, merge]: 61.624us (0.74%)                                                                                                                                                             |
|         6. RESULT_SINK: 32.340us (0.39%)                                                                                                                                                                                      |
|     Top Most Memory-consuming Nodes:                                                                                                                                                                                          |
|         1. OLAP_SCAN (id=0) : 2.525 MB                                                                                                                                                                                        |
|         2. EXCHANGE (id=3) [GATHER]: 7.234 KB                                                                                                                                                                                 |
|         3. AGGREGATION (id=2) [serialize, update]: 3.671 KB                                                                                                                                                                   |
|         4. AGGREGATION (id=4) [finalize, merge]: 2.991 KB                                                                                                                                                                     |
|     NonDefaultVariables:                                                                                                                                                                                                      |
|         enable_adaptive_sink_dop: false -> true                                                                                                                                                                               |
|         enable_async_profile: true -> false                                                                                                                                                                                   |
|         enable_profile: false -> true                                                                                                                                                                                         |
|         prefer_compute_node: false -> true                                                                                                                                                                                    |
|         sql_mode_v2: 32 -> 34                                                                                                                                                                                                 |
| Fragment 0                                                                                                                                                                                                                    |
| │   BackendNum: 1                                                                                                                                                                                                           |
| │   InstancePeakMemoryUsage: 22.194 KB, InstanceAllocatedMemoryUsage: 53.422 KB                                                                                                                                            |
| │   PrepareTime: 599.073us                                                                                                                                                                                                  |
| └──RESULT_SINK                                                                                                                                                                                                              |
| │    TotalTime: 32.340us (0.39%) [CPUTime: 32.340us]                                                                                                                                                                        |
| │    OutputRows: 1                                                                                                                                                                                                           |
| │    SinkType: MYSQL_PROTOCAL                                                                                                                                                                                                |
| └──AGGREGATION (id=4) [finalize, merge]                                                                                                                                                                                     |
| │   Estimates: [row: 1, cpu: 8.00, memory: 8.00, network: 0.00, cost: 16503378.85]                                                                                                                                          |
| │   TotalTime: 61.624us (0.74%) [CPUTime: 61.624us]                                                                                                                                                                         |
| │   OutputRows: 1                                                                                                                                                                                                           |
| │   PeakMemory: 2.991 KB, AllocatedMemory: 4.632 KB                                                                                                                                                                         |
| │   AggExprs: [count(17: count)]                                                                                                                                                                                            |
| │   SubordinateOperators:                                                                                                                                                                                                    |
| │       LOCAL_EXCHANGE [Passthrough]                                                                                                                                                                                         |
| └──EXCHANGE (id=3) [GATHER]                                                                                                                                                                                                 |
| │       Estimates: [row: 1, cpu: 8.00, memory: 0.00, network: 8.00, cost: 16503358.85]                                                                                                                                      |
| │       TotalTime: 1.986ms (23.81%) [CPUTime: 112.324us, NetworkTime: 1.874ms]                                                                                                                                              |
| │       OutputRows: 10                                                                                                                                                                                                       |
| │       PeakMemory: 7.234 KB, AllocatedMemory: 60.896 KB                                                                                                                                                                     |
| │       Detail Timers:                                                                                                                                                                                                       |
| │           OverallTime: 1.290ms [min=1.125ms, max=1.406ms]                                                                                                                                                                  |
| │           WaitTime: 743.468us [min=230.363us, max=1.125ms]                                                                                                                                                                 |
| │                                                                                                                                                                                                                           |
| Fragment 1                                                                                                                                                                                                                    |
| │   BackendNum: 3                                                                                                                                                                                                           |
| │   InstancePeakMemoryUsage: 535.105 KB, InstanceAllocatedMemoryUsage: 115.046 MB                                                                                                                                          |
| │   PrepareTime: 1.055ms                                                                                                                                                                                                    |
| └──DATA_STREAM_SINK (id=3)                                                                                                                                                                                                  |
| │    PartitionType: UNPARTITIONED                                                                                                                                                                                           |
| └──AGGREGATION (id=2) [serialize, update]                                                                                                                                                                                   |
| │   Estimates: [row: 1, cpu: 3000607.50, memory: 0.80, network: 0.00, cost: 16503342.85]                                                                                                                                    |
| │   TotalTime: 172.182us (2.06%) [CPUTime: 172.182us]                                                                                                                                                                       |
| │   OutputRows: 10                                                                                                                                                                                                          |
| │   PeakMemory: 3.671 KB, AllocatedMemory: 46.327 KB                                                                                                                                                                        |
| │   AggExprs: [count(*)]                                                                                                                                                                                                    |
| └──PROJECT (id=1)                                                                                                                                                                                                           |
| │       Estimates: [row: ?, cpu: ?, memory: ?, network: ?, cost: ?]                                                                                                                                                          |
| │       TotalTime: 343.661us (4.12%) [CPUTime: 343.661us]                                                                                                                                                                    |
| │       OutputRows: 6.001M (6001215)                                                                                                                                                                                         |
| │       Expression: [1]                                                                                                                                                                                                      |
| └──OLAP_SCAN (id=0)                                                                                                                                                                                                         |
| │           Estimates: [row: 6001215, cpu: 30006075.00, memory: 0.00, network: 0.00, cost: 15003037.50]                                                                                                                      |
| │           TotalTime: 5.745ms (68.87%) [CPUTime: 1.256ms, ScanTime: 4.489ms]                                                                                                                                                |
| │           OutputRows: 6.001M (6001215)                                                                                                                                                                                     |
| │           PeakMemory: 2.525 MB, AllocatedMemory: 108.166 MB                                                                                                                                                               |
| │           Table: : lineitem                                                                                                                                                                                                |
| │           SubordinateOperators:                                                                                                                                                                                            |
| │               NOOP                                                                                                                                                                                                          |
| │               OLAP_SCAN_PREPARE                                                                                                                                                                                             |
| │           Detail Timers: [ScanTime = IOTaskExecTime + IOTaskWaitTime]                                                                                                                                                      |
| │               IOTaskExecTime: 3.703ms [min=3.224ms, max=4.434ms]                                                                                                                                                           |
| │                   SegmentRead: 3.110ms [min=2.646ms, max=3.808ms]                                                                                                                                                           |
| │                       BlockFetch: 2.944ms [min=2.493ms, max=3.616ms]                                                                                                                                                        |
| │               IOTaskWaitTime: 106.066us [min=41.862us, max=484.711us]                                                                                                                                                      |
| │                                                                                                                                                                                                                           |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
84 rows in set
Time: 0.018s
MySQL root@172.26.81.178:tpch>
```

Пример 2: Запрос Query Profile с указанием ID узла как `0`. Selena возвращает все детальные метрики для Node ID `0` и выделяет метрики с высоким использованием для упрощения определения проблем.

```
MySQL root@172.26.81.178:tpch> analyze profile from '35fc9ef0-8442-11ee-844d-00163e0f59de', 0;
+-----------------------
```

```
|                         GetDelVec: 0ns                                                                                                |
|                         GetDeltaColumnGroup: 6.124us [min=1.295us, max=10.804us]                                                      |
|                         GetRowsets: 305ns [min=223ns, max=396ns]                                                                      |
|                         IOTime: 160.874us [min=148.729us, max=171.534us]                                                              |
|                         ReadPKIndex: 0ns                                                                                              |
|                         SegmentInit: 45.173us [min=37.247us, max=51.996us]                                                            |
|                             BitmapIndexFilter: 0ns                                                                                    |
|                             BitmapIndexFilterRows: 0                                                                                  |
|                             BitmapIndexIteratorInit: 974ns [min=810ns, max=1.335us]                                                   |
|                             BloomFilterFilter: 0ns                                                                                    |
|                             BloomFilterFilterRows: 0                                                                                  |
|                             ColumnIteratorInit: 18.537us [min=11.481us, max=24.101us]                                                 |
|                             RemainingRowsAfterShortKeyFilter: 6.001M (6001215) [min=595.164K (595164), max=604.362K (604362)]         |
|                             SegmentRuntimeZoneMapFilterRows: 0                                                                        |
|                             SegmentZoneMapFilterRows: 0                                                                               |
|                             ShortKeyFilter: 1.012us [min=749ns, max=1.348us]                                                          |
|                             ShortKeyFilterRows: 0                                                                                     |
|                             ShortKeyRangeNumber: 0                                                                                    |
|                             ZoneMapIndexFilterRows: 0                                                                                 |
|                             ZoneMapIndexFiter: 727ns [min=528ns, max=1.096us]                                                         |
|                         SegmentRead: 3.110ms [min=2.646ms, max=3.808ms]                                                               |
|                             BlockFetch: 2.944ms [min=2.493ms, max=3.616ms]                                                            |
|                             BlockFetchCount: 1.470K (1470) [min=146, max=148]                                                         |
|                             BlockSeek: 122.611us [min=101.321us, max=145.219us]                                                       |
|                             BlockSeekCount: 370 [min=37, max=37]                                                                      |
|                             ChunkCopy: 0ns                                                                                            |
|                             DecompressT: 0ns                                                                                          |
|                             DelVecFilterRows: 0                                                                                       |
|                             PredFilter: 0ns                                                                                           |
|                             PredFilterRows: 0                                                                                         |
|                             RowsetsReadCount: 20 [min=2, max=2]                                                                       |
|                             SegmentsReadCount: 10 [min=1, max=1]                                                                      |
|                             TotalColumnsDataPageCount: 370 [min=37, max=37]                                                           |
|                     IOTaskWaitTime: 106.066us [min=41.862us, max=484.711us]                                                           |
|                     MorselsCount: 10 [min=1, max=1]                                                                                   |
|                     PeakChunkBufferMemoryUsage: 128.000 KB                                                                            |
|                     PeakChunkBufferSize: 11                                                                                           |
|                     PeakIOTasks: 1                                                                                                    |
|                     PeakScanTaskQueueSize: 6 [min=0, max=2]                                                                           |
|                     PrepareChunkSourceTime: 169.748us [min=151.592us, max=184.497us]                                                  |
|                     PushdownPredicates: 0                                                                                             |
|                     RawRowsRead: 6.001M (6001215) [min=595.164K (595164), max=604.362K (604362)]                                      |
|                     ReadPagesNum: 370 [min=37, max=37]                                                                                |
|                     RowsRead: 6.001M (6001215) [min=595.164K (595164), max=604.362K (604362)]                                         |
|                     SubmitTaskCount: 30 [min=3, max=3]                                                                                |
|                     SubmitTaskTime: 24.403us [min=14.188us, max=41.869us]                                                             |
|                     TabletCount: 10 [min=3, max=4]                                                                                    |
|                     UncompressedBytesRead: 5.939 MB [min=604.067 KB, max=611.704 KB]                                                  |
|                                                                                                                                       |
+-----------------------------------------------------------------------------------------------------------------------------------------+
141 rows in set
Time: 0.025s
MySQL root@172.26.81.178:tpch>
```

Кроме того, описанные выше методы также поддерживают отображение и анализ Runtime Query Profile, то есть Profile, сгенерированных для выполняющихся запросов. Когда функция Query Profile включена, вы можете использовать этот метод для получения Profile запросов, которые в данный момент выполняются более 10 секунд.

По сравнению с Profile завершенных запросов, текстовый Query Profile для выполняющихся запросов содержит следующую информацию:

- Статус оператора:
  - ⏳: Операторы не запущены. Эти операторы могут не запуститься из-за отношений зависимости.
  - □: Выполняющиеся операторы.
  - □: Операторы, которые завершили выполнение.
- Общий прогресс: Вычисляется на основе `количество операторов, завершивших выполнение / общее количество операторов`. Из-за отсутствия детальной информации о строках данных это значение может быть немного искажено.

- Прогресс оператора: Вычисляется на основе `количество обработанных строк / общее количество строк`. Если общее количество строк не может быть вычислено, прогресс отображается как `?`.

Пример:

```
MySQL root@172.26.81.178:(none)> show profilelist limit 5;
+--------------------------------------+---------------------+-------+----------+-----------------------------------------
```

```
|       SORT (id=1) [ROW_NUMBER, SORT] ✅
|         Estimates: [row: 100000000, cpu: 400000000.00, memory: 400000000.00, network: 400000000.00, cost: 0.00]
|         TotalTime: 2s572ms (13.75%) [CPUTime: 2s572ms]
|         OutputRows: 100.000M (100000000)
|         PeakMemory: 102.787 MB, AllocatedMemory: 14.161 GB
|         OrderByExprs: [<slot 8> 8: fn_int]
|         SubordinateOperators:
|             LOCAL_EXCHANGE [Passthrough]
|       └─OLAP_SCAN (id=7)  ✅
|             Estimates: [row: 100000000, cpu: 400000000.00, memory: 0.00, network: 0.00, cost: 200000000.00]
|             TotalTime: 158.605ms (0.85%) [CPUTime: 48.880ms, ScanTime: 109.725ms]
|             OutputRows: 100.000M (100000000)
|             PeakMemory: 38.408 MB, AllocatedMemory: 1.821 GB
|             Table: : t_large
|             SubordinateOperators:
|                 NOOP
|                 OLAP_SCAN_PREPARE
|
| Fragment 4
|   BackendNum: 1
|   InstancePeakMemoryUsage: 56.469 MB, InstanceAllocatedMemoryUsage: 3.919 GB
|   PrepareTime: 736.558us
| └─DATA_STREAM_SINK (id=6)
|     PartitionType: HASH_PARTITIONED
|     PartitionExprs: [7: max(1: fn_int)]
|   └─PROJECT (id=5)  🚀
|       Estimates: [row: ?, cpu: ?, memory: ?, network: ?, cost: ?]
|       TotalTime: 7.720ms (0.04%) [CPUTime: 7.720ms]
|       OutputRows: 31.277M (31277054)
|       Progress (processed rows/total rows): 31.28%
|       Expression: [7: max(1: fn_int)]
|       └─SELECT (id=4)  🚀
|           Estimates: [row: 100000000, cpu: 0.00, memory: 0.00, network: 0.00, cost: 1400000000.00]
|           TotalTime: 15.959ms (0.09%) [CPUTime: 15.959ms]
|           OutputRows: 31.277M (31277054)
|           Progress (processed rows/total rows): 31.28%
|           Predicates: [7: max(1: fn_int) IS NOT NULL]
|           └─ANALYTIC_EVAL (id=3)  🚀
|               Estimates: [row: 100000000, cpu: 800000000.00, memory: 0.00, network: 0.00, cost: 1400000000.00]
|               TotalTime: 1s18ms (5.45%) [CPUTime: 1s18ms]
|               OutputRows: 31.277M (31277056)
|               PeakMemory: 162.844 MB, AllocatedMemory: 172.471 MB
|               Progress (processed rows/total rows): 31.28%
|               Functions: [max(1: fn_int)]
|               OrderByExprs: [1: fn_int]
|               └─MERGE_EXCHANGE (id=2) [GATHER] ✅
|                   Estimates: [row: 100000000, cpu: 400000000.00, memory: 400000000.00, network: 400000000.00, cost: 0.00]
|                   TotalTime: 2s214ms (11.84%) [CPUTime: 2s198ms, NetworkTime: 15.883ms]
|                   OutputRows: 100.000M (100000000)
|                   PeakMemory: 152.056 MB, AllocatedMemory: 3.641 GB
|                   SubordinateOperators:
|                       LOCAL_EXCHANGE [Passthrough]
|
| Fragment 5
|   BackendNum: 3
|   InstancePeakMemoryUsage: 258.960 MB, InstanceAllocatedMemoryUsage: 16.819 GB
|   PrepareTime: 1.374ms
| └─DATA_STREAM_SINK (id=2)
|     PartitionType: UNPARTITIONED
|   └─SORT (id=1) [ROW_NUMBER, SORT] ✅
|       Estimates: [row: 100000000, cpu: 400000000.00, memory: 400000000.00, network: 400000000.00, cost: 0.00]
|       TotalTime: 2s395ms (12.81%) [CPUTime: 2s395ms]
|       OutputRows: 100.000M (100000000)
|       PeakMemory: 103.850 MB, AllocatedMemory: 14.143 GB
|       OrderByExprs: [<slot 1> 1: fn_int]
|       SubordinateOperators:
|           LOCAL_EXCHANGE [Passthrough]
|     └─OLAP_SCAN (id=0)  ✅
|           Estimates: [row: 100000000, cpu: 400000000.00, memory: 0.00, network: 0.00, cost: 200000000.00]
|           TotalTime: 142.250ms (0.76%) [CPUTime: 28.383ms, ScanTime: 113.867ms]
|           OutputRows: 100.000M (100000000)
|           PeakMemory: 40.218 MB, AllocatedMemory: 1.821 GB
|           Table: : t_large
|           SubordinateOperators:
|               NOOP
|               OLAP_SCAN_PREPARE
|
+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
197 rows in set
Time: 0.033s
MySQL root@172.26.81.178:(none)>
```

Симуляция запроса для анализа Profile с помощью EXPLAIN ANALYZE

Selena предоставляет оператор EXPLAIN ANALYZE, позволяющий симулировать и анализировать profile запроса напрямую. Синтаксис следующий:

```
EXPLAIN ANALYZE <sql_statement>
```

При выполнении `EXPLAIN ANALYZE` Selena по умолчанию включит функцию Query Profile для текущей сессии.

В настоящее время `EXPLAIN ANALYZE` поддерживает два типа SQL-операторов: операторы SELECT и операторы INSERT INTO. Вы можете симулировать и анализировать Query Profile операторов INSERT INTO только для внутренних таблиц в каталоге по умолчанию Selena. Обратите внимание, что при симуляции и анализе Query Profile

операторов INSERT INTO фактические данные не будут загружены. По умолчанию транзакция импорта будет отменена, чтобы гарантировать, что во время анализа не будут внесены непреднамеренные изменения в данные.

Пример 1: Симуляция и анализ оператора SELECT. Результаты запроса отбрасываются.

```
MySQL root@172.26.81.178:tpch> explain analyze select count(*) from lineitem;
+-----------------------
```

Пример 2: Симуляция и анализ оператора INSERT INTO. Транзакция загрузки будет отменена.

```
MySQL root@172.26.81.178:tpch> select count(*) from lineitem;
+----------+
| count(*) |
+----------+
| 6001215  |
+----------+
1 row in set
Time: 0.018s
MySQL root@172.26.81.178:tpch> explain analyze insert into lineitem select * from lineitem;
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Explain String                                                
```

использовать клиент MyCLI. Для клиентов, которые не поддерживают функции ANSI, такие как клиент MySQL, могут возникнуть некоторые незначительные нарушения отображения. Обычно они не влияют на использование. Например:

```
mysql> analyze profile from '35fc9ef0-8442-11ee-844d-00163e0f59de';
+-----------------------
```

1.8.7 Query Hint

Query hint — это директивы или комментарии, которые явно подсказывают оптимизатору запросов, как выполнять запрос. В настоящее время Selena поддерживает три типа hint: hint системных переменных (`SET_VAR`), hint пользовательских переменных (`SET_USER_VARIABLE`) и Join hint. Hint действуют только в пределах одного запроса.

Hint системных переменных

Вы можете использовать hint `SET_VAR` для установки одной или нескольких системных переменных в операторах SELECT и SUBMIT TASK, а затем выполнить эти операторы. Вы также можете использовать hint `SET_VAR` в предложении SELECT, включенном в другие операторы, такие как CREATE MATERIALIZED VIEW AS SELECT и CREATE VIEW AS SELECT. Обратите внимание, что если hint `SET_VAR` используется в предложении SELECT в CTE, то hint `SET_VAR` не действует, даже если оператор выполнен успешно.

По сравнению с общим использованием системных переменных, которое действует на уровне сессии, hint `SET_VAR` действует на уровне оператора и не влияет на всю сессию.

Синтаксис

```
[...] SELECT /*+ SET_VAR(key=value [, key = value]) */ ...
SUBMIT [/*+ SET_VAR(key=value [, key = value]) */] TASK ...
```

Примеры

Чтобы указать режим агрегации для агрегирующего запроса, используйте hint `SET_VAR` для установки системных переменных `streaming_preaggregation_mode` и `new_planner_agg_stage` в агрегирующем запросе.

```
SELECT /*+ SET_VAR (streaming_preaggregation_mode =
'force_streaming',new_planner_agg_stage = '2') */ SUM(sales_amount) AS
total_sales_amount FROM sales_orders;
```

Чтобы указать таймаут выполнения для оператора SUBMIT TASK, используйте hint `SET_VAR` для установки системной переменной `insert_timeout` в операторе SUBMIT TASK.

```
SUBMIT /*+ SET_VAR(insert_timeout=3) */ TASK AS CREATE TABLE temp AS SELECT
count(*) AS cnt FROM tbl1;
```

Чтобы указать таймаут выполнения подзапроса при создании материализованного представления, используйте hint `SET_VAR` для установки системной переменной `query_timeout` в предложении SELECT.

```
CREATE MATERIALIZED VIEW mv
PARTITION BY dt
DISTRIBUTED BY HASH(`key`)
BUCKETS 10
REFRESH ASYNC
AS SELECT /*+ SET_VAR(query_timeout=500) */ * from dual;
```

Hint пользовательских переменных

Вы можете использовать hint `SET_USER_VARIABLE` для установки одной или нескольких пользовательских переменных в операторах SELECT или INSERT. Если другие операторы содержат предложение SELECT, вы также можете использовать hint `SET_USER_VARIABLE` в этом предложении SELECT. Другими операторами могут быть операторы SELECT и INSERT, но не операторы CREATE MATERIALIZED VIEW AS SELECT и CREATE VIEW AS SELECT. Обратите внимание, что если hint `SET_USER_VARIABLE` используется в предложении SELECT в CTE, то hint `SET_USER_VARIABLE` не действует, даже если оператор выполнен успешно. Начиная с версии v1.5.2, Selena поддерживает hint пользовательских переменных.

По сравнению с общим использованием пользовательских переменных, которое действует на уровне сессии, hint `SET_USER_VARIABLE` действует на уровне оператора и не влияет на всю сессию.

Синтаксис

```
[...] SELECT /*+ SET_USER_VARIABLE(@var_name = expr [, @var_name = expr])
*/ ...
INSERT /*+ SET_USER_VARIABLE(@var_name = expr [, @var_name = expr]) */ ...
```

Примеры

Следующий оператор SELECT ссылается на скалярные подзапросы `select max(age) from users` и `select min(name) from users`, поэтому вы можете использовать hint `SET_USER_VARIABLE` для установки этих двух скалярных подзапросов в качестве пользовательских переменных, а затем выполнить запрос.

```
SELECT /*+ SET_USER_VARIABLE (@a = (select max(age) from users), @b =
(select min(name) from users)) */ * FROM sales_orders where
sales_orders.age = @a and sales_orders.name = @b;
```

Join hint

Для многотабличных Join-запросов оптимизатор обычно выбирает оптимальный метод выполнения Join. В особых случаях вы можете использовать Join hint, чтобы явно подсказать оптимизатору метод выполнения Join или отключить Join Reorder. В настоящее время Join hint поддерживает указание Shuffle Join, Broadcast Join, Bucket Shuffle Join или Colocate Join в качестве метода выполнения Join. При использовании Join hint оптимизатор не выполняет Join Reorder. Поэтому вам нужно выбрать меньшую таблицу в качестве правой таблицы. Кроме того, при указании Colocate Join или Bucket Shuffle Join в качестве метода выполнения Join убедитесь, что распределение данных объединяемой таблицы соответствует требованиям этих методов выполнения Join. В противном случае предложенный метод выполнения Join не сможет вступить в силу.

Синтаксис

```
... JOIN { [BROADCAST] | [SHUFFLE] | [BUCKET] | [COLOCATE] | [UNREORDER]}
...
```

> **ПРИМЕЧАНИЕ**
>
> Join Hint не чувствителен к регистру.

Примеры

- Shuffle Join

Если вам нужно перемешать строки данных с одинаковыми значениями ключа бакетирования из таблиц A и B на одной машине перед выполнением операции Join, вы можете указать метод выполнения Join как Shuffle Join.

```
select k1 from t1 join [SHUFFLE] t2 on t1.k1 = t2.k2 group by t2.k2;
```

- Broadcast Join

  Если таблица A — большая таблица, а таблица B — маленькая таблица, вы можете указать метод выполнения Join как Broadcast Join. Данные таблицы B полностью транслируются на машины, на которых находятся данные таблицы A, а затем выполняется операция Join. По сравнению с Shuffle Join, Broadcast Join экономит затраты на перемешивание данных таблицы A.

```
select k1 from t1 join [BROADCAST] t2 on t1.k1 = t2.k2 group by t2.k2;
```

- Bucket Shuffle Join

  Если выражение эквисоединения Join в запросе Join содержит ключ бакетирования таблицы A, особенно когда обе таблицы A и B являются большими таблицами, вы можете указать метод выполнения Join как Bucket Shuffle Join. Данные таблицы B перемешиваются на машины, на которых находятся данные таблицы A, в соответствии с распределением данных таблицы A, а затем выполняется операция Join. По сравнению с Broadcast Join, Bucket Shuffle Join значительно сокращает передачу данных, поскольку данные таблицы B перемешиваются только один раз глобально. Таблицы, участвующие в Bucket Shuffle Join, должны быть либо несекционированными, либо размещенными совместно (colocated).

```
select k1 from t1 join [BUCKET] t2 on t1.k1 = t2.k2 group by t2.k2;
```

- Colocate Join

  Если таблицы A и B принадлежат к одной Colocation Group, которая указана при создании таблицы, строки данных с одинаковыми значениями ключа бакетирования из таблиц A и B распределяются на одном узле BE. Когда выражение эквисоединения Join содержит ключ бакетирования таблиц A и B в запросе Join, вы можете указать

метод выполнения Join как Colocate Join. Данные с одинаковыми значениями ключей объединяются напрямую локально, сокращая время, затрачиваемое на передачу данных между узлами, и повышая производительность запросов.

```
select k1 from t1 join [COLOCATE] t2 on t1.k1 = t2.k2 group by t2.k2;
```

Просмотр метода выполнения Join

Используйте команду `EXPLAIN` для просмотра фактического метода выполнения Join. Если возвращенный результат показывает, что метод выполнения Join соответствует Join hint, это означает, что Join hint эффективен.

```
EXPLAIN select k1 from t1 join [COLOCATE] t2 on t1.k1 = t2.k2 group by
t2.k2;
```

```
2:HASH JOIN
|  join op: INNER JOIN (COLOCATE)
|  colocate: true
|  equal join conjunct: 2: k2 = 5: k2
|
|----1:OlapScanNode
|       TABLE: tbl2
|       PREAGGREGATION: OFF. Reason: Has can not pre-aggregation Join
|       partitions=1/1
|       rollup: tbl2
|       tabletRatio=8/8
|       tabletList=79149977,79149981,79149985,79149989,79149993,79149997,79150001,79150005
|       cardinality=1
|       avgRowSize=13.0
|       numNodes=0
|
```

1.8.8 Метрики Query Profile

> Авторитетный справочник по необработанным метрикам, выдаваемым **Selena Query Profile**, сгруппированным по операторам. Используйте его как глоссарий; для руководства по устранению неполадок переходите к **query_profile_tuning_recipes.md**.

Сводные метрики

Базовая информация о выполнении запроса:

Метрика	Описание
Total	Общее время, затраченное на запрос, включая продолжительность фаз Planning, Executing и Profiling.
Query State	Состояние запроса, возможные состояния включают Finished, Error и Running.
Query ID	Уникальный идентификатор запроса.
Start Time	Временная метка начала запроса.
End Time	Временная метка окончания запроса.
Total	Общая продолжительность запроса.
Query Type	Тип запроса.
Query State	Текущее состояние запроса.
Selena Version	Используемая версия Selena.
User	Пользователь, выполнивший запрос.
Default Db	База данных по умолчанию, используемая для запроса.
Sql Statement	Выполненная SQL-инструкция.
Variables	Важные переменные, используемые для запроса.
NonDefaultSessionVariables	Не по умолчанию сессионные переменные, используемые для запроса.
Collect Profile Time	Время, затраченное на сбор profile.
IsProfileAsync	Указывает, был ли сбор profile асинхронным.

Метрики Planner

Предоставляет комплексный обзор planner. Обычно, если общее время, затраченное на planner, меньше 10 мс, это не вызывает беспокойства.

В некоторых сценариях planner может требовать больше времени:

1. Сложные запросы могут требовать дополнительного времени для парсинга и оптимизации для обеспечения оптимального плана выполнения.
2. Наличие многочисленных materialized views может увеличить время, необходимое для переписывания запроса.
3. Когда множество одновременных запросов исчерпывают системные ресурсы и используется очередь запросов, время `Pending` может быть продолжительным.
4. Запросы, включающие внешние таблицы, могут требовать дополнительного времени для связи с внешним сервером метаданных.

Пример:

```
    - -- Parser[1] 0
    - -- Total[1] 3ms
    -     -- Analyzer[1] 0
    -         -- Lock[1] 0
    -         -- AnalyzeDatabase[1] 0
    -         -- AnalyzeTemporaryTable[1] 0
    -         -- AnalyzeTable[1] 0
    -     -- Transformer[1] 0
    -     -- Optimizer[1] 1ms
    -         -- MVPreprocess[1] 0
    -         -- MVTextRewrite[1] 0
    -         -- RuleBaseOptimize[1] 0
    -         -- CostBaseOptimize[1] 0
    -         -- PhysicalRewrite[1] 0
    -         -- DynamicRewrite[1] 0
    -         -- PlanValidate[1] 0
    -             -- InputDependenciesChecker[1] 0
    -             -- TypeChecker[1] 0
    -             -- CTEUniqueChecker[1] 0
    -             -- ColumnReuseChecker[1] 0
    -     -- ExecPlanBuild[1] 0
    - -- Pending[1] 0
    - -- Prepare[1] 0
    - -- Deploy[1] 2ms
    -     -- DeployLockInternalTime[1] 2ms
    -         -- DeploySerializeConcurrencyTime[2] 0
    -         -- DeployStageByStageTime[6] 0
    -         -- DeployWaitTime[6] 1ms
    -             -- DeployAsyncSendTime[2] 0
    - DeployDataSize: 10916
   Reason:
```

Метрики обзора выполнения

Высокоуровневая статистика выполнения:

Метрика	Описание	Эмпирическое правило
FrontendProfileMergeTime	Время обработки profile на стороне FE	< 10 мс нормально
QueryAllocatedMemoryUsage	Общая выделенная память по узлам	
QueryDeallocatedMemoryUsage	Общая освобожденная память по узлам	

Метрика	Описание	Эмпирическое правило
QueryPeakMemoryUsagePerNode	Максимальная пиковая память на узел	< 80% емкости нормально
QuerySumMemoryUsage	Общая пиковая память по узлам	
QueryExecutionWallTime	Астрономическое время выполнения	
QueryCumulativeCpuTime	Общее время CPU по узлам	Сравните с `walltime * totalCpuCores`
QueryCumulativeOperatorTime	Общее время выполнения операторов	Знаменатель для процентов времени операторов
QueryCumulativeNetworkTime	Общее сетевое время Exchange node	
QueryCumulativeScanTime	Общее время IO Scan node	
QueryPeakScheduleTime	Максимальное ScheduleTime Pipeline	< 1 с нормально для простых запросов
QuerySpillBytes	Данные, выгруженные на диск	< 1 ГБ нормально

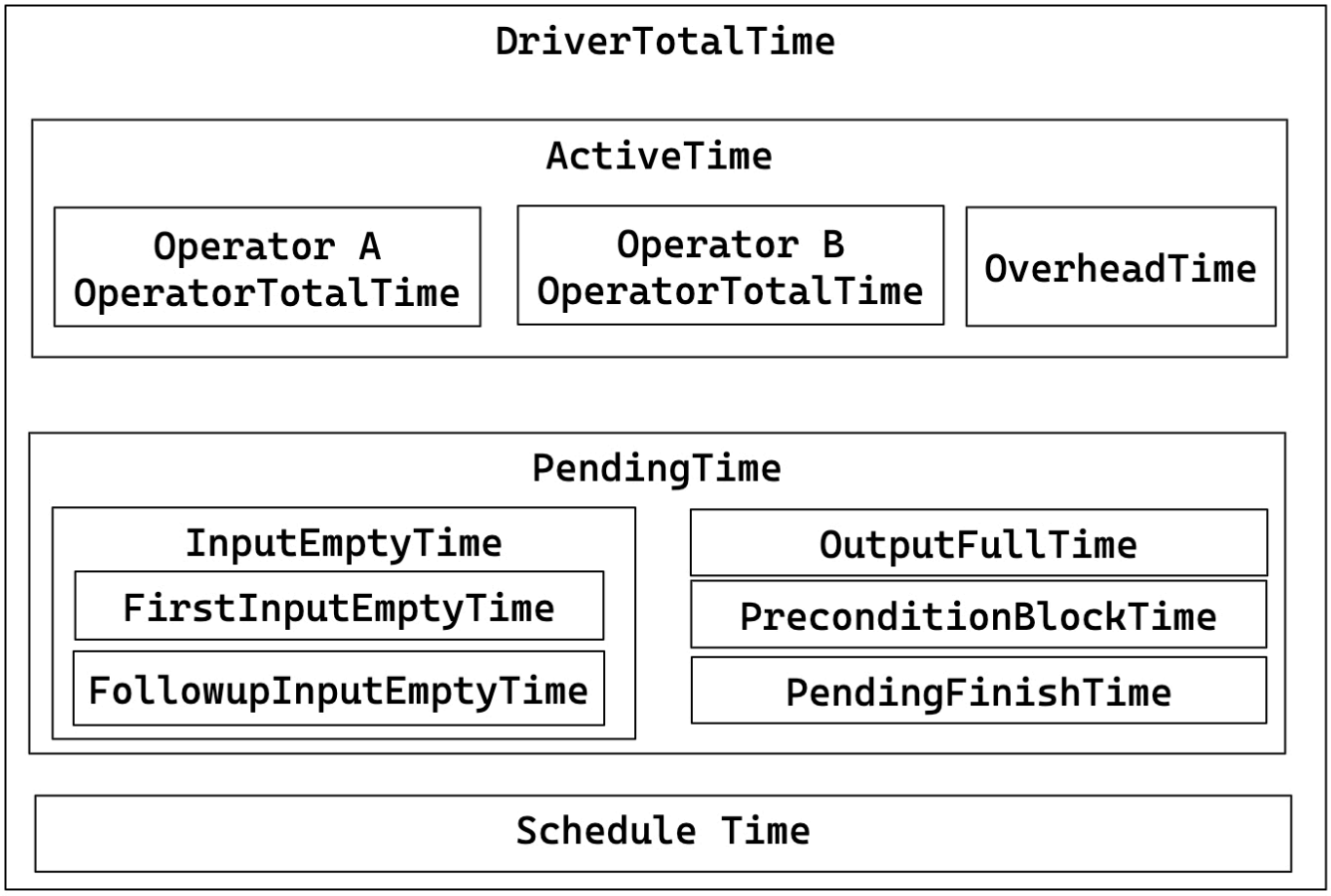
Метрики Fragment

Детали выполнения на уровне Fragment:

Метрика	Описание
InstanceNum	Количество FragmentInstances
InstanceIds	ID всех FragmentInstances
BackendNum	Количество участвующих BE
BackendAddresses	Адреса BE
FragmentInstancePrepareTime	Продолжительность фазы Prepare Fragment
InstanceAllocatedMemoryUsage	Общая выделенная память для instances
InstanceDeallocatedMemoryUsage	Общая освобожденная память для instances
InstancePeakMemoryUsage	Пиковая память по instances

Метрики Pipeline

Детали выполнения Pipeline и взаимосвязи:



Ключевые взаимосвязи:

- DriverTotalTime = ActiveTime + PendingTime + ScheduleTime
- ActiveTime = ∑ OperatorTotalTime + OverheadTime
- PendingTime = InputEmptyTime + OutputFullTime + PreconditionBlockTime + PendingFinishTime
- InputEmptyTime = FirstInputEmptyTime + FollowupInputEmptyTime

Метрика	Описание
DegreeOfParallelism	Степень параллелизма выполнения pipeline.
TotalDegreeOfParallelism	Сумма степеней параллелизма. Поскольку один и тот же Pipeline может выполняться на нескольких машинах, этот элемент агрегирует все значения.
DriverPrepareTime	Время, затраченное на фазу Prepare. Эта метрика не включена в DriverTotalTime.
DriverTotalTime	Общее время выполнения Pipeline, исключая время, проведенное в фазе Prepare.

Метрика	Описание
ActiveTime	Время выполнения Pipeline, включая время выполнения каждого оператора и общие накладные расходы фреймворка, такие как время, потраченное на вызов методов типа has_output, need_input и т.д.
PendingTime	Время, когда Pipeline заблокирован от планирования по различным причинам.
InputEmptyTime	Время, когда Pipeline заблокирован из-за пустой входной очереди.
FirstInputEmptyTime	Время, когда Pipeline впервые заблокирован из-за пустой входной очереди. Первое время блокировки рассчитывается отдельно, потому что первая блокировка в основном вызвана зависимостями Pipeline.
FollowupInputEmptyTime	Время, когда Pipeline впоследствии заблокирован из-за пустой входной очереди.
OutputFullTime	Время, когда Pipeline заблокирован из-за полной выходной очереди.
PreconditionBlockTime	Время, когда Pipeline заблокирован из-за невыполненных зависимостей.
PendingFinishTime	Время, когда Pipeline заблокирован в ожидании завершения асинхронных задач.
ScheduleTime	Время планирования Pipeline, от входа в очередь готовности до планирования на выполнение.
BlockByInputEmpty	Количество раз, когда pipeline заблокирован из-за InputEmpty.
BlockByOutputFull	Количество раз, когда pipeline заблокирован из-за OutputFull.
BlockByPrecondition	Количество раз, когда pipeline заблокирован из-за невыполненных предусловий.

Метрики Operator

Метрика	Описание
PrepareTime	Время, затраченное на подготовку.
OperatorTotalTime	Общее время, затраченное Operator. Оно удовлетворяет уравнению: OperatorTotalTime = PullTotalTime + PushTotalTime + SetFinishingTime + SetFinishedTime + CloseTime. Исключает время, потраченное на подготовку.
PullTotalTime	Общее время, которое Operator тратит на выполнение push_chunk.
PushTotalTime	Общее время, которое Operator тратит на выполнение pull_chunk.

Метрика	Описание
SetFinishingTime	Общее время, которое Operator тратит на выполнение set_finishing.
SetFinishedTime	Общее время, которое Operator тратит на выполнение set_finished.
PushRowNum	Кумулятивное количество входных строк для Operator.
PullRowNum	Кумулятивное количество выходных строк для Operator.
JoinRuntimeFilterEvaluate	Количество раз оценки Join Runtime Filter.
JoinRuntimeFilterHashTime	Время, затраченное на вычисление hash для Join Runtime Filter.
JoinRuntimeFilterInputRows	Количество входных строк для Join Runtime Filter.
JoinRuntimeFilterOutputRows	Количество выходных строк для Join Runtime Filter.
JoinRuntimeFilterTime	Время, затраченное на Join Runtime Filter.

Scan Operator

OLAP Scan Operator

OLAP_SCAN Operator отвечает за чтение данных из нативных таблиц Selena.

Метрика	Описание
Table	Имя таблицы.
Rollup	Имя materialized view. Если materialized view не задействован, эквивалентно имени таблицы.
SharedScan	Включена ли сессионная переменная enable_shared_scan.
TabletCount	Количество tablets.
MorselsCount	Количество morsels, которые являются базовой единицей выполнения IO.
PushdownPredicates	Количество pushdown predicates.
Predicates	Выражения предикатов.
BytesRead	Размер прочитанных данных.
CompressedBytesRead	Размер сжатых данных, прочитанных с диска.
UncompressedBytesRead	Размер несжатых данных, прочитанных с диска.

Метрика	Описание
RowsRead	Количество прочитанных строк (после фильтрации предикатов).
RawRowsRead	Количество необработанных прочитанных строк (до фильтрации предикатов).
ReadPagesNum	Количество прочитанных страниц.
CachedPagesNum	Количество кэшированных страниц.
ChunkBufferCapacity	Емкость Chunk Buffer.
DefaultChunkBufferCapacity	Емкость Chunk Buffer по умолчанию.
PeakChunkBufferMemoryUsage	Пиковое использование памяти Chunk Buffer.
PeakChunkBufferSize	Пиковый размер Chunk Buffer.
PrepareChunkSourceTime	Время, затраченное на подготовку Chunk Source.
ScanTime	Кумулятивное время сканирования. Операции сканирования выполняются в асинхронном пуле потоков I/O.
IOTaskExecTime	Время выполнения IO задач.
IOTaskWaitTime	Время ожидания от успешной отправки до запланированного выполнения IO задач.
SubmitTaskCount	Количество раз отправки IO задач.
SubmitTaskTime	Время, затраченное на отправку задач.
PeakIOTasks	Пиковое количество IO задач.
PeakScanTaskQueueSize	Пиковый размер очереди IO задач.

Connector Scan Operator

Похож на OLAP_SCAN operator, но используется для сканирования внешних таблиц, таких как Iceberg/Hive/Hudi/Delta.

Метрика	Описание
DataSourceType	Тип источника данных, может быть HiveDataSource, ESDataSource и так далее.
Table	Имя таблицы.
TabletCount	Количество tablets.
MorselsCount	Количество morsels.
Predicates	Выражение предиката.
PredicatesPartition	Выражение предиката, применяемое к partitions.
SharedScan	Включена ли сессионная переменная `enable_shared_scan`.
ChunkBufferCapacity	Емкость Chunk Buffer.

Метрика	Описание
DefaultChunkBufferCapacity	Емкость Chunk Buffer по умолчанию.
PeakChunkBufferMemoryUsage	Пиковое использование памяти Chunk Buffer.
PeakChunkBufferSize	Пиковый размер Chunk Buffer.
PrepareChunkSourceTime	Время, затраченное на подготовку Chunk Source.
ScanTime	Кумулятивное время сканирования. Операция сканирования выполняется в асинхронном пуле потоков I/O.
IOTaskExecTime	Время выполнения I/O задач.
IOTaskWaitTime	Время ожидания от успешной отправки до запланированного выполнения IO задач.
SubmitTaskCount	Количество раз отправки IO задач.
SubmitTaskTime	Время, затраченное на отправку задач.
PeakIOTasks	Пиковое количество IO задач.
PeakScanTaskQueueSize	Пиковый размер очереди IO задач.

Exchange Operator

Exchange Operator отвечает за передачу данных между BE узлами. Может быть несколько типов операций exchange: GATHER/BROADCAST/SHUFFLE.

Типичные сценарии, которые могут сделать Exchange Operator узким местом запроса:

1. Broadcast Join: Это подходящий метод для небольшой таблицы. Однако в исключительных случаях, когда оптимизатор выбирает неоптимальный план запроса, это может привести к значительному увеличению сетевой пропускной способности.
2. Shuffle Aggregation/Join: Перемешивание большой таблицы может привести к значительному увеличению сетевой пропускной способности.

Exchange Sink Operator

Метрика	Описание
ChannelNum	Количество каналов. Обычно количество каналов равно количеству получателей.
DestFragments	Список ID целевых FragmentInstance.
DestID	ID целевого узла.
PartType	Режим распределения данных, включая: UNPARTITIONED, RANDOM, HASH_PARTITIONED и BUCKET_SHUFFLE_HASH_PARTITIONED.

Метрика	Описание
SerializeChunkTime	Время, затраченное на сериализацию chunks.
SerializedBytes	Размер сериализованных данных.
ShuffleChunkAppendCounter	Количество операций Chunk Append, когда PartType является HASH_PARTITIONED или BUCKET_SHUFFLE_HASH_PARTITIONED.
ShuffleChunkAppendTime	Время, затраченное на операции Chunk Append, когда PartType является HASH_PARTITIONED или BUCKET_SHUFFLE_HASH_PARTITIONED.
ShuffleHashTime	Время, затраченное на вычисление hash, когда PartType является HASH_PARTITIONED или BUCKET_SHUFFLE_HASH_PARTITIONED.
RequestSent	Количество отправленных пакетов данных.
RequestUnsent	Количество неотправленных пакетов данных. Эта метрика ненулевая, когда есть логика короткого замыкания; в противном случае она равна нулю.
BytesSent	Размер отправленных данных.
BytesUnsent	Размер неотправленных данных. Эта метрика ненулевая, когда есть логика короткого замыкания; в противном случае она равна нулю.
BytesPassThrough	Если целевой узел является текущим узлом, данные не будут передаваться по сети, что называется passthrough данными. Эта метрика указывает размер таких passthrough данных. Passthrough управляется `enable_exchange_pass_through`.
PassThroughBufferPeakMemoryUsage	Пиковое использование памяти PassThrough Buffer.
CompressTime	Время сжатия.
CompressedBytes	Размер сжатых данных.
OverallThroughput	Скорость пропускной способности.
NetworkTime	Время, затраченное на передачу пакетов данных (исключая время обработки после получения).
NetworkBandwidth	Расчетная пропускная способность сети.
WaitTime	Время ожидания из-за полной очереди отправителя.
OverallTime	Общее время для всего процесса передачи, т.е. от отправки первого пакета данных до подтверждения правильного получения последнего пакета данных.
RpcAvgTime	Среднее время для RPC.
RpcCount	Общее количество RPC.

Exchange Source Operator

Метрика	Описание
RequestReceived	Размер полученных пакетов данных.
BytesReceived	Размер полученных данных.
DecompressChunkTime	Время, затраченное на распаковку chunks.
DeserializeChunkTime	Время, затраченное на десериализацию chunks.
ClosureBlockCount	Количество заблокированных Closures.
ClosureBlockTime	Время блокировки для Closures.
ReceiverProcessTotalTime	Общее время, затраченное на обработку на стороне получателя.
WaitLockTime	Время ожидания блокировки.

Aggregate Operator

Список метрик

Метрика	Описание
`GroupingKeys`	Столбцы `GROUP BY`.
`AggregateFunctions`	Время, затраченное на вычисления агрегатных функций.
`AggComputeTime`	Время для AggregateFunctions + Group By.
`ChunkBufferPeakMem`	Пиковое использование памяти Chunk Buffer.
`ChunkBufferPeakSize`	Пиковый размер Chunk Buffer.
`ExprComputeTime`	Время для вычисления выражений.
`ExprReleaseTime`	Время для освобождения выражений.
`GetResultsTime`	Время для извлечения агрегатных результатов.
`HashTableSize`	Размер Hash Table.
`HashTableMemoryUsage`	Размер памяти Hash Table.
`InputRowCount`	Количество входных строк.
`PassThroughRowCount`	В режиме Auto количество строк данных, обработанных в потоковом режиме после низкой агрегации, приводит к деградации до потокового режима.
`ResultAggAppendTime`	Время, затраченное на добавление столбцов агрегатных результатов.

Метрика	Описание
`ResultGroupByAppendTime`	Время, затраченное на добавление столбцов Group By.
`ResultIteratorTime`	Время для итерации по Hash Table.
`StreamingTime`	Время обработки в потоковом режиме.

Join Operator

Список метрик

Метрика	Описание
`DistributionMode`	Тип распределения, включая: BROADCAST, PARTITIONED, COLOCATE и т.д.
`JoinPredicates`	Предикаты Join.
`JoinType`	Тип Join.
`BuildBuckets`	Количество buckets в Hash Table.
`BuildKeysPerBucket`	Количество ключей на bucket в Hash Table.
`BuildConjunctEvaluateTime`	Время, затраченное на оценку conjunct во время фазы build.
`BuildHashTableTime`	Время, затраченное на построение Hash Table.
`ProbeConjunctEvaluateTime`	Время, затраченное на оценку conjunct во время фазы probe.
`SearchHashTableTimer`	Время, затраченное на поиск в Hash Table.
`CopyRightTableChunkTime`	Время, затраченное на копирование chunks из правой таблицы.
`OutputBuildColumnTime`	Время, затраченное на вывод столбца стороны build.
`OutputProbeColumnTime`	Время, затраченное на вывод столбца стороны probe.
`HashTableMemoryUsage`	Использование памяти Hash Table.
`RuntimeFilterBuildTime`	Время, затраченное на построение runtime filters.
`RuntimeFilterNum`	Количество runtime filters.

Window Function Operator

Метрика	Описание
`ProcessMode`	Режим выполнения, включающий две части: первая часть включает Materializing и Streaming; вторая часть включает Cumulative, RemovableCumulative, ByDefinition.
`ComputeTime`	Время, затраченное на вычисления оконных функций.
`PartitionKeys`	Столбцы Partition.
`AggregateFunctions`	Агрегатные функции.

Метрика	Описание
`ColumnResizeTime`	Время, затраченное на изменение размера столбцов.
`PartitionSearchTime`	Время, затраченное на поиск границ partition.
`PeerGroupSearchTime`	Время, затраченное на поиск границ Peer Group. Имеет смысл только когда тип окна `RANGE`.
`PeakBufferedRows`	Пиковое количество строк в буфере.
`RemoveUnusedRowsCount`	Количество раз удаления неиспользуемых буферов.
`RemoveUnusedTotalRows`	Общее количество строк, удаленных из неиспользуемых буферов.

Sort Operator

Метрика	Описание
`SortKeys`	Ключи сортировки.
`SortType`	Метод сортировки результатов запроса: полная сортировка или сортировка топ N результатов.
`MaxBufferedBytes`	Пиковый размер буферизованных данных.
`MaxBufferedRows`	Пиковое количество буферизованных строк.
`NumSortedRuns`	Количество отсортированных прогонов.
`BuildingTime`	Время, затраченное на поддержание внутренних структур данных во время сортировки.
`MergingTime`	Время, затраченное на слияние отсортированных прогонов во время сортировки.
`SortingTime`	Время, затраченное на сортировку.
`OutputTime`	Время, затраченное на построение выходной отсортированной последовательности.

Merge Operator

Метрика	Описание	Уровень
`Limit`	Limit.	Первичный
`Offset`	Offset.	Первичный
`StreamingBatchSize`	Размер данных, обрабатываемых за одну операцию Merge, когда Merge выполняется в режиме Streaming	Первичный

Метрика	Описание	Уровень
`LateMaterializationMaxBufferChunkNum`	Максимальное количество chunks в буфере при включенной поздней материализации.	Первичный
`OverallStageCount`	Общее количество выполнений всех стадий.	Первичный
`OverallStageTime`	Общее время выполнения для каждой стадии.	Первичный
`1-InitStageCount`	Количество выполнений стадии Init.	Вторичный
`2-PrepareStageCount`	Количество выполнений стадии Prepare.	Вторичный
`3-ProcessStageCount`	Количество выполнений стадии Process.	Вторичный
`4-SplitChunkStageCount`	Количество выполнений стадии SplitChunk.	Вторичный
`5-FetchChunkStageCount`	Количество выполнений стадии FetchChunk.	Вторичный
`6-PendingStageCount`	Количество выполнений стадии Pending.	Вторичный
`7-FinishedStageCount`	Количество выполнений стадии Finished.	Вторичный
`1-InitStageTime`	Время выполнения стадии Init.	Вторичный
`2-PrepareStageTime`	Время выполнения стадии Prepare.	Вторичный
`3-ProcessStageTime`	Время выполнения стадии Process.	Вторичный
`4-SplitChunkStageTime`	Время, затраченное на стадию Split.	Вторичный
`5-FetchChunkStageTime`	Время, затраченное на стадию Fetch.	Вторичный
`6-PendingStageTime`	Время, затраченное на стадию Pending.	Вторичный
`7-FinishedStageTime`	Время, затраченное на стадию Finished.	Вторичный
`LateMaterializationGenerateOrdinalTime`	Время, затраченное на генерацию ordinal столбцов во время поздней материализации.	Третичный
`SortedRunProviderTime`	Время, затраченное на получение данных от provider во время стадии Process.	Третичный

TableFunction Operator

Метрика	Описание
`TableFunctionExecTime`	Время вычисления Table Function.
`TableFunctionExecCount`	Количество выполнений Table Function.

Project Operator

Project Operator отвечает за выполнение `SELECT <expr>`. Если в запросе есть некоторые дорогостоящие выражения, этот оператор может занимать значительное время.

Метрика	Описание
`ExprComputeTime`	Время вычисления выражений.
`CommonSubExprComputeTime`	Время вычисления общих подвыражений.

LocalExchange Operator

Метрика	Описание
Type	Тип Local Exchange, включая: `Passthrough`, `Partition` и `Broadcast`.
`ShuffleNum`	Количество shuffles. Эта метрика действительна только когда `Type` является `Partition`.
`LocalExchangePeakMemoryUsage`	Пиковое использование памяти.
`LocalExchangePeakBufferSize`	Пиковый размер буфера.
`LocalExchangePeakBufferMemoryUsage`	Пиковое использование памяти буфера.
`LocalExchangePeakBufferChunkNum`	Пиковое количество chunks в буфере.
`LocalExchangePeakBufferRowNum`	Пиковое количество строк в буфере.
`LocalExchangePeakBufferBytes`	Пиковый размер данных в буфере.
`LocalExchangePeakBufferChunkSize`	Пиковый размер chunks в буфере.
`LocalExchangePeakBufferChunkRowNum`	Пиковое количество строк на chunk в буфере.
`LocalExchangePeakBufferChunkBytes`	Пиковый размер данных на chunk в буфере.

OlapTableSink Operator

OlapTableSink Operator отвечает за выполнение операции `INSERT INTO <table>`.

ПОДСКАЗКА

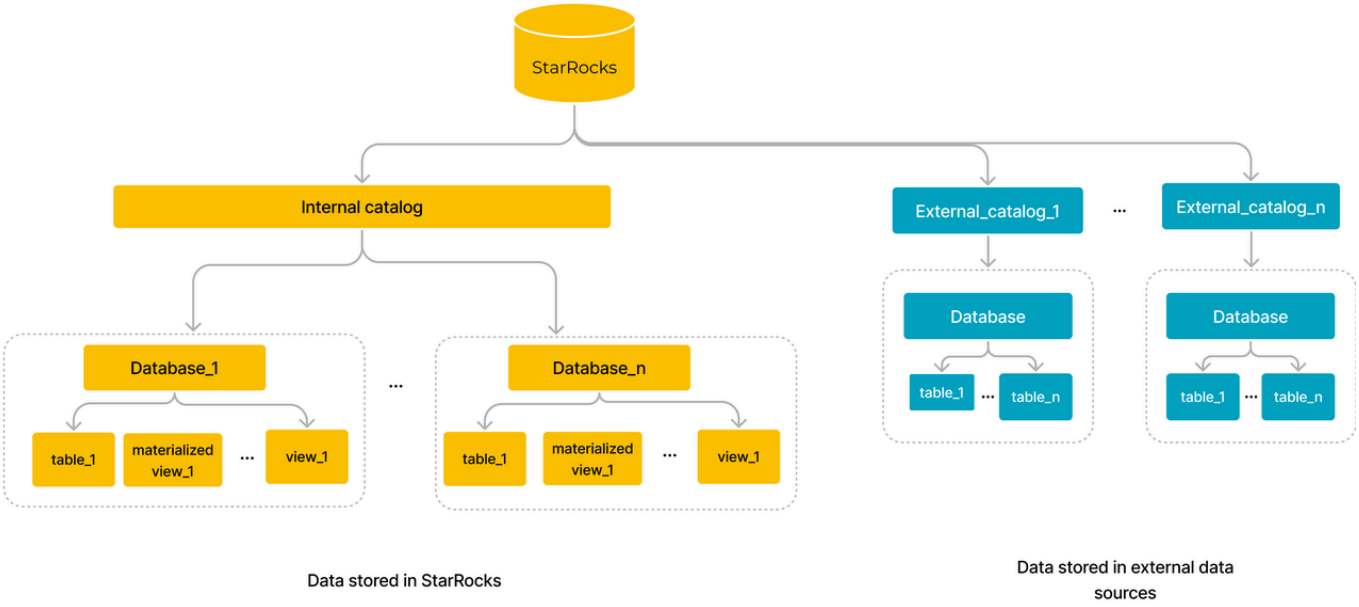
- Чрезмерная разница между значениями Max и Min метрики `PushChunkNum` у `OlapTableSink` указывает на перекос данных в upstream операторах, что может привести к узкому месту в производительности загрузки.
- `RpcClientSideTime` равен `RpcServerSideTime` плюс время сетевой передачи плюс время обработки фреймворком RPC. Если есть значительная разница между `RpcClientSideTime` и `RpcServerSideTime`, рассмотрите возможность включения сжатия для сокращения времени передачи.

Метрика	Описание
`IndexNum`	Количество синхронных materialized views, созданных для целевой таблицы.
`ReplicatedStorage`	Включена ли Single Leader Replication.
`TxnID`	ID транзакции загрузки.
`RowsRead`	Количество строк, прочитанных из upstream операторов.
`RowsFiltered`	Количество строк, отфильтрованных из-за недостаточного качества данных.
`RowsReturned`	Количество строк, записанных в целевую таблицу.
`RpcClientSideTime`	Общее время потребления RPC для загрузки, записанное на стороне клиента.
`RpcServerSideTime`	Общее время потребления RPC для загрузки, записанное на стороне сервера.
`PrepareDataTime`	Общее время потребления для фазы подготовки данных, включая преобразование формата данных и проверку качества данных.
`SendDataTime`	Локальное время потребления для отправки данных, включая время для сериализации и сжатия данных, а также для отправки задач в очередь отправителя.

2 Проектирование таблиц

2.1 Каталоги, базы данных и таблицы

Selena использует внутренние каталоги для управления внутренними данными и внешние каталоги для доступа к внешним данным в озере данных. Данные, хранящиеся в Selena, поддерживаются внутри внутреннего каталога, который состоит из одной или нескольких баз данных. База данных управляет различными объектами, включая таблицы, материализованные представления и представления для хранения, управления и манипулирования данными. Selena использует систему привилегий, которая контролирует права пользователей на выполнение операций над объектами, тем самым повышая безопасность данных.



2.1.1 Каталоги

Каталоги подразделяются на внутренние и внешние каталоги. Каждый cluster имеет только один внутренний каталог `default_catalog`, который может включать одну или несколько баз данных. Внутренний каталог cluster'a Selena управляет данными, загруженными в cluster, и материализованными представлениями. Selena может хорошо служить в качестве хранилища данных для хранения данных, что значительно повышает производительность запросов, особенно для сложной аналитики запросов на крупномасштабных данных.

Внешний каталог, с другой стороны, позволяет получить доступ к внешним данным в вашем озере данных. Вы можете использовать Selena в качестве движка запросов для прямого запроса данных в озере без загрузки данных в Selena.

2.1.2 Базы данных

База данных — это коллекция объектов данных, таких как таблицы, представления и материализованные представления, для хранения, управления и манипулирования данными.

2.1.3 Таблицы

Таблицы подразделяются на внутренние таблицы и внешние таблицы.

Внутренние таблицы

Внутренние таблицы поддерживаются во внутренних каталогах. Данные внутренних таблиц хранятся в Selena. Внутренние таблицы поддерживаются в базах данных под внутренним каталогом. Внутренняя таблица состоит из строк и столбцов, где каждая строка представляет запись, а каждый столбец представляет атрибут или поле строки данных.

> **ПРИМЕЧАНИЕ**
>
> Строки и столбцы внутренних таблиц здесь являются логическими концепциями. Selena использует колоночное хранилище для таблиц. Физически данные в столбце разделяются на блоки данных, кодируются, сжимаются, а затем постоянно хранятся на диске.

Selena предоставляет четыре типа внутренних таблиц: Primary Key таблицы, Duplicate Key таблицы, Aggregate таблицы и Unique Key таблицы для хранения различных данных, таких как необработанные логи, данные реального времени и агрегированные данные, чтобы удовлетворить ваши различные бизнес-потребности.

Внутренняя таблица применяет двухуровневую стратегию распределения данных, то есть partitioning + bucketing, для достижения равномерного распределения данных. Более того, replica каждого bucket равномерно распределены по BE для гарантии высокой доступности данных.

Внешние таблицы

Внешние таблицы поддерживаются во внешних каталогах. Данные внешних таблиц фактически хранятся во внешних источниках данных. Selena хранит только метаданные этих внешних таблиц, и вы можете использовать эти таблицы для запроса данных из внешних источников данных.

2.1.4 Материализованные представления

Материализованные представления — это специальные физические таблицы, которые содержат предварительно вычисленные результаты запросов из одной или нескольких базовых таблиц. Материализованные представления подразделяются на синхронные и асинхронные материализованные представления. Асинхронные материализованные представления более мощные, поскольку они могут хранить предварительно вычисленные результаты из нескольких базовых таблиц (внутренних и внешних таблиц) и поддерживают различные операторы агрегации. Когда сложные запросы выполняются на базовых таблицах, Selena может автоматически повторно использовать предварительно вычисленные результаты, хранящиеся в материализованных представлениях, для достижения прозрачного ускорения запросов, ускорения запросов в озерах данных и создания моделирования данных.

2.1.5 Представления

Представления, или логические представления, являются виртуальными таблицами, которые не содержат никаких данных. Представления отображают только данные, которые являются результатами запросов к другим существующим физическим таблицам. Запрос, который определяет представление, выполняется каждый раз, когда на представление ссылаются в запросе.

2.1.6 Привилегии

Привилегии определяют, какие пользователи могут выполнять какие операции над какими объектами. Selena применяет два типа моделей привилегий: контроль доступа на основе идентификации и контроль доступа на основе ролей. Вы можете сначала назначить привилегии ролям, а затем назначить роли пользователям. В этом случае привилегии передаются пользователям через роли. Или вы можете напрямую назначить привилегии идентификаторам пользователей.

2.1.7 Хранение данных в архитектуре разделения хранилища и вычислений

Начиная с v1.5.2, Selena вводит новую архитектуру разделения хранилища и вычислений (shared-data). Хранение данных отделено от BE. Данные постоянно хранятся в удаленном объектном хранилище или HDFS, в то время как локальные диски используются для кэширования горячих данных для ускорения запросов.

2.2 Обзор таблиц

Таблицы являются единицами хранения данных. Понимание структуры таблиц в Selena и того, как проектировать эффективную структуру таблиц, помогает оптимизировать организацию данных и повысить эффективность запросов. Кроме того, по сравнению с традиционными базами данных, Selena может хранить сложные полуструктурированные данные, такие как JSON, ARRAY, в колоночном формате для повышения производительности запросов.

Эта тема знакомит со структурой таблиц в Selena как с базовой, так и с общей точки зрения.

Начиная с версии v1.5.2, Selena поддерживает создание временных таблиц в Default Catalog.

2.2.1 Знакомство с базовой структурой таблиц

Как и в других реляционных базах данных, таблица логически состоит из строк и столбцов:

- Строки: Каждая строка содержит запись. Каждая строка содержит набор связанных значений данных.
- Столбцы: Столбцы определяют атрибуты для каждой записи. Каждый столбец содержит данные определенного атрибута. Например, таблица сотрудников может включать столбцы, такие как имя, ID сотрудника, отдел и зарплата, где каждый столбец хранит соответствующие данные. Данные в каждом столбце имеют одинаковый тип данных. Все строки в таблице имеют одинаковое количество столбцов.

Создать таблицу в Selena просто. Вам просто нужно определить столбцы и их типы данных в операторе CREATE TABLE, чтобы создать таблицу. Пример:

```
CREATE DATABASE example_db;
USE example_db;
CREATE TABLE user_access (
    uid int,
    name varchar(64),
    age int,
    phone varchar(16),
    last_access datetime,
    credits double
)
ORDER BY (uid, name);
```

Приведенный выше пример CREATE TABLE создает таблицу Duplicate Key. К столбцам в этом типе таблиц не добавляются ограничения, поэтому в таблице могут существовать дублирующиеся строки данных. Первые два столбца таблицы Duplicate Key указываются как столбцы сортировки для формирования ключа сортировки. Данные хранятся после сортировки на основе ключа сортировки, что может ускорить индексирование во время запросов.

Начиная с версии v1.5.2, таблица Duplicate Key поддерживает указание ключа сортировки с помощью `ORDER BY`. Если используются и `ORDER BY`, и `DUPLICATE KEY`, `DUPLICATE KEY` не действует.

> **ПРИМЕЧАНИЕ**
>
> Если cluster Selena в тестовой среде содержит только один BE, количество replica можно установить равным `1` в предложении `PROPERTIES`, например `PROPERTIES( "replication_num" = "1" )`. Количество replica по умолчанию равно 3, что также является рекомендуемым значением для production-кластеров Selena. Если вы хотите использовать значение по умолчанию, настраивать параметр `replication_num` не нужно.

Выполните DESCRIBE для просмотра схемы таблицы.

```
MySQL [test]> DESCRIBE user_access;
+-------------+-------------+------+-------+---------+-------+
| Field       | Type        | Null | Key   | Default | Extra |
+-------------+-------------+------+-------+---------+-------+
| uid         | int         | YES  | true  | NULL    |       |
| name        | varchar(64) | YES  | true  | NULL    |       |
| age         | int         | YES  | false | NULL    |       |
| phone       | varchar(16) | YES  | false | NULL    |       |
| last_access | datetime    | YES  | false | NULL    |       |
| credits     | double      | YES  | false | NULL    |       |
+-------------+-------------+------+-------+---------+-------+
6 rows in set (0.00 sec)
```

Выполните SHOW CREATE TABLE для просмотра оператора CREATE TABLE.

```
MySQL [example_db]> SHOW CREATE TABLE user_access\G
*************************** 1. row ***************************
       Table: user_access
Create Table: CREATE TABLE `user_access` (
  `uid` int(11) NULL COMMENT "",
  `name` varchar(64) NULL COMMENT "",
  `age` int(11) NULL COMMENT "",
  `phone` varchar(16) NULL COMMENT "",
  `last_access` datetime NULL COMMENT "",
  `credits` double NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`uid`, `name`)
DISTRIBUTED BY RANDOM
ORDER BY(`uid`, `name`)
PROPERTIES (
"bucket_size" = "4294967296",
"compression" = "LZ4",
"fast_schema_evolution" = "true",
"replicated_storage" = "true",
"replication_num" = "3"
);
1 row in set (0.01 sec)
```

2.2.2 Понимание комплексной структуры таблиц

Глубокое погружение в структуры таблиц Selena помогает проектировать эффективную структуру управления данными, адаптированную к вашим бизнес-потребностям.

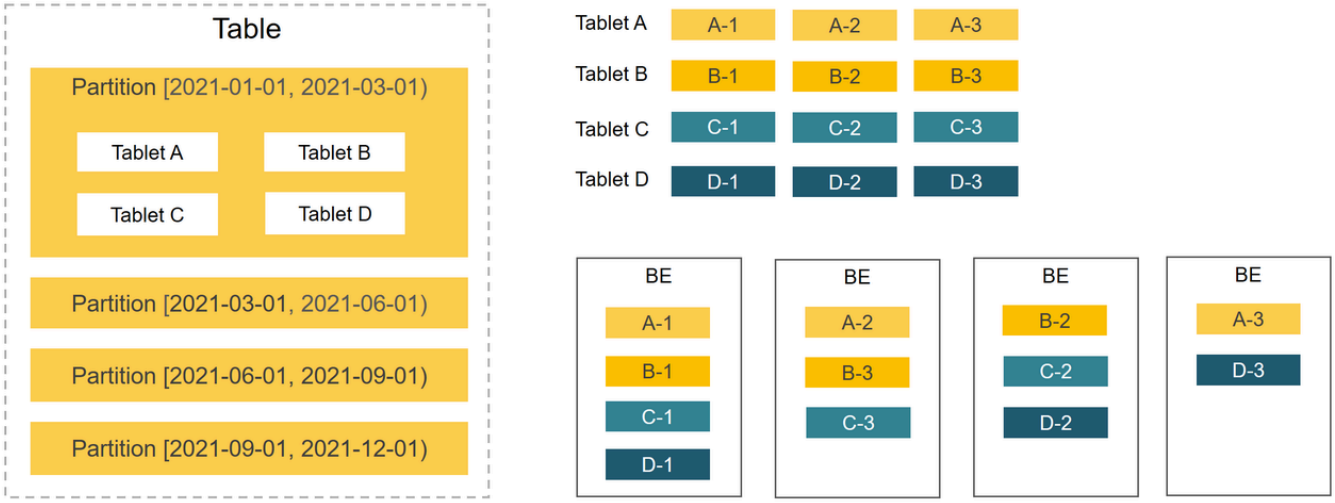
Типы таблиц

Selena предоставляет четыре типа таблиц: таблицы Duplicate Key, таблицы Primary Key, таблицы Aggregate и таблицы Unique Key для хранения данных в различных бизнес-сценариях, таких как необработанные данные, часто обновляемые данные в реальном времени и агрегированные данные.

- Таблицы Duplicate Key просты и легки в использовании. К столбцам в этом типе таблиц не добавляются ограничения, поэтому в таблице могут существовать дублирующиеся строки данных. Таблицы Duplicate Key подходят для хранения необработанных данных, таких как логи, которые не нуждаются в каких-либо ограничениях или предварительной агрегации.
- Таблицы Primary Key являются мощными. К столбцам первичного ключа добавляются ограничения уникальности и NOT NULL. Таблицы Primary Key поддерживают частые обновления в реальном времени и частичные обновления столбцов, обеспечивая при этом высокую производительность запросов, и поэтому подходят для сценариев запросов в реальном времени.
- Таблицы Aggregate подходят для хранения предварительно агрегированных данных, помогая уменьшить объем сканируемых и вычисляемых данных и повысить эффективность для агрегационных запросов.
- Таблицы Unique также подходят для хранения часто обновляемых данных в реальном времени. Однако этот тип таблиц заменяется таблицами Primary Key, которые более мощные.

Распределение данных

Selena использует двухуровневую стратегию распределения данных с разбиением на партиции+bucketing, чтобы равномерно распределять данные по BE. Хорошо спроектированная стратегия распределения данных может эффективно уменьшить объем сканируемых данных и максимизировать возможности параллельной обработки Selena, тем самым повышая производительность запросов.



Партиционирование

Первый уровень - это партиционирование: Данные в таблицах могут быть разделены на более мелкие единицы управления данными на основе столбцов партиционирования, которые обычно являются столбцами, содержащими даты и время. Во время запросов отсечение партиций может уменьшить объем данных, которые необходимо сканировать, эффективно оптимизируя производительность запросов.

Selena предоставляет простой в использовании метод партиционирования, партиционирование по выражению, а также предлагает более гибкие методы, такие как партиционирование по диапазону и списку.

Bucketing

Второй уровень - это bucketing: Данные внутри партиции далее делятся на более мелкие единицы управления данными через bucketing. Реплики каждого bucket равномерно распределены по BE для обеспечения высокой доступности данных.

Selena предоставляет два метода bucketing:

- Hash bucketing: Данные распределяются в bucket на основе хеш-значений ключа bucketing. Вы можете выбрать столбцы, часто используемые в качестве условных столбцов в запросах, в качестве столбцов bucketing, что помогает повысить эффективность запросов.
- Random bucketing: Данные случайным образом распределяются в bucket. Этот метод bucketing более простой и легкий в использовании.

Типы данных

Помимо базовых типов данных, таких как NUMERIC, DATE и STRING, Selena поддерживает сложные полуструктурированные типы данных, включая ARRAY, JSON, MAP и STRUCT.

Индексы

Индекс - это специальная структура данных, используемая в качестве указателя на данные в таблице. Когда условные столбцы в запросах являются индексированными столбцами, Selena может быстро найти данные, которые соответствуют условиям.

Selena предоставляет встроенные индексы: индексы Prefix, индексы Ordinal и индексы ZoneMap. Selena также позволяет пользователям создавать индексы, то есть индексы Bitmap и индексы Bloom Filter, для дальнейшего повышения эффективности запросов.

Ограничения

Ограничения помогают обеспечить целостность, согласованность и точность данных. Столбцы первичного ключа в таблицах Primary Key должны иметь уникальные значения и значения NOT NULL. Столбцы ключа агрегации в таблицах Aggregate и столбцы уникального ключа в таблицах Unique Key должны иметь уникальные значения.

Временные таблицы

При обработке данных вам может потребоваться сохранить промежуточные результаты для будущего повторного использования. В ранних версиях Selena поддерживает только использование CTE (Common Table Expressions) для определения временных результатов в рамках одного запроса. Однако CTE являются просто логическими конструкциями, не хранят результаты физически и не могут использоваться в разных запросах, что представляет определенные ограничения. Если вы решите создать таблицы для сохранения промежуточных результатов, вам нужно будет управлять жизненным циклом этих таблиц, что может быть затратным.

Чтобы решить эту проблему, Selena вводит временные таблицы в версии v1.5.2. Временные таблицы позволяют временно хранить данные (например, промежуточные результаты из ETL-процессов) в таблице, причем их жизненный цикл привязан к сессии и управляется Selena. Когда сессия завершается, временные таблицы автоматически очищаются. Временные таблицы видны только в текущей сессии, и разные сессии могут создавать временные таблицы с одинаковыми именами.

Использование

Вы можете использовать ключевое слово `TEMPORARY` в следующих SQL-операторах для создания и удаления временных таблиц:

- CREATE TABLE
- CREATE TABLE AS SELECT
- CREATE TABLE LIKE
- DROP TABLE

> **ПРИМЕЧАНИЕ**
>
> Подобно другим типам нативных таблиц, временные таблицы должны быть созданы в базе данных под Default Catalog. Однако, поскольку временные таблицы основаны на сессиях, они не подчиняются ограничениям уникальности имен. Вы можете создавать временные таблицы с одинаковыми именами в разных сессиях или даже создавать временные таблицы с теми же именами, что и другие не-временные, нативные таблицы.
>
> Если в базе данных есть временные и не-временные таблицы с одинаковым именем, временная таблица имеет приоритет. В рамках сессии все запросы и операции над таблицами с одинаковым именем будут влиять только на временную таблицу.

Ограничения

Хотя использование временных таблиц аналогично использованию нативных таблиц, существуют некоторые ограничения и различия:

- Временные таблицы должны быть созданы в Default Catalog.
- Установка colocate group не поддерживается. Если свойство `colocate_with` явно указано при создании таблицы, оно будет проигнорировано.
- `ENGINE` должен быть указан как `olap` при создании таблицы.
- Операторы ALTER TABLE не поддерживаются.
- Создание представлений и материализованных представлений на основе временных таблиц не поддерживается.
- Операторы EXPORT не поддерживаются.
- Создание временных таблиц или загрузка данных из/в них с помощью асинхронных задач с использованием SUBMIT TASK не поддерживается.

Дополнительные функции

Помимо перечисленных выше функций, вы можете использовать дополнительные функции на основе ваших бизнес-требований для проектирования более надежной структуры таблиц. Например, использование столбцов Bitmap и HLL для ускорения подсчета уникальных значений, указание генерируемых столбцов или столбцов с автоинкрементом для ускорения некоторых запросов, настройка гибких и автоматических методов охлаждения хранилища для снижения затрат на обслуживание и настройка Colocate Join для ускорения многотабличных JOIN-запросов. Для получения дополнительной информации см. CREATE TABLE.

2.3 Обзор типов таблиц

Вы должны указать тип таблицы и определить один или несколько столбцов в качестве ключа сортировки при создании таблицы. Таким образом, когда данные первоначально загружаются в созданную вами таблицу, Selena может сортировать, обрабатывать и хранить данные на основе ключа сортировки. В этом разделе описаны типы таблиц, которые предоставляет Selena для удовлетворения различных бизнес-требований.

2.3.1 Основные концепции

Типы таблиц

Selena предоставляет четыре типа таблиц: Duplicate Key table, Aggregate table, Unique Key table и Primary Key table. Эти четыре типа таблиц хорошо подходят для широкого спектра сценариев аналитики данных, таких как анализ логов, агрегация и анализ данных, и аналитика данных в реальном времени.

Ключи сортировки

Когда данные загружаются в таблицу, созданную с использованием определённого типа таблицы, данные сортируются и хранятся в соответствии с одним или несколькими столбцами, определёнными как ключ сортировки при создании таблицы. Ключ сортировки обычно представляет собой один или несколько столбцов, которые часто используются в качестве условий фильтрации в запросах, тем самым ускоряя запросы.

Начиная с версии 3.0, Primary Key tables поддерживают определение ключей сортировки с помощью `ORDER BY`. Начиная с версии 3.3, Duplicate Key tables, Aggregate tables и Unique Key tables поддерживают определение ключей сортировки с помощью `ORDER BY`.

Для более подробного описания ключей сортировки см. Ключи сортировки и префиксные индексы.

2.3.2 Меры предосторожности

- После создания таблицы вы не можете изменить её тип. Например, вы не можете изменить Duplicate Key table на Primary Key table. Вы должны создать новую Primary Key table.

- Столбцы ключа сортировки должны быть определены до других столбцов в операторе создания таблицы.

- Порядок столбцов ключа сортировки в операторе создания таблицы определяет порядок условий, на основе которых сортируются строки в таблице.

- Длина префиксного индекса для таблицы ограничена 36 байтами. Если общая длина столбцов ключа сортировки превышает 36 байт, Selena сохраняет только первые несколько столбцов ключа сортировки в пределах ограничения длины для префиксного индекса.

- Если записи, которые должны быть загружены в таблицу, имеют одинаковый первичный ключ, Selena обрабатывает и хранит записи в зависимости от типа таблицы:

  - Duplicate Key table

    Selena загружает каждую из записей как отдельную строку в таблицу. После завершения загрузки данных таблица содержит строки с одинаковым первичным ключом, и строки отображают исходные записи в соотношении один к одному. Вы можете получить все исторические данные, которые вы загрузили.

  - Aggregate table

    Selena агрегирует записи в одну запись и загружает агрегированную запись как строку в таблицу. После завершения загрузки таблица не содержит строк с одинаковым первичным ключом. Вы можете получить результаты агрегации всех исторических данных, которые вы загрузили. Однако вы не можете получить все исторические данные.

- Unique Key table и Primary Key table

  Selena заменяет каждую ранее загруженную запись вновь загруженной записью и сохраняет только последнюю загруженную запись как строку в таблице. После завершения загрузки таблица не содержит предыдущих строк с одинаковым первичным ключом. Сохраняется только последняя запись, содержащая тот же unique key/primary key. Unique Key table и Primary Key table можно рассматривать как специальную Aggregate table, в которой для столбцов метрик указана агрегатная функция REPLACE для возврата самой последней записи среди группы записей с одинаковым первичным ключом.

2.3.3 Возможности различных типов таблиц

Ключевые столбцы и ключ сортировки

	Primary Key table	Duplicate Key table	Aggregate table	Unique Key table
Ключевые столбцы и ограничения	Первичный ключ имеет ограничение UNIQUE и ограничение NOT NULL.	Duplicate key не имеет ограничения UNIQUE.	Aggregate key имеет ограничение UNIQUE.	Unique key имеет ограничение UNIQUE.

	Primary Key table	Duplicate Key table	Aggregate table	Unique Key table
UNIQUE				
Логические связи между ключевыми столбцами	Если значение первичного ключа новой строки данных совпадает со значением существующей строки данных в таблице, возникает нарушение ограничения уникальности. Тогда новая строка данных заменит существующую строку данных. По сравнению с Unique Key table, Primary Key table имеет улучшенный механизм хранения и поэтому может заменить Unique Key table.	Duplicate key не имеет ограничения UNIQUE. Поэтому, если значение duplicate key новой строки данных совпадает со значением существующей строки данных в таблице, обе новая и старая строки данных сохраняются в таблице.	Если значение aggregate key новой строки данных совпадает со значением существующей строки данных в таблице, новая и старая строки данных будут агрегированы на основе aggregate key и агрегатных функций столбцов значений.	Если значение unique key новой строки данных совпадает со значением существующей строки данных в таблице, новая строка данных заменит существующую строку данных. Unique Key table можно рассматривать как Aggregate table, агрегатная функция которой - replace.

	Primary Key table	Duplicate Key table	Aggregate table	Unique Key table
изменениями данных				
Связь между ключевыми столбца	Начиная с версии 3.0.0, ключ сортировки отделён от первичного ключа в Primary Key table.	Начиная с версии 3.3.0, Duplicate Key tables поддерживают указание ключей сортировки с помощью `ORDER BY`. Если используются и `ORDER BY`, и `DUPLICATE KEY`, то `ORDER BY` имеет приоритет.	Начиная с версии 3.3.0, ключ сортировки отделён от aggregate key в Aggregate table. Aggregate table поддерживает указание ключа сортировки с помощью `ORDER BY` и указание aggregate key с помощью `AGGREGATE KEY`. Столбцы в ключе сортировки и aggregate key должны быть одинаковыми, но порядок столбцов не обязательно должен быть одинаковым.	Начиная с версии 3.3.0, ключ сортировки отделён от unique key в Unique Key table. Unique Key table поддерживает указание ключа сортировки с помощью `ORDER BY` и указание unique key с помощью `UNIQUE KEY`. Столбцы в ключе сортировки и unique key должны быть одинаковыми, но порядок столбцов не обязательно должен быть одинаковым.

	Primary Key table	Duplicate Key table	Aggregate table	Unique Key table
мини ключом сортировки				
Типы данных, поддерживаемые к	Числовые (включая целые числа и BOOLEAN), строковые и дата (DATE и DATETIME).	Числовые (включая целые числа, BOOLEAN и DECIMAL), строковые и дата (DATE и DATETIME).		

	Primary Key table	Duplicate Key table	Aggregate table	Unique Key table
ключевыми столбцами и ключом сортировки				
Связь между	Столбцы партиционирования и bucketing должны быть в первичном ключе.	Отсутствует	Столбцы партиционирования и bucketing должны быть в aggregate key.	Столбцы партиционирования и bucketing должны быть в unique key.

	Primary Key table	Duplicate Key table	Aggregate table	Unique Key table
ключевыми столбцами и столбцами партиционирования				

	Primary Key table	Duplicate Key table	Aggregate table	Unique Key table
я/bucketing				

Типы данных ключевых столбцов и столбцов значений

Ключевые столбцы поддерживают следующие типы данных: числовые (включая целые числа, BOOLEAN и DECIMAL), строковые и дата (DATE и DATETIME).

> **ПРИМЕЧАНИЕ**
>
> Ключевые столбцы Primary Key table не поддерживают тип данных DECIMAL.

С другой стороны, столбцы значений поддерживают базовые типы данных, включая числовые, строковые и дата (DATE и DATETIME). Поддержка BITMAP, HLL и полуструктурированных типов варьируется для столбцов значений различных типов таблиц, как подробно описано ниже:

	Primary Key table	Duplicate Key table	Aggregate table	Unique Key table
BITMAP	Поддерживается	Не поддерживается	Поддерживается. Агрегатные функции должны быть bitmap_union, replace или replace_if_not_null.	Поддерживается
HLL	Поддерживается	Не поддерживается	Поддерживается. Агрегатные функции должны быть hll_union, replace или replace_if_not_null.	Поддерживается
PERCENTILE	Поддерживается	Не поддерживается	Поддерживается. Агрегатные функции должны быть percentile_union, replace или replace_if_not_null.	Поддерживается
полуструктурированные типы данных : JSON/ARRAY/MAP/STRUCT	Поддерживается	Поддерживается	Поддерживается. Агрегатные функции должны быть replace или replace_if_not_null.	Поддерживается

Изменение данных

<table>
<tr><th></th><th>Primary Key Table</th><th>Duplicate Key table</th><th>Aggregate table</th><th>Unique Key table</th></tr>
<tr><td>INSERT через загрузку данных</td><td rowspan="2">Поддерживается. Настройте <code>__op=0</code> в задании загрузки для выполнения INSERT. Во внутренней реализации Selena рассматривает операции INSERT и UPDATE как операции UPSERT.</td><td>Поддерживается</td><td>Поддерживается (Строки данных с одинаковым значением aggregate key будут агрегированы.)</td><td>Поддерживается (Строки данных с одинаковым значением unique key будут обновлены.)</td></tr>
<tr><td>UPDATE через загрузку данных</td><td>Не поддерживается</td><td>Поддерживается (Это может быть достигнуто с помощью replace в качестве агрегатной функции.)</td><td>Поддерживается (Unique Key table сама по себе может рассматриваться как Aggregate table, использующая агрегатную функцию replace.)</td></tr>
<tr><td>DELETE через загрузку данных</td><td>Поддерживается. Настройте <code>__op=1</code> в задании загрузки для выполнения DELETE.</td><td colspan="3">Не поддерживается</td></tr>
<tr><td>Целостность значений столбцов данных для загрузки</td><td>По умолчанию необходимо загрузить все значения столбцов. Однако, если включено частичное обновление столбцов (<code>partial_update</code>), или столбец имеет значение по умолчанию, не все значения столбцов необходимо загружать.</td><td>По умолчанию необходимо загрузить все значения столбцов. Однако, если столбец имеет значение по умолчанию, не все значения столбцов необходимо загружать.</td><td>По умолчанию необходимо загрузить все значения столбцов. Однако, Aggregate tables могут достичь частичного обновления столбцов, указав агрегатные функции для столбцов значений как REPLACE_IF_NOT_NULL. Для получения дополнительной информации см. aggr_type. Также, если столбец имеет значение по умолчанию, не все</td><td>По умолчанию необходимо загрузить все значения столбцов. Однако, если столбец имеет значение по умолчанию, не все значения столбцов необходимо загружать.</td></tr>
</table>

	Primary Key Table	Duplicate Key table	Aggregate table	Unique Key table
			значения столбцов необходимо загружать.	
DML INSERT	Поддерживается			
DML UPDATE	• Ключевые столбцы в качестве условий фильтрации: Поддерживается • Столбцы значений в качестве условий фильтрации: Поддерживается	Не поддерживается		
DML DELETE	• Ключевые столбцы в качестве условий фильтрации: Поддерживается • Столбцы значений в качестве условий фильтрации: Поддерживается	• Ключевые столбцы в качестве условий фильтрации: Поддерживается • Столбцы значений в качестве условий фильтрации: Поддерживается Обратите внимание, что поддерживаются только простые условия фильтрации на основе самих ключевых или столбцов значений, такие как =, <, >. Сложные условия фильтрации, такие как функции или подзапросы, не поддерживаются.	• Ключевые столбцы в качестве условий фильтрации: Поддерживается. Обратите внимание, что поддерживаются только простые условия фильтрации на основе самих ключевых столбцов, такие как =, <, >. Сложные условия фильтрации, такие как функции или подзапросы, не поддерживаются. • Столбцы значений в качестве условий фильтрации: Не поддерживается.	

Совместимость с другими функциями

		Primary Key table	Duplicate Key table	Aggregate table	Unique Key table
Bitmap index/bloom filter index	**Создание индексов на ключевых столбцах**	Поддерживается			
	Создание индексов на столбцах значений	Поддерживается	Поддерживается	Не поддерживается	Не поддерживается
Партиционирование/bucketing	**Партиционирование по выражению/ партиционирование списком**	Поддерживается			
	Случайный bucketing	Не поддерживается	Поддерживается начиная с версии 3.1	Не поддерживается	Не поддерживается
Материализованные представления	**Асинхронные материализованные представления**	Поддерживается			
	Синхронные материализованные представления	Не поддерживается	Поддерживается	Поддерживается	Поддерживается
Другие функции	**CTAS**	Поддерживается	Поддерживается	Не поддерживается	Не поддерживается
	Backup & restore	Поддерживается начиная с версии 2.5	Поддерживается		

2.3.4 Primary Key table

Primary Key table использует новый механизм хранения, разработанный Selena. Его основное преимущество заключается в поддержке обновления данных в реальном времени при обеспечении эффективной производительности для сложных ad-hoc запросов. В аналитике бизнеса в реальном времени принятие решений может выиграть от Primary Key tables, которые используют самые свежие данные для анализа результатов в реальном времени, что может смягчить задержку данных в анализе данных.

Первичный ключ Primary Key table имеет ограничение UNIQUE и ограничение NOT NULL, и используется для уникальной идентификации каждой строки данных. Если значение первичного ключа новой строки данных совпадает со значением существующей строки

данных в таблице, возникает нарушение ограничения UNIQUE. Тогда новая строка данных заменит существующую строку данных.

К СВЕДЕНИЮ

- Начиная с версии 3.0, ключ сортировки Primary Key table отделён от первичного ключа таблицы, и ключ сортировки может быть указан отдельно. Таким образом, повышается гибкость создания таблиц.
- Начиная с версии 3.1, Selena shared-data clusters поддерживают создание Primary Key tables.
  - Начиная с версии 3.1.4, персистентные индексы могут быть созданы и сохранены на **локальных дисках**.
  - Начиная с версии 3.3.2, персистентные индексы могут быть созданы и сохранены в **объектном хранилище**.

Сценарии

Primary Key table может поддерживать обновление данных в реальном времени, обеспечивая при этом эффективную производительность запросов. Она подходит для следующих сценариев:

- **Потоковая передача данных в реальном времени из систем обработки транзакций в Selena.** В обычных случаях системы обработки транзакций включают большое количество операций обновления и удаления в дополнение к операциям вставки. Если вам необходимо синхронизировать данные из системы обработки транзакций в Selena, мы рекомендуем создать Primary Key table. Затем вы можете использовать инструменты, такие как CDC Connectors for Apache Flink®, для синхронизации бинарных логов системы обработки транзакций в Selena. Selena использует бинарные логи для добавления, удаления и обновления данных в таблице в реальном времени. Это упрощает синхронизацию данных и обеспечивает производительность запросов в 3-10 раз выше, чем при использовании Unique Key table, которая использует стратегию Merge-On-Read. Для получения дополнительной информации см. Синхронизация в реальном времени из MySQL.
- **Объединение нескольких потоков путём выполнения частичных обновлений отдельных столбцов**. В бизнес-сценариях, таких как профилирование пользователей, предпочтительно использовать плоские таблицы для повышения производительности многомерного анализа и упрощения аналитической модели,

используемой аналитиками данных. Данные upstream в этих сценариях могут поступать из различных приложений, таких как приложения для шоппинга, приложения для доставки и банковские приложения, или из систем, таких как системы машинного обучения, которые выполняют вычисления для получения различных тегов и свойств пользователей. Primary Key table хорошо подходит для этих сценариев, потому что она поддерживает обновления отдельных столбцов. Каждое приложение или система может обновлять только столбцы, которые содержат данные в пределах их собственной области обслуживания, при этом пользуясь добавлением, удалением и обновлением данных в реальном времени с высокой производительностью запросов.

Как это работает

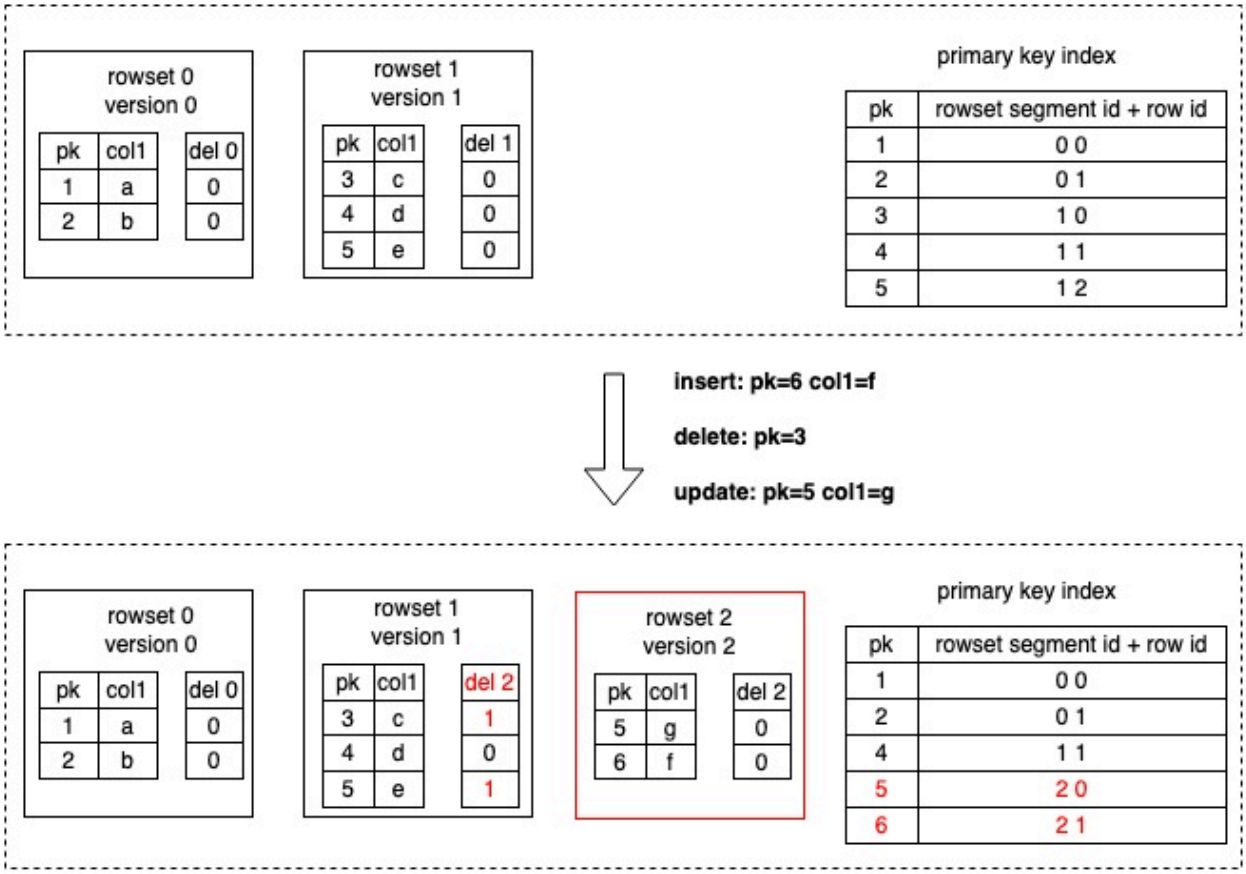
Unique Key table и Aggregate table используют стратегию Merge-On-Read. Эта стратегия делает запись данных простой и эффективной, но требует объединения нескольких версий файлов данных онлайн во время чтения данных. Более того, поскольку существует оператор Merge, предикаты и индексы не могут быть проталкнуты в базовые данные, что серьёзно влияет на производительность запросов.

Однако для баланса производительности обновлений в реальном времени и запросов структура метаданных и механизм чтения/записи в Primary Key table отличаются от таковых в других типах таблиц. Primary Key table использует стратегию Delete+Insert. Эта стратегия реализуется с использованием индекса первичного ключа и DelVector. Эта стратегия гарантирует, что во время запросов нужно читать только последнюю запись среди записей с одинаковым значением первичного ключа, что устраняет необходимость объединения нескольких версий файлов данных. Более того, предикаты и индексы могут быть проталкнуты в базовые данные, что значительно улучшает производительность запросов.

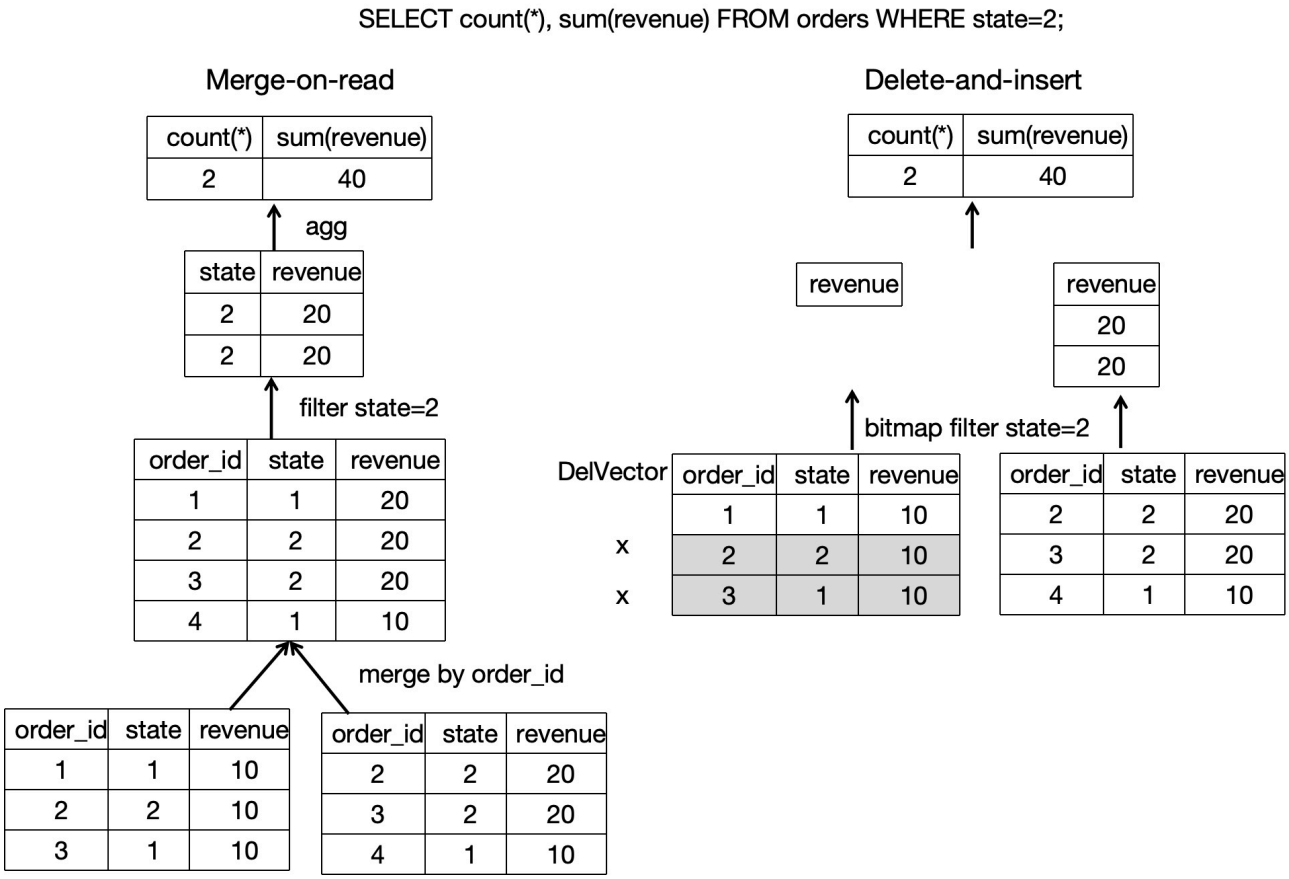
Общий процесс записи и чтения данных в Primary Key table выглядит следующим образом:

- Запись данных достигается через внутреннее задание загрузки Selena (Loadjob), которое включает пакет операций изменения данных (Insert, Update и Delete). Selena загружает индексы первичного ключа соответствующих tablets в память. Для операций Delete Selena сначала использует индекс первичного ключа для поиска исходного местоположения (файл данных и номер строки) каждой строки данных, отмечая строку данных как удалённую в DelVector (который хранит и управляет

маркерами удаления, сгенерированными во время загрузки данных). Для операций Update, помимо отметки исходной строки данных как удалённой в DelVector, Selena также записывает последнюю строку данных в новый файл данных, по сути преобразуя Update в Delete+Insert (как показано на следующем рисунке). Индекс первичного ключа также обновляется для записи нового местоположения (файл данных и номер строки) изменённой строки данных.



- Во время чтения данных, поскольку исторические дублирующие записи в различных файлах данных уже были отмечены как удалённые во время записи данных, необходимо читать только последнюю строку данных с одинаковым значением первичного ключа. Больше не нужно читать онлайн несколько версий файлов данных для дедупликации данных и поиска последних данных. Когда сканируются базовые файлы данных, операторы фильтрации и различные индексы помогают уменьшить затраты на сканирование (как показано на следующем рисунке). Таким образом, производительность запросов может быть значительно улучшена. По сравнению со стратегией Merge-On-Read Unique Key table, стратегия Delete+Insert Primary Key table может улучшить производительность запросов в 3-10 раз.



Подробнее

Использование

Создание Primary Key table

Вам просто нужно определить первичный ключ в операторе `CREATE TABLE`, чтобы создать Primary Key table. Пример:

```
CREATE TABLE orders1 (
    order_id bigint NOT NULL,
    dt date NOT NULL,
    user_id INT NOT NULL,
    good_id INT NOT NULL,
    cnt int NOT NULL,
    revenue int NOT NULL
)
PRIMARY KEY (order_id)
DISTRIBUTED BY HASH (order_id)
;
```

> **К СВЕДЕНИЮ**
>
> Поскольку Primary Key table поддерживает только hash bucketing в качестве стратегии bucketing, вам также необходимо определить ключ hash bucketing с помощью `DISTRIBUTED BY HASH ()`.

Однако в реальных бизнес-сценариях при создании Primary Key table часто используются дополнительные функции, такие как распределение данных и ключ сортировки, для ускорения запросов и более эффективного управления данными.

Например, поле `order_id` в таблице заказов может однозначно идентифицировать строки данных, поэтому поле `order_id` может использоваться в качестве первичного ключа.

Начиная с версии 3.0, ключ сортировки Primary Key table отделён от первичного ключа таблицы. Поэтому вы можете выбрать столбцы, часто используемые в качестве условий фильтрации запросов, для формирования ключа сортировки. Например, если вы часто запрашиваете эффективность продаж товаров на основе комбинации двух измерений, дата заказа и продавец, вы можете указать ключ сортировки как `dt` и `merchant_id` с помощью предложения `ORDER BY (dt,merchant_id)`.

Обратите внимание, что если вы используете стратегии распределения данных, Primary Key table в настоящее время требует, чтобы первичный ключ включал столбцы партиционирования и bucketing. Например, стратегия распределения данных использует `dt` в качестве столбца партиционирования и `merchant_id` в качестве столбца hash bucketing. Первичный ключ также должен включать `dt` и `merchant_id`.

Таким образом, оператор CREATE TABLE для вышеупомянутой таблицы заказов может выглядеть следующим образом:

```
CREATE TABLE orders2 (
    order_id bigint NOT NULL,
    dt date NOT NULL,
    merchant_id int NOT NULL,
    user_id int NOT NULL,
    good_id int NOT NULL,
    good_name string NOT NULL,
    price int NOT NULL,
    cnt int NOT NULL,
    revenue int NOT NULL,
    state tinyint NOT NULL
)
PRIMARY KEY (order_id,dt,merchant_id)
PARTITION BY date_trunc('day', dt)
DISTRIBUTED BY HASH (merchant_id)
ORDER BY (dt,merchant_id)
PROPERTIES (
    "enable_persistent_index" = "true"
);
```

Первичный ключ

Первичный ключ таблицы используется для уникальной идентификации каждой строки в этой таблице. Один или несколько столбцов, составляющих первичный ключ, определяются в `PRIMARY KEY` и имеют ограничение UNIQUE и ограничение NOT NULL.

Примите во внимание следующие соображения о первичном ключе:

- В операторе CREATE TABLE столбцы первичного ключа должны быть определены перед другими столбцами.
- Столбцы первичного ключа должны включать столбцы партиционирования и bucketing.
- Столбцы первичного ключа поддерживают следующие типы данных: числовые (включая целые числа и BOOLEAN), строковые и дата (DATE и DATETIME).
- По умолчанию максимальная длина закодированного значения первичного ключа составляет 128 байт.
- Первичный ключ нельзя изменить после создания таблицы.
- В целях обеспечения согласованности данных значения первичного ключа не могут быть обновлены.

Индекс первичного ключа

Индекс первичного ключа используется для хранения отображения между значениями первичного ключа и местоположениями строк данных, идентифицированных значениями первичного ключа. Обычно индексы первичного ключа соответствующих tablets загружаются в память только во время загрузки данных (которая включает пакет изменений данных). Вы можете рассмотреть возможность сохранения индексов первичного ключа после комплексной оценки требований к производительности запросов и обновлений, а также памяти и диска.

Персистентный индекс первичного ключа	**Полностью хранящийся в памяти индекс первичного ключа**

Когда `enable_persistent_index` установлен в `true` (по умолчанию), индексы первичного ключа могут быть сохранены на диске. Во время загрузки небольшая часть индексов первичного ключа загружается в память, в то время как большая часть хранится на диске, чтобы избежать слишком большого использования памяти. В общем, производительность запросов и обновлений таблицы с персистентными индексами первичного ключа почти эквивалентна таковой у таблицы с полностью хранящимися в памяти индексами первичного ключа.

Если диск является SSD, рекомендуется установить значение `true`. Если диск является HDD и частота загрузки невысока, вы также можете установить значение `true`.

Начиная с версии 3.1.4, Primary Key tables, созданные в Selena shared-data clusters, поддерживают сохранение индекса на локальных дисках. А начиная с версии 3.3.2 Selena shared-data clusters дополнительно поддерживают сохранение индекса в объектном хранилище. Вы можете включить эту функцию, установив свойство таблицы `persistent_index_type` в `CLOUD_NATIVE`.

Ключ сортировки

Начиная с версии 3.0, Primary Key table отделяет ключ сортировки от Primary Key. Ключ сортировки состоит из столбцов, определённых в `ORDER BY`, и может состоять из любой комбинации столбцов, при условии, что тип данных столбцов соответствует требованиям ключа сортировки.

Во время загрузки данных данные хранятся после сортировки в соответствии с ключом сортировки. Ключ сортировки также используется для построения префиксного индекса для ускорения запросов. Рекомендуется правильно спроектировать ключ сортировки для формирования префиксного индекса, который может ускорить запросы.

К СВЕДЕНИЮ

- Если указан ключ сортировки, префиксный индекс строится на основе ключа сортировки. Если ключ сортировки не указан, префиксный индекс строится на основе Primary Key.
- После создания таблицы вы можете использовать `ALTER TABLE ... ORDER BY ...` для изменения ключа сортировки. Удаление ключа сортировки не поддерживается, а изменение типов данных столбцов сортировки не поддерживается.

Что ещё

- Для загрузки данных в созданную таблицу вы можете обратиться к Обзор загрузки, чтобы выбрать подходящий вариант загрузки.
- Если вам необходимо изменить данные в Primary Key table, вы можете обратиться к изменение данных через загрузку или использовать DML (INSERT, UPDATE и DELETE).
- Если вы хотите дополнительно ускорить запросы, вы можете обратиться к Ускорение запросов.
- Если вам необходимо изменить схему таблицы, вы можете обратиться к ALTER TABLE.
- Столбец AUTO_INCREMENT может использоваться в качестве Primary Key.

2.3.5 Duplicate Key таблица

Duplicate Key таблица является моделью по умолчанию в Selena. Если вы не указали модель при создании таблицы, по умолчанию создается Duplicate Key таблица.

При создании Duplicate Key таблицы вы можете определить ключ сортировки для этой таблицы. Если условия фильтрации содержат столбцы ключа сортировки, Selena может быстро фильтровать данные из таблицы для ускорения запросов.

Duplicate Key таблица подходит для сценариев, таких как анализ данных логов. Она поддерживает добавление новых данных, но не поддерживает изменение исторических данных.

Сценарии

Duplicate Key таблица подходит для следующих сценариев:

- Анализ необработанных данных, таких как необработанные логи и необработанные записи операций.
- Запрос данных с использованием различных методов без ограничений методом предварительной агрегации.
- Загрузка данных логов или данных временных рядов. Новые данные записываются в режиме только добавления, и существующие данные не обновляются.

Создание таблицы

Предположим, что вы хотите проанализировать данные о событиях в течение определенного временного диапазона. В этом примере создайте таблицу с именем `detail` и определите `event_time` и `event_type` в качестве столбцов ключа сортировки.

Инструкция для создания таблицы:

```
CREATE TABLE detail (
    event_time DATETIME NOT NULL COMMENT "datetime of event",
    event_type INT NOT NULL COMMENT "type of event",
    user_id INT COMMENT "id of user",
    device_code INT COMMENT "device code",
    channel INT COMMENT "")
ORDER BY (event_time, event_type);
```

Примечания по использованию

- **Ключ сортировки**

  Начиная с v1.5.2, Duplicate Key таблица поддерживает указание ключа сортировки с использованием `ORDER BY`, который может быть комбинацией любых столбцов. Если используются оба `ORDER BY` и `DUPLICATE KEY`, `DUPLICATE KEY` не действует. Если не используется ни `ORDER BY`, ни `DUPLICATE KEY`, первые три столбца таблицы используются в качестве ключа сортировки по умолчанию.

- **Bucketing**

  - **Метод bucketing**:
  Начиная с v1.5.2, Selena поддерживает случайный bucketing для Duplicate Key таблиц (метод bucketing по умолчанию). При создании таблиц или добавлении partition вам не нужно устанавливать ключ hash bucketing (то есть клаузу `DISTRIBUTED BY HASH`). До v1.5.2 Selena поддерживает только hash bucketing. Вам необходимо установить ключ hash bucketing (то есть клаузу `DISTRIBUTED BY HASH`) при создании таблиц или добавлении partition, иначе таблица не будет создана. Для получения дополнительной информации о ключах hash bucketing см. Hash Bucketing.

  - **Количество bucket**: Начиная с v1.5.2, Selena может автоматически устанавливать количество bucket (BUCKETS) при создании таблицы или добавлении partition. Вам больше не нужно вручную устанавливать количество bucket. Для получения подробной информации см. set the number of buckets.
- При создании таблицы вы можете создать индексы Bitmap или индексы Bloom Filter на всех столбцах таблицы.

Что делать дальше

После создания таблицы вы можете использовать различные методы загрузки данных для загрузки данных в Selena. Для получения информации о методах загрузки данных, поддерживаемых Selena, см. Loading options.

> **ПРИМЕЧАНИЕ**
>
> - Когда вы загружаете данные в таблицу, которая использует Duplicate Key таблицу, вы можете только добавлять данные в таблицу. Вы не можете изменять существующие данные в таблице.
> - Если загружаются две идентичные записи, Duplicate Key таблица сохраняет их как две записи, а не как одну.

2.3.6 Aggregate таблица

При создании таблицы вы можете определить ключ агрегации и указать функцию агрегации для столбца значений. Когда несколько строк данных имеют одинаковый ключ агрегации, значения в столбцах значений агрегируются. Кроме того, вы можете отдельно определить ключ сортировки. Если условия фильтрации в запросах включают ключ

сортировки, Selena может быстро фильтровать данные, повышая эффективность запросов.

В сценариях анализа и агрегации данных Aggregate таблицы могут уменьшить объем данных, которые необходимо обработать, тем самым повышая эффективность запросов.

Сценарии

Aggregate таблица хорошо подходит для сценариев статистики и аналитики данных. Несколько примеров следующие:

- Помочь провайдерам веб-сайтов или приложений анализировать объем трафика и время, которое их пользователи проводят на конкретном веб-сайте или в приложении, и общее количество посещений веб-сайта или приложения.

- Помочь рекламным агентствам анализировать общее количество кликов, общее количество просмотров и статистику потребления рекламы, которую они предоставляют для своих клиентов.

- Помочь компаниям электронной коммерции анализировать свои годовые торговые данные, чтобы определить географические бестселлеры в отдельных кварталах или месяцах.

Запрос данных и загрузка в вышеуказанных сценариях имеют следующие характеристики:

- Большинство запросов — это агрегированные запросы, такие как SUM, MAX и MIN.
- Необработанные детализированные данные не нужно извлекать.
- Исторические данные не обновляются часто. Добавляются только новые данные.

Принцип

От фазы загрузки данных до запроса данных данные в Aggregate таблицах агрегируются несколько раз следующим образом:

1. На этапе загрузки данных каждая партия данных формирует версию, когда данные загружаются в Aggregate таблицу партиями. В одной версии данные с одинаковым ключом агрегации будут агрегированы.

2. На этапе фоновой компактификации, когда файлы нескольких версий данных, которые создаются при загрузке данных, периодически компактифицируются в

большой файл, Selena агрегирует данные, которые имеют одинаковый ключ агрегации в большом файле.

3. На этапе запроса данных Selena агрегирует данные, которые имеют одинаковый ключ агрегации среди всех версий данных, прежде чем вернуть результат запроса.

Операции агрегации помогают уменьшить объем данных, которые необходимо обработать, тем самым ускоряя запросы.

Предположим, у вас есть таблица, которая использует Aggregate таблицу, и вы хотите загрузить следующие четыре необработанные записи в таблицу.

Date	Country	PV
2020.05.01	CHN	1
2020.05.01	CHN	2
2020.05.01	USA	3
2020.05.01	USA	4

Selena агрегирует четыре необработанные записи в следующие две записи при загрузке данных.

Date	Country	PV
2020.05.01	CHN	3
2020.05.01	USA	7

Создание таблицы

Предположим, что вы хотите проанализировать количество посещений пользователями из разных городов на различные веб-страницы. В этом примере создайте таблицу с именем `example_db.aggregate_tbl`, определите `site_id`, `date` и `city_code` в качестве ключа агрегации, определите `pv` в качестве столбца значений и укажите функцию SUM для столбца `pv`.

Инструкция для создания таблицы следующая:

```
CREATE TABLE aggregate_tbl (
    site_id LARGEINT NOT NULL COMMENT "id of site",
    date DATE NOT NULL COMMENT "time of event",
    city_code VARCHAR(20) COMMENT "city_code of user",
    pv BIGINT SUM DEFAULT "0" COMMENT "total page views"
)
AGGREGATE KEY(site_id, date, city_code)
DISTRIBUTED BY HASH(site_id);
```

> **NOTICE**
>
> - При создании таблицы вы должны указать столбец bucketing с использованием клаузы `DISTRIBUTED BY HASH`. Для получения подробной информации см. bucketing.
> - Начиная с v1.5.2, Selena может автоматически устанавливать количество bucket (BUCKETS) при создании таблицы или добавлении partition. Вам больше не нужно вручную устанавливать количество bucket. Для получения подробной информации см. set the number of buckets.

Обобщенные состояния функций агрегации

> *Beta* **BETA FEATURE**
>
> Рекомендации по использованию бета-функций

Selena поддерживает обобщенные состояния функций агрегации начиная с v1.5.2.

В анализе и суммировании данных Aggregate таблицы уменьшают объем данных, обрабатываемых во время запросов, повышая производительность запросов. Для больших наборов данных Aggregate таблицы очень эффективны, поскольку они суммируют данные по измерениям перед запросом. Они также служат важным методом для инкрементного вычисления функций агрегации в Selena. Однако в более ранних версиях поддержка ограничена встроенными функциями, такими как `SUM`, `MAX`, `MIN`, `REPLACE`, `HLL_UNION`, `PERCENTILE_UNION` и `BITMAP_UNION`, в то время как теоретически все встроенные функции агрегации могут использоваться в aggregate таблицах. Для решения этого ограничения введены обобщенные состояния агрегации для поддержки хранения всех состояний встроенных функций.

Хранение обобщенных состояний агрегации

Вы можете определить обобщенные состояния агрегации в Aggregate таблицах, указав имя функции и типы входных параметров для уникальной идентификации функции агрегации. Тип столбца будет автоматически определен как тип промежуточного состояния функции агрегации.

Определение:

```
col_name agg_func_name(parameter1_type, [parameter2_type], ...)
```

- **col_name**: Имя столбца.
- **agg_func_name**: Имя функции агрегации, промежуточные состояния которой необходимо сохранить.
- **parameter_type**: Тип входного параметра функции агрегации. Функция может быть однозначно идентифицирована с помощью типа параметра.

> **ПРИМЕЧАНИЕ**
>
> - Поддерживаются только встроенные функции Selena с по крайней мере одним параметром. Java и Python UDAF не поддерживаются.
> - Для стабильности и расширяемости тип столбца состояния агрегации всегда Nullable (за исключением функции count) и не может быть изменен.
> - Значения параметров не нужны для определения многопараметрических функций, поскольку типы могут быть выведены, а значения параметров не участвуют в вычислении.
> - Сложные параметры, такие как ORDER BY и DISTINCT, не поддерживаются.
> - Поддержка конкретных встроенных функций, таких как `GROUP_CONCAT`, `WINDOW_FUNNEL` и `APPROX_TOP_K`, все еще находится в разработке. Они будут поддерживаться в будущих релизах. Для получения подробной информации см. FunctionSet.java#UNSUPPORTED_AGG_STATE_FUNCTIONS.

Пример:

```
CREATE TABLE test_create_agg_table (
  dt VARCHAR(10),
  -- Define generic aggregate state storage.
  hll_sketch_agg ds_hll_count_distinct(varchar),
  avg_agg avg(bigint),
  array_agg_agg array_agg(int),
  min_by_agg min_by(varchar, bigint)
)
AGGREGATE KEY(dt)
PARTITION BY (dt)
DISTRIBUTED BY HASH(dt) BUCKETS 4;
```

Функции-комбинаторы

Обобщенные состояния агрегации используют функции-комбинаторы для инкапсуляции вычисления промежуточного состояния и потока.

Функция-комбинатор `_state`

Функция `_state` преобразует входные параметры в типы промежуточного состояния.

Определение:

```
agg_intermediate_type {agg_func_name}_state(input_col1, [input_col2], ...)
```

- **agg_func_name**: Имя функции агрегации, которая должна передать входные параметры в типы промежуточного состояния.
- **input_col1/col2**: Входные столбцы функции агрегации.
- **agg_intermediate_type**: Тип возвращаемого значения функции `_state`, то есть тип промежуточного состояния функции агрегации.

> **ПРИМЕЧАНИЕ**
>
> `_state` — это скалярная функция. Вам не нужно определять столбец агрегации для вычисления состояния входного параметра.

Пример:

```
CREATE TABLE t1 (
  id BIGINT NOT NULL,
  province VARCHAR(64),
  age SMALLINT,
  dt VARCHAR(10) NOT NULL
)
DUPLICATE KEY(id)
PARTITION BY (dt)
DISTRIBUTED BY HASH(id) BUCKETS 4;

INSERT INTO t1 SELECT generate_series, generate_series, generate_series %
10, "2024-07-24" FROM table(generate_series(1, 100));

-- Transfer the data in t1 with _state combinator function, and insert it
into the Aggregate table.
INSERT INTO test_create_agg_table
SELECT
    dt,
    ds_hll_count_distinct_state(id),
    avg_state(id),
    array_agg_state(id),
    min_by_state(province, id)
FROM t1;
```

Функция-комбинатор `_union`

Функция `_union` объединяет несколько столбцов промежуточного состояния в одно состояние.

Определение:

```
-- Union multiple aggregate intermediate states.
agg_intermediate_type {agg_func_name}_union(input_col)
```

- **agg_func_name**: Имя функции агрегации.
- **input_col**: Входные столбцы функции агрегации. Тип входного столбца — это тип промежуточного состояния функции агрегации. Вы можете получить его с помощью функций `_state`.
- **agg_intermediate_type**: Тип возвращаемого значения функции `_union`, то есть тип промежуточного состояния функции агрегации.

> **ПРИМЕЧАНИЕ**
>
> `_union` — это функция агрегации. Она возвращает тип промежуточного состояния вместо типа окончательного результата функции.

Пример:

```
-- Case 1: Union the intermediate states of the Aggregate table.
SELECT
    dt,
    ds_hll_count_distinct_union(hll_sketch_agg),
    avg_union(avg_agg),
    array_agg_union(array_agg_agg),
    min_by_union(min_by_agg)
FROM test_create_agg_table
GROUP BY dt
LIMIT 1;

-- Case 2: Union the intermediate states input by the _state combinator
function.
SELECT
    dt,
    ds_hll_count_distinct_union(ds_hll_count_distinct_state(id)),
    avg_union(avg_state(id)),
    array_agg_union(array_agg_state(id)),
    min_by_union(min_by_state(province, id))
FROM t1
GROUP BY dt
LIMIT 1;
```

Функция-комбинатор `_merge`

Функция-комбинатор `_merge` инкапсулирует функции агрегации как обобщенную функцию агрегации для вычисления окончательного результата агрегации нескольких промежуточных состояний.

Определение:

```
-- Merge multiple aggregate intermediate states.
agg_result_type {agg_func_name}_merge(input_col)
```

- **agg_func_name**: Имя функции агрегации.
- **input_col**: Входные столбцы функции агрегации. Тип входного столбца — это тип промежуточного состояния функции агрегации. Вы можете получить его с помощью функций `_state`.
- **agg_intermediate_type**: Тип возвращаемого значения функции `_merge`, то есть окончательный результат агрегации функции агрегации.

Пример:

```
-- Case 1: Merge the intermediate states of the Aggregate table to obtain
the final aggregation result.
SELECT
    dt,
    ds_hll_count_distinct_merge(hll_sketch_agg),
    avg_merge(avg_agg),
    array_agg_merge(array_agg_agg),
    min_by_merge(min_by_agg)
FROM test_create_agg_table
GROUP BY dt
LIMIT 1;

-- Case 2:  Merge the intermediate states input by the _state combinator
function to obtain the final aggregation result.
SELECT
    dt,
    ds_hll_count_distinct_merge(ds_hll_count_distinct_state(id)),
    avg_merge(avg_state(id)),
    array_agg_merge(array_agg_state(id)),
    min_by_merge(min_by_state(province, id))
FROM t1
GROUP BY dt
LIMIT 1;
```

Использование обобщенных состояний агрегации в материализованных представлениях

Обобщенные состояния агрегации могут использоваться в синхронных и асинхронных материализованных представлениях для ускорения производительности запросов со сверткой состояний агрегации.

Обобщенные состояния агрегации в синхронных материализованных представлениях

Пример:

```
-- Create a synchronous materialized view test_mv1 to store aggregate
states.
CREATE MATERIALIZED VIEW test_mv1
AS
SELECT
    dt,
    -- Original aggregate functions.
    min(id) AS min_id,
    max(id) AS max_id,
    sum(id) AS sum_id,
    bitmap_union(to_bitmap(id)) AS bitmap_union_id,
    hll_union(hll_hash(id)) AS hll_union_id,
    percentile_union(percentile_hash(id)) AS percentile_union_id,
    -- Generic aggregate state functions.
    ds_hll_count_distinct_union(ds_hll_count_distinct_state(id)) AS hll_id,
    avg_union(avg_state(id)) AS avg_id,
    array_agg_union(array_agg_state(id)) AS array_agg_id,
    min_by_union(min_by_state(province, id)) AS min_by_province_id
FROM t1
GROUP BY dt;

-- Wait until rollup creation finishes.
show alter table rollup;

-- Direct queries against the aggregate function will be transparently
accelerated by test_mv1.
SELECT
    dt,
    min(id),
    max(id),
    sum(id),
    bitmap_union_count(to_bitmap(id)), -- count(distinct id)
    hll_union_agg(hll_hash(id)), -- approx_count_distinct(id)
    percentile_approx(id, 0.5),
    ds_hll_count_distinct(id),
    avg(id),
    array_agg(id),
    min_by(province, id)
FROM t1
WHERE dt >= '2024-01-01'
GROUP BY dt;

-- Direct queries against the aggregate function and the rollup will also
be transparently accelerated by test_mv1.
SELECT
```

```
    min(id),
    max(id),
    sum(id),
    bitmap_union_count(to_bitmap(id)), -- count(distinct id)
    hll_union_agg(hll_hash(id)), -- approx_count_distinct(id)
    percentile_approx(id, 0.5),
    ds_hll_count_distinct(id),
    avg(id),
    array_agg(id),
    min_by(province, id)
FROM t1
WHERE dt >= '2024-01-01';

DROP MATERIALIZED VIEW test_mv1;
```

Обобщенные состояния агрегации в асинхронных материализованных представлениях

Пример:

```
-- Create an asynchronous materialized view test_mv2 to store aggregate
states.
CREATE MATERIALIZED VIEW test_mv2
PARTITION BY (dt)
DISTRIBUTED BY RANDOM
AS
SELECT
    dt,
    -- Original aggregate functions.
    min(id) AS min_id,
    max(id) AS max_id,
    sum(id) AS sum_id,
    bitmap_union(to_bitmap(id)) AS bitmap_union_id,
    hll_union(hll_hash(id)) AS hll_union_id,
    percentile_union(percentile_hash(id)) AS percentile_union_id,
    -- Generic aggregate state functions.
    ds_hll_count_distinct_union(ds_hll_count_distinct_state(id)) AS hll_id,
    avg_union(avg_state(id)) AS avg_id,
    array_agg_union(array_agg_state(id)) AS array_agg_id,
    min_by_union(min_by_state(province, id)) AS min_by_province_id
FROM t1
GROUP BY dt;

-- Refresh the materialized view.
REFRESH MATERIALIZED VIEW test_mv2 WITH SYNC MODE;

-- Direct queries against the aggregate function will be transparently
accelerated by test_mv2.
SELECT
    dt,
    min(id),
    max(id),
    sum(id),
    bitmap_union_count(to_bitmap(id)), -- count(distinct id)
    hll_union_agg(hll_hash(id)), -- approx_count_distinct(id)
    percentile_approx(id, 0.5),
    ds_hll_count_distinct(id),
    avg(id),
    array_agg(id),
    min_by(province, id)
FROM t1
WHERE dt >= '2024-01-01'
GROUP BY dt;

SELECT
```

```
    min(id),
    max(id),
    sum(id),
    bitmap_union_count(to_bitmap(id)), -- count(distinct id)
    hll_union_agg(hll_hash(id)), -- approx_count_distinct(id)
    percentile_approx(id, 0.5),
    ds_hll_count_distinct(id),
    avg(id),
    array_agg(id),
    min_by(province, id)
FROM t1
WHERE dt >= '2024-01-01';
```

Примечания по использованию

- **Ключ агрегации**:
  - В инструкции CREATE TABLE ключ агрегации должен быть определен перед другими столбцами.
  - Ключ агрегации может быть явно определен с использованием `AGGREGATE KEY`. `AGGREGATE KEY` должен включать все столбцы, кроме столбцов значений, иначе таблица не будет создана.

    Если ключ агрегации не определен явно с использованием `AGGREGATE KEY`, все столбцы, кроме столбцов значений, считаются ключом агрегации по умолчанию.
  - Ключ агрегации имеет ограничение уникальности.
- **Столбец значений**: Определите столбец как столбец значений, указав функцию агрегации после имени столбца. Этот столбец обычно содержит данные, которые необходимо агрегировать.
- **Функция агрегации**: Функция агрегации, используемая для столбца значений. Для поддерживаемых функций агрегации для Aggregate таблиц см. CREATE TABLE.
- **Ключ сортировки**
  - Начиная с v1.5.2, ключ сортировки отделен от ключа агрегации в Aggregate таблице. Aggregate таблица поддерживает указание ключа сортировки с

использованием `ORDER BY` и указание ключа агрегации с использованием `AGGREGATE KEY`. Столбцы в ключе сортировки и ключе агрегации должны быть одинаковыми, но порядок столбцов не обязательно должен быть одинаковым.

  - При выполнении запросов столбцы ключа сортировки фильтруются перед агрегацией нескольких версий данных, тогда как столбцы значений фильтруются после агрегации нескольких версий данных. Поэтому мы рекомендуем определять столбцы, которые часто используются в качестве условий фильтрации, и определять эти столбцы как ключ сортировки. Таким образом, фильтрация данных может начаться перед агрегацией нескольких версий данных для повышения производительности запросов.

- При создании таблицы вы можете создавать только индексы Bitmap или индексы Bloom Filter на ключевых столбцах таблицы.

Что делать дальше

После создания таблицы вы можете использовать различные методы загрузки данных для загрузки данных в Selena. Для получения информации о методах загрузки данных, поддерживаемых Selena, см. Loading options.

> Note: Когда вы загружаете данные в таблицу, которая использует Aggregate таблицу, вы можете обновлять только все столбцы таблицы. Например, при обновлении вышеупомянутой таблицы `example_db.aggregate_tbl` вы должны обновить все ее столбцы, которые являются `site_id`, `date`, `city_code` и `pv`.

2.3.7 Таблица с уникальным ключом

При создании таблицы необходимо определить уникальный ключ. Когда несколько строк данных имеют одинаковый уникальный ключ, значения в колонках значений заменяются. При выполнении запросов возвращаются последние данные из группы данных с одинаковым уникальным ключом. Кроме того, вы можете определить ключ сортировки отдельно. Если условия фильтрации в запросах включают ключ сортировки, Selena может быстро отфильтровать данные, повышая эффективность запросов.

Таблица с уникальным ключом поддерживает обновление данных в реальном времени и частое обновление. Однако она постепенно заменяется таблицей с первичным ключом.

Сценарии использования

Таблица с уникальным ключом подходит для бизнес-сценариев, в которых данные необходимо часто обновлять в реальном времени. Например, в сценариях электронной коммерции может быть размещено сотни миллионов заказов в день, и статусы заказов часто меняются.

Принцип работы

Таблицу с уникальным ключом можно рассматривать как специальную агрегатную таблицу, в которой для колонок значений указана агрегатная функция REPLACE, возвращающая самую последнюю запись из группы записей с одинаковым уникальным ключом.

При загрузке данных в таблицу с уникальным ключом данные разделяются на несколько пакетов. Каждому пакету присваивается номер версии. Таким образом, записи с одинаковым уникальным ключом могут быть включены в несколько версий. Для запросов возвращаются данные из самой последней версии (то есть запись с наибольшим номером версии).

Как показано в следующей таблице, `ID` является уникальным ключом, `value` — колонкой значения, а `_version` содержит номера версий данных, сгенерированные внутри Selena. В этом примере запись с `ID`, равным 1, загружается двумя пакетами, номера версий которых равны `1` и `2`, а запись с `ID`, равным `2`, загружается тремя пакетами, номера версий которых равны `3`, `4` и `5`.

ID	value	_version
1	100	1
1	101	2
2	100	3
2	101	4
2	102	5

При запросе записи с `ID`, равным `1`, возвращается самая последняя запись с наибольшим номером версии, который в данном случае равен `2`. При запросе записи с `ID`, равным `2`, возвращается самая последняя запись с наибольшим номером версии, который в данном случае равен `5`. В следующей таблице показаны записи, возвращаемые двумя запросами:

ID	value
1	101
2	102

Создание таблицы

В сценариях электронной коммерции часто требуется собирать и анализировать статусы заказов по датам. В этом примере создайте таблицу с именем `orders` для хранения заказов, определите `create_time` и `order_id`, которые часто используются в качестве условий для фильтрации заказов, в качестве колонок уникального ключа, и определите две другие колонки, `order_state` и `total_price`, в качестве колонок значений. Таким образом, заказы могут обновляться в реальном времени по мере изменения их статусов и могут быть быстро отфильтрованы для ускорения запросов.

Инструкция для создания таблицы выглядит следующим образом:

```
CREATE TABLE orders (
    create_time DATE NOT NULL COMMENT "create time of an order",
    order_id BIGINT NOT NULL COMMENT "id of an order",
    order_state INT COMMENT "state of an order",
    total_price BIGINT COMMENT "price of an order"
)
UNIQUE KEY(create_time, order_id)
DISTRIBUTED BY HASH(order_id);
```

> **ВАЖНО**
>
> - При создании таблицы необходимо указать колонку bucketing с помощью предложения `DISTRIBUTED BY HASH`. Для получения подробной информации см. bucketing.
> - Начиная с версии 2.5.7, Selena может автоматически устанавливать количество buckets (BUCKETS) при создании таблицы или добавлении partition. Вам больше не нужно вручную устанавливать количество buckets. Для получения подробной информации см. set the number of buckets.

Примечания по использованию

- **Уникальный ключ**:

  - В инструкции CREATE TABLE уникальный ключ должен быть определен перед другими колонками.
  - Уникальный ключ должен быть явно определен с использованием `UNIQUE KEY`.
  - Уникальный ключ имеет ограничение уникальности.
- **Ключ сортировки**:
  - Начиная с версии 3.3.0, ключ сортировки отделен от уникального ключа в таблице с уникальным ключом. Таблица с уникальным ключом поддерживает указание ключа сортировки с помощью `ORDER BY` и указание уникального ключа с помощью `UNIQUE KEY`. Колонки в ключе сортировки и уникальном ключе должны быть одинаковыми, но порядок колонок не обязательно должен быть одинаковым.
  - При выполнении запросов данные могут быть отфильтрованы на основе ключей сортировки перед агрегацией. Однако данные могут быть отфильтрованы на основе колонок значений после многоверсионной агрегации. Поэтому рекомендуется использовать часто фильтруемые поля в качестве ключей сортировки для фильтрации данных перед агрегацией и тем самым повышения производительности запросов.
- При создании таблицы вы можете создавать только индексы Bitmap или индексы Bloom Filter на ключевых колонках таблицы.

Что делать дальше

После создания таблицы вы можете использовать различные методы загрузки данных для загрузки данных в Selena. Для получения информации о методах загрузки данных, поддерживаемых Selena, см. Опции загрузки.

ПРИМЕЧАНИЕ

- При загрузке данных в таблицу, использующую таблицу с уникальным ключом, вы можете обновлять только все колонки таблицы. Например, при обновлении предыдущей таблицы `orders` вы должны обновить все её колонки: `create_time`, `order_id`, `order_state` и `total_price`.
- При запросе данных из таблицы, использующей таблицу с уникальным ключом, Selena необходимо агрегировать записи нескольких версий данных. В этой ситуации большое количество версий данных снижает производительность запросов. Поэтому мы рекомендуем указывать подходящую частоту загрузки данных в таблицу, чтобы соответствовать вашим требованиям для аналитики данных в реальном времени, избегая при этом большого количества версий данных. Если вам требуются данные на уровне минут, вы можете указать частоту загрузки 1 минута вместо частоты загрузки 1 секунда.

2.4 Распределение данных

Настройка правильного партиционирования и bucketing'a при создании таблицы помогает обеспечить равномерное распределение данных. Равномерное распределение данных означает разделение данных на подмножества в соответствии с определенными правилами и их равномерное распределение по различным узлам. Это также позволяет уменьшить объем сканируемых данных и в полной мере использовать возможности кластера для параллельной обработки, тем самым улучшая производительность запросов.

ПРИМЕЧАНИЕ

- После того, как распределение данных было указано при создании таблицы и паттерны запросов или характеристики данных в бизнес-сценарии изменились, начиная с версии 3.2 Selena поддерживает изменение определенных свойств, связанных с распределением данных, после создания таблицы, чтобы соответствовать требованиям производительности запросов в последних бизнес-сценариях.
- Начиная с версии 3.1, вам не нужно указывать bucketing key в клаузе DISTRIBUTED BY при создании таблицы или добавлении партиции. Selena поддерживает случайный bucketing, который случайным образом распределяет

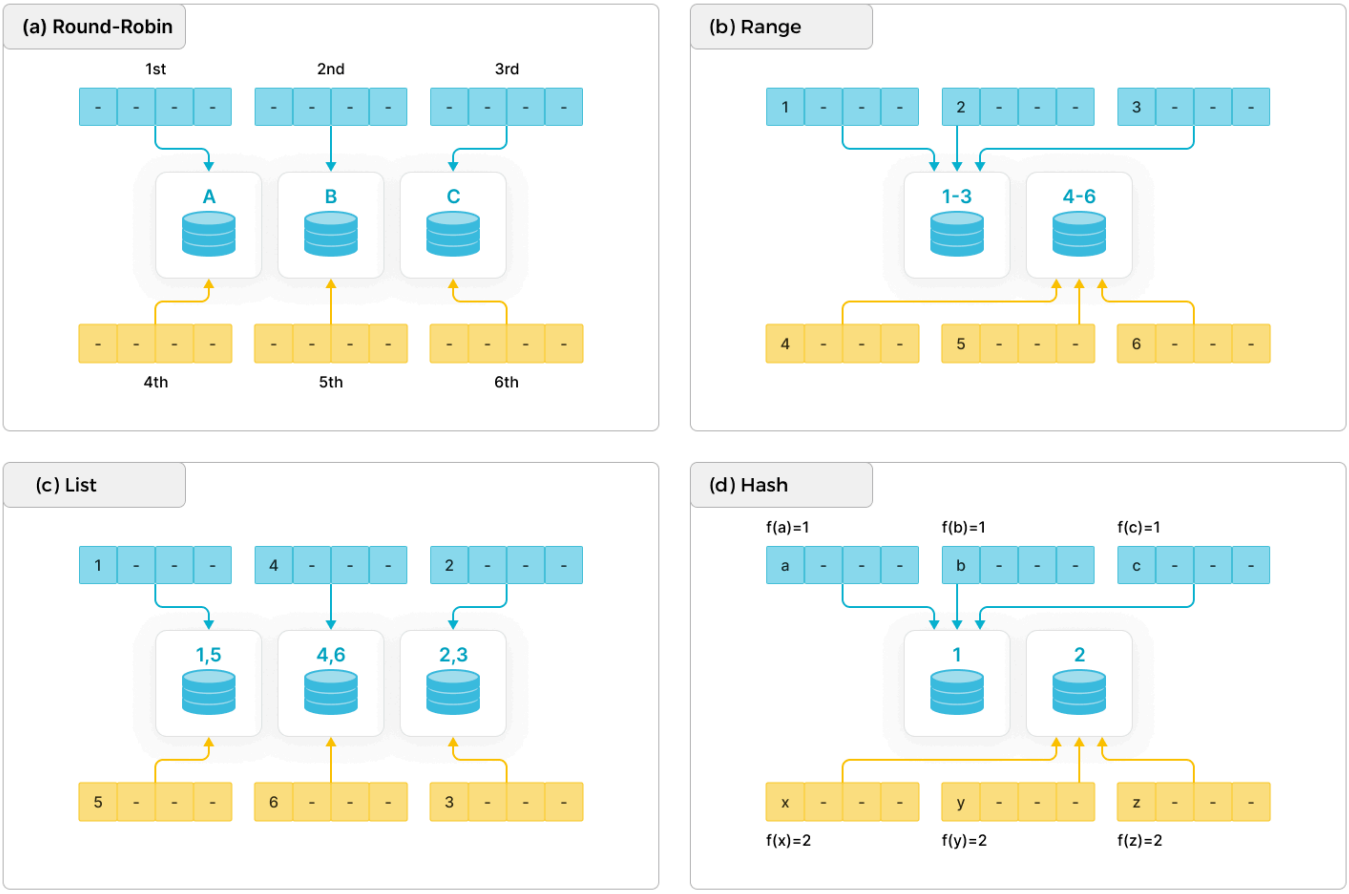
данные по всем bucket'ам. Для получения дополнительной информации см. Случайный bucketing.

- Начиная с версии 2.5.7, вы можете не устанавливать количество bucket'ов вручную при создании таблицы или добавлении партиции. Selena может автоматически установить количество bucket'ов (BUCKETS). Однако, если производительность не соответствует вашим ожиданиям после того, как Selena автоматически установила количество bucket'ов, и вы знакомы с механизмом bucketing'а, вы все еще можете установить количество bucket'ов вручную.

2.4.1 Введение

Методы распределения в общем случае

Современные системы распределенных баз данных обычно используют следующие базовые методы распределения: Round-Robin, Range, List и Hash.



- **Round-Robin**: распределяет данные по различным узлам циклически.
- **Range**: распределяет данные по различным узлам на основе диапазонов значений партиционирующих колонок. Как показано на диаграмме, диапазоны [1-3] и [4-6] соответствуют различным узлам.

- **List**: распределяет данные по различным узлам на основе дискретных значений партиционирующих колонок, таких как пол и провинция. Каждое дискретное значение сопоставлено с узлом, и несколько различных значений могут быть сопоставлены с одним и тем же узлом.
- **Hash**: распределяет данные по различным узлам на основе хеш-функции.

Для достижения более гибкого партиционирования данных, помимо использования одного из вышеперечисленных методов распределения данных, вы также можете комбинировать эти методы на основе конкретных бизнес-требований. Распространенные комбинации включают Hash+Hash, Range+Hash и Hash+List.

Методы распределения в Selena

Selena поддерживает как отдельное, так и комбинированное использование методов распределения данных.

> **ПРИМЕЧАНИЕ**
>
> В дополнение к общим методам распределения, Selena также поддерживает Random распределение для упрощения конфигурации bucketing'a.

Также Selena распределяет данные путем реализации двухуровневого метода партиционирования + bucketing'a.

- Первый уровень - это партиционирование: Данные внутри таблицы могут быть партиционированы. Поддерживаемые методы партиционирования - это expression partitioning, range partitioning и list partitioning. Или вы можете не использовать партиционирование (вся таблица рассматривается как одна партиция).
- Второй уровень - это bucketing: Данные в партиции необходимо далее распределить на более мелкие bucket'ы. Поддерживаемые методы bucketing'a - это hash и random bucketing.

Метод распределения	Метод партиционирования и bucketing'a	Описание
Random распределение	Random bucketing	Вся таблица рассматривается как партиция. Данные в таблице случайным образом распределяются по различным bucket'ам. Это метод распределения данных по умолчанию.
Hash распределе	Hash bucketing	Вся таблица рассматривается как партиция. Данные в таблице распределяются по соответствующим bucket'ам на основе хеш-

Метод распределения	Метод партиционирования и bucketing'a	Описание
ние		значений bucketing key данных с использованием хеш-функции.
Range+Random распределение	1. Range партиционирование 2. Random bucketing	1. Данные в таблице распределяются по соответствующим партициям на основе диапазонов, в которые попадают значения партиционирующих колонок. 2. Данные в партиции случайным образом распределяются по различным bucket'ам.
Range+Hash распределение	1. Range партиционирование 2. Hash bucketing	1. Данные в таблице распределяются по соответствующим партициям на основе диапазонов, в которые попадают значения партиционирующих колонок. 2. Данные в партиции распределяются по соответствующим bucket'ам на основе хеш-значений bucketing key данных с использованием хеш-функции.
List+Random распределение	1. Expression партиционирование или list партиционирование 2. Random bucketing	1. Данные в таблице партиционируются на основе списков значений, к которым принадлежат значения партиционирующих колонок. 2. Данные в партиции случайным образом распределяются по различным bucket'ам.
List+Hash распределение	1. Expression партиционирование или List партиционирование 2. Hash bucketing	1. Данные в таблице партиционируются на основе списков значений, к которым принадлежат значения партиционирующих колонок. 2. Данные в партиции распределяются по соответствующим bucket'ам на основе хеш-значений bucketing key данных с использованием хеш-функции.

- **Random распределение**

  Если вы не настраиваете методы партиционирования и bucketing'a при создании таблицы, по умолчанию используется random распределение. Этот метод распределения в настоящее время может использоваться только для создания таблицы Duplicate Key.

```
CREATE TABLE site_access1 (
    event_day DATE,
    site_id INT DEFAULT '10',
    pv BIGINT DEFAULT '0' ,
    city_code VARCHAR(100),
    user_name VARCHAR(32) DEFAULT ''
)
DUPLICATE KEY (event_day,site_id,pv);
-- Поскольку методы партиционирования и bucketing'a не настроены, по
умолчанию используется random распределение.
```

- **Hash распределение**

```
CREATE TABLE site_access2 (
    event_day DATE,
    site_id INT DEFAULT '10',
    city_code SMALLINT,
    user_name VARCHAR(32) DEFAULT '',
    pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY (event_day, site_id, city_code, user_name)
-- Использовать hash bucketing в качестве метода bucketing'a и
необходимо указать bucketing key.
DISTRIBUTED BY HASH(event_day,site_id);
```

- **Range+Random распределение** (Этот метод распределения в настоящее время может использоваться только для создания таблицы Duplicate Key.)

```
CREATE TABLE site_access3 (
    event_day DATE,
    site_id INT DEFAULT '10',
    pv BIGINT DEFAULT '0' ,
    city_code VARCHAR(100),
    user_name VARCHAR(32) DEFAULT ''
)
DUPLICATE KEY(event_day,site_id,pv)
-- Использовать expression партиционирование в качестве метода
партиционирования и настроить выражение временной функции.
-- Вы также можете использовать range партиционирование.
PARTITION BY date_trunc('day', event_day);
-- Поскольку метод bucketing'a не настроен, по умолчанию используется
random bucketing.
```

- **Range+Hash распределение**

```
CREATE TABLE site_access4 (
    event_day DATE,
    site_id INT DEFAULT '10',
    city_code VARCHAR(100),
    user_name VARCHAR(32) DEFAULT '',
    pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(event_day, site_id, city_code, user_name)
-- Использовать expression партиционирование в качестве метода
партиционирования и настроить выражение временной функции.
-- Вы также можете использовать range партиционирование.
PARTITION BY date_trunc('day', event_day)
-- Использовать hash bucketing в качестве метода bucketing'a и
необходимо указать bucketing key.
DISTRIBUTED BY HASH(event_day, site_id);
```

- **List+Random распределение** (Этот метод распределения в настоящее время может использоваться только для создания таблицы Duplicate Key.)

```
CREATE TABLE t_recharge_detail1 (
    id bigint,
    user_id bigint,
    recharge_money decimal(32,2),
    city varchar(20) not null,
    dt date not null
)
DUPLICATE KEY(id)
-- Использовать expression партиционирование в качестве метода
партиционирования и указать партиционирующую колонку.
-- Вы также можете использовать list партиционирование.
PARTITION BY (city);
-- Поскольку метод bucketing'a не настроен, по умолчанию используется
random bucketing.
```

- **List+Hash распределение**

```
CREATE TABLE t_recharge_detail2 (
    id bigint,
    user_id bigint,
    recharge_money decimal(32,2),
    city varchar(20) not null,
    dt date not null
)
DUPLICATE KEY(id)
-- Использовать expression партиционирование в качестве метода
партиционирования и указать партиционирующую колонку.
-- Вы также можете использовать list партиционирование.
PARTITION BY (city)
-- Использовать hash bucketing в качестве метода bucketing'a и
необходимо указать bucketing key.
DISTRIBUTED BY HASH(city,id);
```

Партиционирование

Метод партиционирования разделяет таблицу на несколько партиций. Партиционирование в основном используется для разделения таблицы на различные единицы управления (партиции) на основе партиционирующего ключа. Вы можете установить стратегию хранения для каждой партиции, включая количество bucket'ов, стратегию хранения горячих и холодных данных, тип носителя хранения и количество реплик. Selena позволяет использовать различные типы носителей хранения в рамках

кластера. Например, вы можете хранить последние данные на твердотельных накопителях (SSD) для повышения производительности запросов, а исторические данные - на жестких дисках SATA для снижения затрат на хранение.

Метод партиционирования	Сценарии	Методы создания партиций
Expression партиционирование (рекомендуется)	Ранее известное как автоматическое партиционирование. Этот метод партиционирования более гибкий и простой в использовании. Он подходит для большинства сценариев, включая запросы и управление данными на основе непрерывных диапазонов дат или перечисляемых значений.	Автоматически создается во время загрузки данных
Range партиционирование (Legacy)	Типичный сценарий - хранение простых упорядоченных данных, которые часто запрашиваются и управляются на основе непрерывных диапазонов дат/чисел. Например, в некоторых особых случаях исторические данные необходимо партиционировать по месяцам, в то время как последние данные необходимо партиционировать по дням.	Создается вручную, динамически или в пакетном режиме
List партиционирование (Legacy)	Типичный сценарий - запрос и управление данными на основе перечисляемых значений, и партиция должна включать данные с различными значениями для каждой партиционирующей колонки. Например, если вы часто запрашиваете и управляете данными на основе стран и городов, вы можете использовать этот метод и выбрать `city` в качестве партиционирующей колонки. Таким образом, партиция может хранить данные для нескольких городов, принадлежащих одной стране.	Создается вручную

Как выбрать партиционирующие колонки и гранулярность

- **Партиционирующий ключ состоит из одной или нескольких партиционирующих колонок**. Выбор правильной партиционирующей колонки может эффективно уменьшить объем данных, сканируемых во время запросов. В большинстве бизнес-систем партиционирование на основе времени обычно применяется для решения определенных проблем, вызванных удалением устаревших данных, и для облегчения управления многоуровневым хранением горячих и холодных данных. В этом случае вы можете использовать expression партиционирование или range партиционирование и указать колонку времени в качестве партиционирующей колонки. Кроме того, если данные часто запрашиваются и управляются на основе значений ENUM, вы можете использовать expression партиционирование или list

партиционирование и указать колонку, включающую эти значения, в качестве партиционирующей колонки.

- При выборе гранулярности партиционирования необходимо учитывать объем данных, паттерны запросов и гранулярность управления данными.
  - Пример 1: Если ежемесячный объем данных в таблице невелик, партиционирование по месяцам может уменьшить количество метаданных по сравнению с партиционированием по дням, тем самым уменьшая потребление ресурсов на управление метаданными и планирование.
  - Пример 2: Если ежемесячный объем данных в таблице велик, и запросы в основном запрашивают данные определенных дней, партиционирование по дням может эффективно уменьшить объем данных, сканируемых во время запросов.
  - Пример 3: Если данные должны истекать ежедневно, рекомендуется партиционирование по дням.

Bucketing

Метод bucketing'a разделяет партицию на несколько bucket'ов. Данные в bucket'e называются tablet.

Поддерживаемые методы bucketing'a - это random bucketing (начиная с версии 3.1) и hash bucketing.

- Random bucketing: При создании таблицы или добавлении партиций вам не нужно устанавливать bucketing key. Данные внутри партиции случайным образом распределяются по различным bucket'ам.

- Hash Bucketing: При создании таблицы или добавлении партиций вам необходимо указать bucketing key. Данные внутри одной и той же партиции делятся на bucket'ы на основе значений bucketing key, и строки с одинаковым значением в bucketing key распределяются по соответствующему и уникальному bucket'у.

Количество bucket'ов: По умолчанию Selena автоматически устанавливает количество bucket'ов (начиная с версии 2.5.7). Вы также можете вручную установить количество bucket'ов. Для получения дополнительной информации см. определение количества bucket'ов.

2.4.2 Создание и управление партициями

Создание партиций

Expression партиционирование (рекомендуется)

> **ВНИМАНИЕ**
>
> Режим shared-data Selena поддерживает выражение временной функции начиная с версии 3.1.0 и выражение колонки начиная с версии 3.1.1.

Начиная с версии 3.0, Selena поддерживает expression партиционирование (ранее известное как автоматическое партиционирование), которое более гибкое и простое в использовании. Этот метод партиционирования подходит для большинства сценариев, таких как запросы и управление данными на основе непрерывных диапазонов дат или значений ENUM.

Вам нужно только настроить выражение партиционирования при создании таблицы, и Selena автоматически создаст партиции во время загрузки данных. Вам больше не нужно вручную создавать множество партиций заранее, ни настраивать свойства динамического партиционирования.

Начиная с версии 3.4, expression партиционирование дополнительно оптимизировано для унификации всех стратегий партиционирования и поддерживает более сложные решения. Оно рекомендуется в большинстве случаев и заменит другие стратегии партиционирования в будущих релизах.

Пример 1: Использование простого выражения временной функции с колонкой DATETIME.

```
CREATE TABLE site_access(
    event_day DATETIME NOT NULL,
    site_id INT DEFAULT '10',
    city_code VARCHAR(100),
    user_name VARCHAR(32) DEFAULT '',
    pv BIGINT DEFAULT '0'
)
DUPLICATE KEY(event_day, site_id, city_code, user_name)
PARTITION BY time_slice(event_day, INTERVAL 7 day)
DISTRIBUTED BY HASH(event_day, site_id)
```

Пример 2: Использование выражения колонки с несколькими колонками.

```
CREATE TABLE t_recharge_detail1 (
    id bigint,
    user_id bigint,
    recharge_money decimal(32,2),
    city varchar(20) not null,
    dt varchar(20) not null
)
DUPLICATE KEY(id)
PARTITION BY dt,city
DISTRIBUTED BY HASH(`id`);
```

Пример 3: Использование сложного выражения временной функции с колонкой Unix timestamp.

```
CREATE TABLE orders (
    ts BIGINT NOT NULL,
    id BIGINT NOT NULL,
    city STRING NOT NULL
)
PARTITION BY from_unixtime(ts,'%Y%m%d');
```

Пример 4: Использование смешанного выражения временной функции и выражения колонки.

```
CREATE TABLE orders (
    ts BIGINT NOT NULL,
    id BIGINT NOT NULL,
    city STRING NOT NULL
)
PARTITION BY from_unixtime(ts,'%Y%m%d'), city;
```

Range партиционирование

Range партиционирование подходит для хранения простых непрерывных данных, таких как данные временных рядов или непрерывные числовые данные.

Ручное создание партиций

Определите отображение между каждой партицией и диапазоном значений партиционирующей колонки.

- **Партиционирующая колонка имеет тип date.**

```
CREATE TABLE site_access(
    event_day DATE,
    site_id INT,
    city_code VARCHAR(100),
    user_name VARCHAR(32),
    pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(event_day, site_id, city_code, user_name)
PARTITION BY RANGE(event_day)(
    PARTITION p1 VALUES LESS THAN ("2020-01-31"),
    PARTITION p2 VALUES LESS THAN ("2020-02-29"),
    PARTITION p3 VALUES LESS THAN ("2020-03-31")
)
DISTRIBUTED BY HASH(site_id);
```

- **Партиционирующая колонка имеет целочисленный тип.**

```
CREATE TABLE site_access(
    datekey INT,
    site_id INT,
    city_code SMALLINT,
    user_name VARCHAR(32),
    pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(datekey, site_id, city_code, user_name)
PARTITION BY RANGE (datekey) (
    PARTITION p1 VALUES LESS THAN ("20200131"),
    PARTITION p2 VALUES LESS THAN ("20200229"),
    PARTITION p3 VALUES LESS THAN ("20200331")
)
DISTRIBUTED BY HASH(site_id);
```

- **Три специфические временные функции могут использоваться в качестве партиционирующих колонок (поддерживается с версии 3.3.0).**

  При явном определении отображения между партициями и диапазонами значений партиционирующих колонок вы можете использовать специфическую временную функцию для преобразования значений партиционирующих колонок с timestamp'ами

или строками в значения дат, а затем разделить партиции на основе преобразованных значений дат.

Значения партиционирующей колонки являются timestamp'ами | **Значения партиционирующей колонки являются строками**

```
-- 10-значный timestamp с точностью до секунды, например, 1703832553.
CREATE TABLE site_access(
    event_time bigint,
    site_id INT,
    city_code SMALLINT,
    user_name VARCHAR(32),
    pv BIGINT SUM DEFAULT '0'
  )
AGGREGATE KEY(event_time, site_id, city_code, user_name)
PARTITION BY RANGE(from_unixtime(event_time)) (
    PARTITION p1 VALUES LESS THAN ("2021-01-01"),
    PARTITION p2 VALUES LESS THAN ("2021-01-02"),
    PARTITION p3 VALUES LESS THAN ("2021-01-03")
)
DISTRIBUTED BY HASH(site_id)
;

-- 13-значный timestamp с точностью до миллисекунды, например,
1703832553219.
CREATE TABLE site_access(
    event_time bigint,
    site_id INT,
    city_code SMALLINT,
    user_name VARCHAR(32),
    pv BIGINT SUM DEFAULT '0'
  )
AGGREGATE KEY(event_time, site_id, city_code, user_name)
PARTITION BY RANGE(from_unixtime_ms(event_time))(
    PARTITION p1 VALUES LESS THAN ("2021-01-01"),
    PARTITION p2 VALUES LESS THAN ("2021-01-02"),
    PARTITION p3 VALUES LESS THAN ("2021-01-03")
)
DISTRIBUTED BY HASH(site_id);
```

Динамическое партиционирование

Свойства, связанные с динамическим партиционированием, настраиваются при создании таблицы. Selena автоматически создает новые партиции заранее и удаляет устаревшие партиции для обеспечения свежести данных, что реализует управление временем жизни (TTL) для партиций.

В отличие от возможности автоматического создания партиций, предоставляемой expression партиционированием, динамическое партиционирование может только периодически создавать новые партиции на основе свойств. Если новые данные не принадлежат этим партициям, для задания загрузки возвращается ошибка. Однако возможность автоматического создания партиций, предоставляемая expression партиционированием, может всегда создавать соответствующие новые партиции на основе загружаемых данных.

Создание нескольких партиций в пакетном режиме

Несколько партиций могут быть созданы в пакетном режиме при и после создания таблицы. Вы можете указать начальное и конечное время для всех партиций, созданных в пакетном режиме, в `START()` и `END()` и значение приращения партиции в `EVERY()`. Однако обратите внимание, что диапазон партиций является левозамкнутым и правооткрытым, который включает начальное время, но не включает конечное время. Правило именования для партиций такое же, как и для динамического партиционирования.

- **Партиционирующая колонка имеет тип date.**

  Когда партиционирующая колонка имеет тип date, при создании таблицы вы можете использовать `START()` и `END()` для указания начальной и конечной даты для всех партиций, созданных в пакетном режиме, и `EVERY(INTERVAL xxx)` для указания интервала приращения между двумя партициями. В настоящее время гранулярность интервала поддерживает `HOUR` (с версии 3.0), `DAY`, `WEEK`, `MONTH` и `YEAR`.

  с одинаковым интервалом дат | **с разными интервалами дат**

  В следующем примере партиции, созданные в пакетном режиме, начинаются с `2021-01-01` и заканчиваются `2021-01-04`, с приращением партиции в один день:

```
CREATE TABLE site_access (
    datekey DATE,
    site_id INT,
    city_code SMALLINT,
    user_name VARCHAR(32),
    pv BIGINT DEFAULT '0'
)
DUPLICATE KEY(datekey, site_id, city_code, user_name)
PARTITION BY RANGE (datekey) (
    START ("2021-01-01") END ("2021-01-04") EVERY (INTERVAL 1 DAY)
)
DISTRIBUTED BY HASH(site_id);
```

Это эквивалентно использованию следующей клаузы `PARTITION BY` в операторе CREATE TABLE:

```
PARTITION BY RANGE (datekey) (
    PARTITION p20210101 VALUES [('2021-01-01'), ('2021-01-02')),
    PARTITION p20210102 VALUES [('2021-01-02'), ('2021-01-03')),
    PARTITION p20210103 VALUES [('2021-01-03'), ('2021-01-04'))
)
```

- **Партиционирующая колонка имеет целочисленный тип.**

Когда тип данных партиционирующей колонки - INT, вы указываете диапазон партиций в `START` и `END` и определяете значение приращения в `EVERY`. Пример:

> **ПРИМЕЧАНИЕ**
>
> Значения партиционирующих колонок в **START()** и **END()** должны быть заключены в двойные кавычки, в то время как значение приращения в **EVERY()** не нужно заключать в двойные кавычки.

с одинаковым числовым интервалом	**с разными числовыми интервалами**

В следующем примере диапазон всех партиций начинается с `1` и заканчивается `5`, с приращением партиции `1`:

```
CREATE TABLE site_access (
    datekey INT,
    site_id INT,
    city_code SMALLINT,
    user_name VARCHAR(32),
    pv BIGINT DEFAULT '0'
)
DUPLICATE KEY(datekey, site_id, city_code, user_name)
PARTITION BY RANGE (datekey) (
    START ("1") END ("5") EVERY (1)
)
DISTRIBUTED BY HASH(site_id);
```

Это эквивалентно использованию следующей клаузы `PARTITION BY` в операторе CREATE TABLE:

```
PARTITION BY RANGE (datekey) (
    PARTITION p1 VALUES [("1"), ("2")),
    PARTITION p2 VALUES [("2"), ("3")),
    PARTITION p3 VALUES [("3"), ("4")),
    PARTITION p4 VALUES [("4"), ("5"))
)
```

- **Три специфические временные функции могут использоваться в качестве партиционирующих колонок (поддерживается с версии 3.3.0).**

  При явном определении отображения между партициями и диапазонами значений партиционирующих колонок вы можете использовать специфическую временную функцию для преобразования значений партиционирующих колонок с timestamp'ами или строками в значения дат, а затем разделить партиции на основе преобразованных значений дат.

Значения партиционирующей колонки являются timestamp'ами	Значения партиционирующей колонки являются строками

```
-- 10-значный timestamp с точностью до секунды, например, 1703832553.
CREATE TABLE site_access(
    event_time bigint,
    site_id INT,
    city_code SMALLINT,
    user_name VARCHAR(32),
    pv BIGINT DEFAULT '0'
  )
PARTITION BY RANGE(from_unixtime(event_time)) (
    START ("2021-01-01") END ("2021-01-10") EVERY (INTERVAL 1 DAY)
)
DISTRIBUTED BY HASH(site_id);
-- 13-значный timestamp с точностью до миллисекунды, например,
1703832553219.
CREATE TABLE site_access(
    event_time bigint,
    site_id INT,
    city_code SMALLINT,
    user_name VARCHAR(32),
    pv BIGINT DEFAULT '0'
)
PARTITION BY RANGE(from_unixtime_ms(event_time))(
    START ("2021-01-01") END ("2021-01-10") EVERY (INTERVAL 1 DAY)
)
DISTRIBUTED BY HASH(site_id);
```

List партиционирование (с версии 3.1)

List Партиционирование подходит для ускорения запросов и эффективного управления данными на основе перечисляемых значений. Оно особенно полезно для сценариев, где партиция должна включать данные с различными значениями в партиционирующей колонке. Например, если вы часто запрашиваете и управляете данными на основе стран и городов, вы можете использовать этот метод партиционирования и выбрать колонку `city` в качестве партиционирующей колонки. В этом случае одна партиция может содержать данные для различных городов, принадлежащих одной стране.

Selena хранит данные в соответствующих партициях на основе явного сопоставления предопределенного списка значений для каждой партиции.

Управление партициями

Добавление партиций

Для range партиционирования и list партиционирования вы можете вручную добавить новые партиции для хранения новых данных. Однако для expression партиционирования, поскольку партиции создаются автоматически во время загрузки данных, вам не нужно этого делать.

Следующий оператор добавляет новую партицию в таблицу `site_access` для хранения данных за новый месяц:

```
ALTER TABLE site_access
ADD PARTITION p4 VALUES LESS THAN ("2020-04-30")
DISTRIBUTED BY HASH(site_id);
```

Удаление партиции

Следующий оператор удаляет партицию `p1` из таблицы `site_access`.

> **ПРИМЕЧАНИЕ**
>
> Эта операция не удаляет данные в партиции немедленно. Данные сохраняются в Корзине в течение определенного периода времени (один день по умолчанию). Если партиция была ошибочно удалена, вы можете использовать команду RECOVER для восстановления партиции и ее данных.

```
ALTER TABLE site_access
DROP PARTITION p1;
```

Восстановление партиции

Следующий оператор восстанавливает партицию `p1` и ее данные в таблицу `site_access`.

```
RECOVER PARTITION p1 FROM site_access;
```

Просмотр партиций

Следующий оператор возвращает подробную информацию обо всех партициях в таблице `site_access`.

```
SHOW PARTITIONS FROM site_access;
```

2.4.3 Настройка bucketing'a

Random bucketing (с версии 3.1)

Selena распределяет данные в партиции случайным образом по всем bucket'ам. Подходит для сценариев с небольшими объемами данных и относительно низкими требованиями к производительности запросов. Если вы не устанавливаете метод bucketing'a, Selena использует random bucketing по умолчанию и автоматически устанавливает количество bucket'ов.

Однако обратите внимание, что если вы запрашиваете большие объемы данных и часто используете определенные колонки в качестве условий фильтрации, производительность запросов, обеспечиваемая random bucketing, может быть не оптимальной. В таких сценариях рекомендуется использовать hash bucketing. Когда эти колонки используются в качестве условий фильтрации для запросов, необходимо сканировать и вычислять только данные в небольшом количестве bucket'ов, которые попадают в запрос, что может значительно улучшить производительность запросов.

Ограничения

- Вы можете использовать random bucketing только для создания таблицы Duplicate Key.
- Вы не можете указать таблицу с random bucketing для принадлежности к Colocation Group.
- Spark Load не может использоваться для загрузки данных в таблицы с random bucketing.

В следующем примере CREATE TABLE оператор `DISTRIBUTED BY xxx` не используется, поэтому Selena использует random bucketing по умолчанию и автоматически устанавливает количество bucket'ов.

```
CREATE TABLE site_access1(
    event_day DATE,
    site_id INT DEFAULT '10',
    pv BIGINT DEFAULT '0' ,
    city_code VARCHAR(100),
    user_name VARCHAR(32) DEFAULT ''
)
DUPLICATE KEY(event_day,site_id,pv);
```

Однако, если вы знакомы с механизмом bucketing'a Selena, вы также можете вручную установить количество bucket'ов при создании таблицы с random bucketing.

```
CREATE TABLE site_access2(
    event_day DATE,
    site_id INT DEFAULT '10',
    pv BIGINT DEFAULT '0' ,
    city_code VARCHAR(100),
    user_name VARCHAR(32) DEFAULT ''
)
DUPLICATE KEY(event_day,site_id,pv)
DISTRIBUTED BY RANDOM BUCKETS 8; -- вручную установить количество bucket'ов
на 8
```

Hash bucketing

Selena может использовать hash bucketing для подразделения данных в партиции на bucket'ы на основе bucketing key и количества bucket'ов. При hash bucketing хеш-функция принимает значение bucketing key данных в качестве входных данных и вычисляет хеш-значение. Данные хранятся в соответствующем bucket'е на основе сопоставления между хеш-значениями и bucket'ами.

Преимущества

- Улучшенная производительность запросов: Строки с одинаковыми значениями bucketing key хранятся в одном bucket'е, что уменьшает объем данных, сканируемых во время запросов.

- Равномерное распределение данных: Выбирая колонки с более высокой кардинальностью (большее количество уникальных значений) в качестве bucketing key, данные могут быть более равномерно распределены по bucket'ам.

Как выбрать bucketing колонки

Мы рекомендуем выбирать колонку, которая удовлетворяет следующим двум требованиям, в качестве bucketing колонки.

- колонка с высокой кардинальностью, такая как ID
- колонка, которая часто используется в фильтре для запросов

Но если ни одна колонка не удовлетворяет обоим требованиям, вам нужно определить bucketing колонку в соответствии со сложностью запросов.

- Если запрос сложный, рекомендуется выбрать колонки с высокой кардинальностью в качестве bucketing колонок, чтобы обеспечить равномерное распределение данных по всем bucket'ам и улучшить использование ресурсов кластера.
- Если запрос относительно простой, рекомендуется выбрать колонки, которые часто используются в качестве условий фильтрации в запросах, в качестве bucketing колонок для повышения эффективности запросов.

Если данные партиции не могут быть равномерно распределены по всем bucket'ам с использованием одной bucketing колонки, вы можете выбрать несколько bucketing колонок. Обратите внимание, что рекомендуется использовать не более 3 колонок.

Меры предосторожности

- **При создании таблицы необходимо указать bucketing колонки**.
- Типы данных bucketing колонок должны быть INTEGER, DECIMAL, DATE/DATETIME или CHAR/VARCHAR/STRING.
- С версии 3.2 bucketing колонки могут быть изменены с помощью ALTER TABLE после создания таблицы.

Примеры

В следующем примере таблица `site_access` создается с использованием `site_id` в качестве bucketing колонки. Кроме того, когда данные в таблице `site_access` запрашиваются, данные часто фильтруются по сайтам. Использование `site_id` в качестве bucketing key может исключить значительное количество нерелевантных bucket'ов во время запросов.

```
CREATE TABLE site_access(
    event_day DATE,
    site_id INT DEFAULT '10',
    city_code VARCHAR(100),
    user_name VARCHAR(32) DEFAULT '',
    pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(event_day, site_id, city_code, user_name)
PARTITION BY RANGE(event_day)
(
    PARTITION p1 VALUES LESS THAN ("2020-01-31"),
    PARTITION p2 VALUES LESS THAN ("2020-02-29"),
    PARTITION p3 VALUES LESS THAN ("2020-03-31")
)
DISTRIBUTED BY HASH(site_id);
```

Предположим, что каждая партиция таблицы `site_access` имеет 10 bucket'ов. В следующем запросе 9 из 10 bucket'ов исключаются, поэтому Selena нужно сканировать только 1/10 данных в таблице `site_access`:

```
select sum(pv)
from site_access
where site_id = 54321;
```

Однако, если `site_id` распределен неравномерно и большое количество запросов запрашивает данные только нескольких сайтов, использование только одной bucketing колонки может привести к серьезному перекосу данных, вызывая узкие места в производительности системы. В этом случае вы можете использовать комбинацию bucketing колонок. Например, следующий оператор использует `site_id` и `city_code` в качестве bucketing колонок.

```
CREATE TABLE site_access
(
    site_id INT DEFAULT '10',
    city_code SMALLINT,
    user_name VARCHAR(32) DEFAULT '',
    pv BIGINT SUM DEFAULT '0'
)
AGGREGATE KEY(site_id, city_code, user_name)
DISTRIBUTED BY HASH(site_id,city_code);
```

С практической точки зрения, вы можете использовать одну или две bucketing колонки в зависимости от характеристик вашего бизнеса. Использование одной bucketing колонки `site_id` очень выгодно для коротких запросов, поскольку оно уменьшает обмен данными между узлами, тем самым повышая общую производительность кластера. С другой стороны, принятие двух bucketing колонок `site_id` и `city_code` выгодно для длинных запросов, поскольку оно может использовать общую параллельность распределенного кластера для значительного повышения производительности.

> **ПРИМЕЧАНИЕ**
>
> - Короткие запросы включают сканирование небольшого объема данных и могут быть завершены на одном узле.
> - Длинные запросы включают сканирование большого объема данных, и их производительность может быть значительно улучшена за счет параллельного сканирования на нескольких узлах в распределенном кластере.

Установка количества bucket'ов

Bucket'ы отражают, как файлы данных фактически организованы в Selena.

При создании таблицы

- Автоматическая установка количества bucket'ов (рекомендуется)

  Начиная с версии 2.5.7, Selena поддерживает автоматическую установку количества bucket'ов на основе ресурсов машины и объема данных для партиции.

> **ПОДСКАЗКА**
>
> Если размер необработанных данных партиции превышает 100 ГБ, мы рекомендуем вручную настроить количество bucket'ов с помощью Метода 2.

Таблица настроена с hash bucketing	**Таблица настроена с random bucketing**

Пример:

```
CREATE TABLE site_access (
    site_id INT DEFAULT '10',
    city_code SMALLINT,
    user_name VARCHAR(32) DEFAULT '',
    event_day DATE,
    pv BIGINT SUM DEFAULT '0')
AGGREGATE KEY(site_id, city_code, user_name,event_day)
PARTITION BY date_trunc('day', event_day)
DISTRIBUTED BY HASH(site_id,city_code); -- не нужно устанавливать
количество bucket'ов
```

- Ручная установка количества bucket'ов

Начиная с версии 2.4.0, Selena поддерживает использование нескольких потоков для сканирования tablet параллельно во время запроса, тем самым уменьшая зависимость производительности сканирования от количества tablet'ов. Мы рекомендуем, чтобы каждый tablet содержал около 10 ГБ необработанных данных. Если вы намереваетесь вручную установить количество bucket'ов, вы можете оценить объем данных в каждой партиции таблицы, а затем решить количество tablet'ов.

Чтобы включить параллельное сканирование на tablet'ах, убедитесь, что параметр `enable_tablet_internal_parallel` установлен в `TRUE` глобально для всей системы (`SET GLOBAL enable_tablet_internal_parallel = true;`).

Таблица настроена с hash bucketing	**Таблица настроена с random bucketing**

```sql
CREATE TABLE site_access (
    site_id INT DEFAULT '10',
    city_code SMALLINT,
    user_name VARCHAR(32) DEFAULT '',
    event_day DATE,
    pv BIGINT SUM DEFAULT '0')
AGGREGATE KEY(site_id, city_code, user_name,event_day)
PARTITION BY date_trunc('day', event_day)
DISTRIBUTED BY HASH(site_id,city_code) BUCKETS 30;
-- Предположим, что объем необработанных данных, которые вы хотите загрузить в партицию, составляет 300 ГБ.
-- Поскольку мы рекомендуем, чтобы каждый tablet содержал 10 ГБ необработанных данных, количество bucket'ов может быть установлено на 30.
DISTRIBUTED BY HASH(site_id,city_code) BUCKETS 30;
```

После создания таблицы

- Автоматическая установка количества bucket'ов (рекомендуется)

  Начиная с версии 2.5.7, Selena поддерживает автоматическую установку количества bucket'ов на основе ресурсов машины и объема данных для партиции.

  > **ПОДСКАЗКА**
  >
  > Если размер необработанных данных партиции превышает 100 ГБ, мы рекомендуем вручную настроить количество bucket'ов с помощью Метода 2.

Таблица настроена с hash bucketing	Таблица настроена с random bucketing

```
-- Автоматически установить количество bucket'ов для всех партиций.
ALTER TABLE site_access DISTRIBUTED BY HASH(site_id,city_code);

-- Автоматически установить количество bucket'ов для конкретных
партиций.
ALTER TABLE site_access PARTITIONS (p20230101, p20230102)
DISTRIBUTED BY HASH(site_id,city_code);

-- Автоматически установить количество bucket'ов для новых партиций.
ALTER TABLE site_access ADD PARTITION p20230106 VALUES [('2023-01-06'),
('2023-01-07'))
DISTRIBUTED BY HASH(site_id,city_code);
```

- Ручная установка количества bucket'ов

  Вы также можете вручную указать количество bucket'ов. Чтобы рассчитать количество bucket'ов для партиции, вы можете обратиться к подходу, используемому при ручной установке количества bucket'ов при создании таблицы, как упоминалось выше.

Таблица настроена с hash bucketing	**Таблица настроена с random bucketing**

```
-- Вручную установить количество bucket'ов для всех партиций
ALTER TABLE site_access
DISTRIBUTED BY HASH(site_id,city_code) BUCKETS 30;
-- Вручную установить количество bucket'ов для конкретных партиций.
ALTER TABLE site_access
partitions p20230104
DISTRIBUTED BY HASH(site_id,city_code)  BUCKETS 30;
-- Вручную установить количество bucket'ов для новых партиций.
ALTER TABLE site_access
ADD PARTITION p20230106 VALUES [('2023-01-06'), ('2023-01-07'))
DISTRIBUTED BY HASH(site_id,city_code) BUCKETS 30;
```

Просмотр количества bucket'ов

После создания таблицы вы можете выполнить SHOW PARTITIONS, чтобы просмотреть количество bucket'ов, установленное Selena для каждой партиции. Таблицы, настроенные с hash bucketing, имеют фиксированное количество bucket'ов на партицию.

К СВЕДЕНИЮ

- Что касается таблицы, настроенной с random bucketing, которая включает увеличение количества bucket'ов по требованию и динамически, количество bucket'ов в каждой партиции динамически увеличивается. Таким образом, возвращенный результат отображает текущее количество bucket'ов для каждой партиции.
- Для этого типа таблиц фактическая иерархия внутри партиции выглядит следующим образом: партиция > субпартиция > bucket. Чтобы увеличить количество bucket'ов, Selena фактически добавляет новую субпартицию, которая включает определенное количество bucket'ов. В результате оператор SHOW PARTITIONS может вернуть несколько строк данных с одинаковым именем партиции, которые показывают информацию о субпартициях внутри одной и той же партиции.

2.4.4 Оптимизация распределения данных после создания таблицы (с версии 3.2)

> **ВНИМАНИЕ**
>
> Режим shared-data Selena в настоящее время не поддерживает эту функцию.

По мере развития паттернов запросов и объема данных в бизнес-сценариях, конфигурации, указанные при создании таблицы, такие как метод bucketing'a, количество bucket'ов и ключ сортировки, могут больше не подходить для нового бизнес-сценария и даже могут привести к снижению производительности запросов. В этот момент вы можете использовать `ALTER TABLE` для изменения метода bucketing'a, количества bucket'ов и ключа сортировки для оптимизации распределения данных. Например:

- **Увеличьте количество bucket'ов, когда объем данных внутри партиций значительно увеличился**

  Когда объем данных внутри партиций становится значительно больше, чем раньше, необходимо изменить количество bucket'ов, чтобы поддерживать размеры tablet в целом в диапазоне от 1 ГБ до 10 ГБ.

- **Измените bucketing key, чтобы избежать перекоса данных**

Когда текущий bucketing key может вызвать перекос данных (например, только колонка `k1` настроена в качестве bucketing key), необходимо указать более подходящие колонки или добавить дополнительные колонки к bucketing key. Например:

```
ALTER TABLE t DISTRIBUTED BY HASH(k1, k2) BUCKETS 20;
-- Когда версия Selena 3.1 или позже, и таблица является таблицей
Duplicate Key, вы можете рассмотреть возможность прямого использования
настроек bucketing'а системы по умолчанию, то есть random bucketing и
количество bucket'ов, автоматически установленное Selena.
ALTER TABLE t DISTRIBUTED BY RANDOM;
```

- **Адаптация ключа сортировки из-за изменений в паттернах запросов**

  Если бизнес-паттерны запросов значительно изменились и дополнительные колонки используются в качестве условных колонок, может быть выгодно настроить ключ сортировки. Например:

```
ALTER TABLE t ORDER BY k2, k1;
```

Для получения дополнительной информации см. ALTER TABLE.

2.4.5 Expression partitioning (рекомендуется)

Начиная с версии v1.5.2, Selena поддерживает expression partitioning (ранее известное как автоматическое партиционирование), которое является более гибким и удобным для пользователя. Этот метод партиционирования подходит для большинства сценариев, таких как запрос и управление данными на основе непрерывных временных диапазонов или значений ENUM.

Вам нужно лишь указать простое выражение partition при создании таблицы. Во время загрузки данных Selena будет автоматически создавать partitions на основе данных и правила, определенного в выражении partition. Вам больше не нужно вручную создавать множество partitions при создании таблицы или настраивать свойства динамических partitions.

Начиная с версии v1.5.2, expression partitioning дополнительно оптимизирован для унификации всех стратегий партиционирования и поддержки более сложных решений. Он

рекомендуется в большинстве случаев и заменит другие стратегии партиционирования в будущих релизах.

Начиная с версии v1.5.2, Selena поддерживает слияние expression partitions на основе временных функций для оптимизации эффективности хранения и производительности запросов. Подробную информацию см. в разделе Слияние expression partitions.

Партиционирование на основе простого выражения временной функции

Если вы часто запрашиваете и управляете данными на основе непрерывных временных диапазонов, вам нужно лишь указать столбец типа date (DATE или DATETIME) в качестве столбца partition и указать год, месяц, день или час в качестве гранулярности partition в выражении временной функции. Selena автоматически создаст partitions и установит даты начала и окончания partitions или datetime на основе загруженных данных и выражения partition.

Однако в некоторых особых сценариях, таких как партиционирование исторических данных в partitions по месяцам, а недавних данных в partitions по дням, вы должны использовать range partitioning для создания partitions.

> **ПРИМЕЧАНИЕ**
>
> Обратите внимание, что `PARTITION BY date_trunc(column)` и `PARTITION BY time_slice(column)` считаются range partitioning, несмотря на их формат expression partitioning. Поэтому вы можете использовать оператор `ALTER TABLE ... ADD PARTITION` для range partitions, чтобы добавлять новые partitions к таблицам, использующим такие стратегии партиционирования.

Синтаксис

```
PARTITION BY expression
...
[ PROPERTIES( { 'partition_live_number' = 'xxx' |
'partition_retention_condition' = 'expr' } ) ]

expression ::=
    { date_trunc ( <time_unit> , <partition_column> ) |
      time_slice ( <partition_column> , INTERVAL <N> <time_unit> [ ,
boundary ] ) }
```

Параметры

`expression`

Обязательный: ДА
Описание: Простое выражение временной функции, использующее функции date_trunc или time_slice. Если вы используете функцию `time_slice`, вам не нужно передавать параметр `boundary`. Это связано с тем, что в этом сценарии значением по умолчанию и допустимым значением для этого параметра является `floor`, и значение не может быть `ceil`.

`time_unit`

Обязательный: ДА
Описание: Гранулярность partition, которая может быть `hour`, `day`, `month` или `year`. Гранулярность partition `week` не поддерживается. Если гранулярность partition - `hour`, столбец partition должен иметь тип данных DATETIME и не может быть типа DATE.

`partition_column`

Обязательный: ДА
Описание: Имя столбца partition.

- Столбец partition может быть только типа DATE или DATETIME. Столбец partition допускает значения `NULL`.
- Столбец partition может быть типа DATE или DATETIME, если используется функция `date_trunc`. Столбец partition должен быть типа DATETIME, если используется функция `time_slice`.

- Если столбец partition имеет тип DATE, поддерживаемый диапазон - [0000-01-01 ~ 9999-12-31]. Если столбец partition имеет тип DATETIME, поддерживаемый диапазон - [0000-01-01 01:01:01 ~ 9999-12-31 23:59:59].
- В настоящее время вы можете указать только один столбец partition; множественные столбцы partition не поддерживаются.

`partition_live_number`

Обязательный: НЕТ
Описание: Количество последних partitions, которые необходимо сохранить. Partitions сортируются в хронологическом порядке, **с текущей датой в качестве эталона**; partitions старше текущей даты минус `partition_live_number` удаляются. Selena планирует задачи для управления количеством partitions, и интервал планирования может быть настроен через динамический параметр FE `dynamic_partition_check_interval_seconds`, который по умолчанию составляет 600 секунд (10 минут). Предположим, что текущая дата - 4 апреля 2023 года, `partition_live_number` установлен в `2`, и partitions включают `p20230401`, `p20230402`, `p20230403`, `p20230404`. Partitions `p20230403` и `p20230404` сохраняются, а другие partitions удаляются. Если загружены грязные данные, такие как данные с будущих дат 5 и 6 апреля, partitions включают `p20230401`, `p20230402`, `p20230403`, `p20230404`, `p20230405` и `p20230406`. Тогда partitions `p20230403`, `p20230404`, `p20230405` и `p20230406` сохраняются, а другие partitions удаляются.

`partition_retention_condition`

Начиная с версии v1.5.2, нативные таблицы Selena поддерживают Common Partition Expression TTL.

`partition_retention_condition`: Выражение, которое объявляет partitions, которые должны быть динамически сохранены. Partitions, которые не соответствуют условию в выражении, будут регулярно удаляться. Пример: `'partition_retention_condition' = 'dt >= CURRENT_DATE() - INTERVAL 3 MONTH'`.

- Выражение может содержать только столбцы partition и константы. Столбцы, не являющиеся столбцами partition, не поддерживаются.
- Common Partition Expression применяется к List partitions и Range partitions по-разному:

- Для таблиц с List partitions Selena поддерживает удаление partitions, отфильтрованных Common Partition Expression.
- Для таблиц с Range partitions Selena может фильтровать и удалять partitions только с использованием возможности partition pruning FE. Partitions, соответствующие предикатам, которые не поддерживаются partition pruning, не могут быть отфильтрованы и удалены.

Примечания по использованию

- Во время загрузки данных Selena автоматически создает некоторые partitions на основе загруженных данных, но если задание загрузки завершается неудачей по какой-либо причине, partitions, автоматически созданные Selena, не могут быть автоматически удалены.
- Selena устанавливает максимальное количество автоматически создаваемых partitions для одной загрузки по умолчанию в 4096, что можно настроить через параметр FE `auto_partition_max_creation_number_per_load`. Этот параметр может предотвратить случайное создание слишком большого количества partitions.
- Правило именования для partitions согласовано с правилом именования для динамического партиционирования.

Примеры

Пример 1: Предположим, вы часто запрашиваете данные по дням. Вы можете использовать выражение partition `date_trunc()` и установить столбец partition как `event_day`, а гранулярность partition как `day` при создании таблицы. Данные автоматически партиционируются на основе дат во время загрузки. Данные одного дня хранятся в одном partition, и partition pruning может быть использован для значительного повышения эффективности запросов.

```
CREATE TABLE site_access1 (
    event_day DATETIME NOT NULL,
    site_id INT DEFAULT '10',
    city_code VARCHAR(100),
    user_name VARCHAR(32) DEFAULT '',
    pv BIGINT DEFAULT '0'
)
DUPLICATE KEY(event_day, site_id, city_code, user_name)
PARTITION BY date_trunc('day', event_day)
DISTRIBUTED BY HASH(event_day, site_id);
```

Например, когда загружаются следующие две строки данных, Selena автоматически создаст два partitions, `p20230226` и `p20230227`, с диапазонами [2023-02-26 00:00:00, 2023-02-27 00:00:00) и [2023-02-27 00:00:00, 2023-02-28 00:00:00) соответственно. Если последующие загруженные данные попадают в эти диапазоны, они автоматически направляются в соответствующие partitions.

```
-- вставка двух строк данных
INSERT INTO site_access1
    VALUES ("2023-02-26 20:12:04",002,"New York","Sam Smith",1),
           ("2023-02-27 21:06:54",001,"Los Angeles","Taylor Swift",1);

-- просмотр partitions
mysql > SHOW PARTITIONS FROM site_access1;
+-------------+---------------+----------------+---------------------+-----
---------------+--------+--------------+-----------------------------------
-----------------------------------------------------------------+-------
-------------+---------+----------------+---------------+------------------
---+---------------------------+----------+------------+----------+
| PartitionId | PartitionName | VisibleVersion | VisibleVersionTime  |
VisibleVersionHash | State  | PartitionKey | Range
| DistributionKey    | Buckets | ReplicationNum | StorageMedium |
CooldownTime        | LastConsistencyCheckTime | DataSize | IsInMemory |
RowCount |
+-------------+---------------+----------------+---------------------+-----
---------------+--------+--------------+-----------------------------------
-----------------------------------------------------------------+-------
-------------+---------+----------------+---------------+------------------
---+---------------------------+----------+------------+----------+
| 17138       | p20230226     | 2              | 2023-07-19 17:53:59 | 0
| NORMAL | event_day    | [types: [DATETIME]; keys: [2023-02-26 00:00:00];
..types: [DATETIME]; keys: [2023-02-27 00:00:00]; ) | event_day, site_id |
6       | 3              | HDD           | 9999-12-31 23:59:59 | NULL
| 0B       | false      | 0        |
| 17113       | p20230227     | 2              | 2023-07-19 17:53:59 | 0
| NORMAL | event_day    | [types: [DATETIME]; keys: [2023-02-27 00:00:00];
..types: [DATETIME]; keys: [2023-02-28 00:00:00]; ) | event_day, site_id |
6       | 3              | HDD           | 9999-12-31 23:59:59 | NULL
| 0B       | false      | 0        |
+-------------+---------------+----------------+---------------------+-----
---------------+--------+--------------+-----------------------------------
-----------------------------------------------------------------+-------
-------------+---------+----------------+---------------+------------------
---+---------------------------+----------+------------+----------+
2 rows in set (0.00 sec)
```

Пример 2: Если вы хотите реализовать управление жизненным циклом partitions, то есть сохранять только определенное количество недавних partitions и удалять исторические partitions, вы можете использовать свойство `partition_live_number` для указания количества partitions, которые необходимо сохранить.

```
CREATE TABLE site_access2 (
    event_day DATETIME NOT NULL,
    site_id INT DEFAULT '10',
    city_code VARCHAR(100),
    user_name VARCHAR(32) DEFAULT '',
    pv BIGINT DEFAULT '0'
)
DUPLICATE KEY(event_day, site_id, city_code, user_name)
PARTITION BY date_trunc('month', event_day)
DISTRIBUTED BY HASH(event_day, site_id)
PROPERTIES(
    "partition_live_number" = "3" -- сохраняет только последние три
partitions
);
```

Пример 3: Предположим, вы часто запрашиваете данные по неделям. Вы можете использовать выражение partition `time_slice()` и установить столбец partition как `event_day`, а гранулярность partition как семь дней при создании таблицы. Данные одной недели хранятся в одном partition, и partition pruning может быть использован для значительного повышения эффективности запросов.

```
CREATE TABLE site_access(
    event_day DATETIME NOT NULL,
    site_id INT DEFAULT '10',
    city_code VARCHAR(100),
    user_name VARCHAR(32) DEFAULT '',
    pv BIGINT DEFAULT '0'
)
DUPLICATE KEY(event_day, site_id, city_code, user_name)
PARTITION BY time_slice(event_day, INTERVAL 7 day)
DISTRIBUTED BY HASH(event_day, site_id)
```

Партиционирование на основе выражения столбца (начиная с v1.5.2)

Если вы часто запрашиваете и управляете данными определенного типа, вам нужно лишь указать столбец, представляющий тип, в качестве столбца partition. Selena автоматически создаст partitions на основе значений столбца partition загруженных данных.

Однако в некоторых особых сценариях, например, когда таблица содержит столбец `city`, и вы часто запрашиваете и управляете данными на основе стран и городов, вы должны

использовать list partitioning для хранения данных нескольких городов в одной стране в одном partition.

Синтаксис

```
PARTITION BY expression
...

expression ::=
    partition_columns

partition_columns ::=
    <column>, [ <column> [,...] ]
```

Параметры

`partition_columns`

Обязательный: ДА
Описание: Имена столбцов partition.

- Значения столбцов partition могут быть строковыми (BINARY не поддерживается), date или datetime, целочисленными и логическими значениями. Столбец partition допускает значения `NULL`.
- Каждый partition может содержать только данные с одинаковым значением в столбце partition. Чтобы включить данные с различными значениями в столбце partition в partition, см. List partitioning.

> **ПРИМЕЧАНИЕ**
> Начиная с версии v1.5.2, вы можете опустить скобки, которые используются для обертывания столбцов partition. Например, вы можете заменить `PARTITION BY (dt,city)` на `PARTITION BY dt,city`.

Примечания по использованию

- Во время загрузки данных Selena автоматически создает некоторые partitions на основе загруженных данных, но если задание загрузки завершается неудачей по

какой-либо причине, partitions, автоматически созданные Selena, не могут быть автоматически удалены.

- Selena устанавливает максимальное количество автоматически создаваемых partitions для одной загрузки по умолчанию в 4096, что можно настроить через параметр FE `auto_partition_max_creation_number_per_load`. Этот параметр может предотвратить случайное создание слишком большого количества partitions.
- Правило именования для partitions: если указано несколько столбцов partition, значения различных столбцов partition соединяются подчеркиванием `_` в имени partition, и формат - `p<значение в столбце partition 1>_<значение в столбце partition 2>_...`. Например, если два столбца `dt` и `province` указаны как столбцы partition, оба из которых имеют строковый тип, и загружается строка данных со значениями `2022-04-01` и `beijing`, соответствующий автоматически созданный partition именуется `p20220401_beijing`.

Примеры

Пример 1: Предположим, вы часто запрашиваете детали биллинга центра обработки данных на основе временных диапазонов и конкретных городов. При создании таблицы вы можете использовать выражение partition для указания первых столбцов partition как `dt` и `city`. Таким образом, данные, относящиеся к одной и той же дате и городу, направляются в один и тот же partition, и partition pruning может быть использован для значительного повышения эффективности запросов.

```
CREATE TABLE t_recharge_detail1 (
    id bigint,
    user_id bigint,
    recharge_money decimal(32,2),
    city varchar(20) not null,
    dt varchar(20) not null
)
DUPLICATE KEY(id)
PARTITION BY dt,city
DISTRIBUTED BY HASH(`id`);
```

Вставьте одну строку данных в таблицу.

```
INSERT INTO t_recharge_detail1
    VALUES (1, 1, 1, 'Houston', '2022-04-01');
```

Просмотрите partitions. Результат показывает, что Selena автоматически создает partition `p20220401_Houston1` на основе загруженных данных. Во время последующей загрузки данные со значениями `2022-04-01` и `Houston` в столбцах partition `dt` и `city` хранятся в этом partition.

> **ПОДСКАЗКА**
>
> Каждый partition может содержать только данные с указанным одним значением для столбца partition. Чтобы указать несколько значений для столбца partition в partition, см. List partitions.

```
MySQL > SHOW PARTITIONS from t_recharge_detail1\G
*************************** 1. row ***************************
             PartitionId: 16890
           PartitionName: p20220401_Houston
          VisibleVersion: 2
      VisibleVersionTime: 2023-07-19 17:24:53
      VisibleVersionHash: 0
                   State: NORMAL
            PartitionKey: dt, city
                    List: (('2022-04-01', 'Houston'))
         DistributionKey: id
                 Buckets: 6
          ReplicationNum: 3
           StorageMedium: HDD
            CooldownTime: 9999-12-31 23:59:59
LastConsistencyCheckTime: NULL
                DataSize: 2.5KB
              IsInMemory: false
                RowCount: 1
1 row in set (0.00 sec)
```

Партиционирование на основе сложного выражения временной функции (начиная с v1.5.2)

Начиная с версии v1.5.2, expression partitioning поддерживает любые выражения, которые возвращают типы DATE или DATETIME, чтобы адаптироваться к еще более сложным сценариям партиционирования. Для поддерживаемых временных функций см. Приложение - Поддерживаемые временные функции.

Например, вы можете определить столбец Unix timestamp и использовать from_unixtime() непосредственно для столбца в выражении partition для определения ключа partition вместо определения сгенерированного столбца DATE или DATETIME с функцией. Для получения дополнительной информации об использовании см. Примеры.

Начиная с версии v1.5.2, partition pruning поддерживается для partitions на основе большинства функций, связанных с DATETIME.

Примеры

Пример 1: Предположим, вы назначаете Unix timestamp каждой строке данных и часто запрашиваете данные по дням. Вы можете использовать столбец timestamp с функцией from_unixtime() в выражении для определения timestamp в качестве столбца partition, а гранулярность partition - день при создании таблицы. Данные каждого дня хранятся в одном partition, и partition pruning может быть использован для улучшения эффективности запросов.

```
CREATE TABLE orders (
    ts BIGINT NOT NULL,
    id BIGINT NOT NULL,
    city STRING NOT NULL
)
PARTITION BY from_unixtime(ts,'%Y%m%d');
```

Пример 2: Предположим, вы назначаете INT тип timestamp каждой строке данных и храните данные ежемесячно. Вы можете использовать столбец timestamp с функциями cast() и str_to_date() в выражении для преобразования timestamp в тип DATE, установить его в качестве столбца partition, а гранулярность partition - месяц, используя date_trunc() при создании таблицы. Данные каждого месяца хранятся в одном partition, и partition pruning может быть использован для улучшения эффективности запросов.

```
CREATE TABLE orders_new (
    ts INT NOT NULL,
    id BIGINT NOT NULL,
    city STRING NOT NULL
)
PARTITION BY date_trunc('month', str_to_date(CAST(ts as STRING),'%Y%m%d'));
```

Примечания по использованию

Partition pruning применим к случаям партиционирования на основе сложного выражения временной функции:

- Если предложение partition - `PARTITION BY from_unixtime(ts)`, запросы с фильтрами в формате `ts>1727224687` могут быть ограничены до соответствующих partitions.
- Если предложение partition - `PARTITION BY str2date(CAST(ts AS string),'%Y%m')`, запросы с фильтрами в формате `ts = "20240506"` могут быть ограничены.
- Вышеуказанные случаи также применимы к Партиционированию на основе смешанного выражения.

Партиционирование на основе смешанного выражения (начиная с v1.5.2)

Начиная с версии v1.5.2, expression partitioning поддерживает несколько столбцов partition, один из которых является выражением временной функции.

Примеры

Пример 1: Предположим, вы назначаете Unix timestamp каждой строке данных и часто запрашиваете данные по дням и конкретному городу. Вы можете использовать столбец timestamp (с функцией from_unixtime()) и столбец city в качестве столбцов partition при создании таблицы. Данные каждого дня в каждом городе хранятся в одном partition, и partition pruning может быть использован для улучшения эффективности запросов.

```
CREATE TABLE orders (
    ts BIGINT NOT NULL,
    id BIGINT NOT NULL,
    city STRING NOT NULL
)
PARTITION BY from_unixtime(ts,'%Y%m%d'), city;
```

Управление partitions

Загрузка данных в partitions

Во время загрузки данных Selena автоматически создаст partitions на основе загруженных данных и правила partition, определенного выражением partition.

Обратите внимание, что если вы используете expression partitioning при создании таблицы и вам необходимо использовать INSERT OVERWRITE для перезаписи данных в конкретном partition, независимо от того, создан ли partition или нет, в настоящее время вам необходимо явно указать диапазон partition в `PARTITION()`. Это отличается от Range Partitioning или List Partitioning, которые позволяют вам указывать только имя partition в `PARTITION (<partition_name>)`.

Если вы используете выражение временной функции при создании таблицы и хотите перезаписать данные в конкретном partition, вам необходимо указать начальную дату или datetime этого partition (гранулярность partition, настроенная при создании таблицы). Если partition не существует, он может быть автоматически создан во время загрузки данных.

```
INSERT OVERWRITE site_access1 PARTITION(event_day='2022-06-08 20:12:04')
    SELECT * FROM site_access2 PARTITION(p20220608);
```

Если вы используете выражение столбца при создании таблицы и хотите перезаписать данные в конкретном partition, вам необходимо указать значения столбца partition, которые содержит partition. Если partition не существует, он может быть автоматически создан во время загрузки данных.

```
INSERT OVERWRITE t_recharge_detail1 PARTITION(dt='2022-04-02',city='texas')
    SELECT * FROM t_recharge_detail2 PARTITION(p20220402_texas);
```

Просмотр partitions

Когда вы хотите просмотреть конкретную информацию об автоматически созданных partitions, вам необходимо использовать оператор `SHOW PARTITIONS FROM <table_name>`. Оператор `SHOW CREATE TABLE <table_name>` возвращает только синтаксис для expression partitioning, настроенного при создании таблицы.

```
MySQL > SHOW PARTITIONS FROM t_recharge_detail1;
+-------------+-------------------+----------------+---------------------+-
-------------------+--------+--------------+----------------------------+-
----------------+---------+----------------+---------------+---------------
------+--------------------------+----------+------------+----------+
| PartitionId | PartitionName     | VisibleVersion | VisibleVersionTime  |
VisibleVersionHash | State  | PartitionKey | List                        |
DistributionKey | Buckets | ReplicationNum | StorageMedium | CooldownTime
| LastConsistencyCheckTime | DataSize | IsInMemory | RowCount |
+-------------+-------------------+----------------+---------------------+-
-------------------+--------+--------------+----------------------------+-
----------------+---------+----------------+---------------+---------------
------+--------------------------+----------+------------+----------+
| 16890       | p20220401_Houston | 2              | 2023-07-19 17:24:53 |
0                  | NORMAL | dt, city     | (('2022-04-01', 'Houston')) |
id              | 6       | 3              | HDD           | 9999-12-31
23:59:59 | NULL                     | 2.5KB    | false      | 1        |
| 17056       | p20220402_texas   | 2              | 2023-07-19 17:27:42 |
0                  | NORMAL | dt, city     | (('2022-04-02', 'texas'))   |
id              | 6       | 3              | HDD           | 9999-12-31
23:59:59 | NULL                     | 2.5KB    | false      | 1        |
+-------------+-------------------+----------------+---------------------+-
-------------------+--------+--------------+----------------------------+-
----------------+---------+----------------+---------------+---------------
------+--------------------------+----------+------------+----------+
2 rows in set (0.00 sec)
```

Слияние expression partitions

В управлении данными партиционирование на основе различной временной гранулярности имеет решающее значение для оптимизации запросов и хранения. Для повышения эффективности хранения и производительности запросов Selena поддерживает слияние нескольких expression partitions более тонкой временной гранулярности в один partition более грубой временной гранулярности, например, слияние partitions по дням в один partition по месяцам. Путем слияния partitions, которые соответствуют указанному условию (временной диапазон), Selena позволяет партиционировать данные с различной временной гранулярностью.

Синтаксис

```
ALTER TABLE [<db_name>.]<table_name>
PARTITION BY <time_expr>
[WHERE <time_range_column>] BETWEEN <start_time> AND <end_time>
```

Параметры

`PARTITION BY <time_expr>`

Обязательный: ДА
Описание: Указывает новую временную гранулярность для стратегии партиционирования, например, `PARTITION BY date_trunc('month', dt)`.

`WHERE <time_range_column> BETWEEN <start_time> AND <end_time>`

Обязательный: ДА
Описание: Указывает временной диапазон partitions, которые необходимо объединить. Partitions в пределах этого диапазона будут объединены на основе правил, определенных в предложении `PARTITION BY`.

Пример

Объединить partitions в таблице `site_access1` и изменить гранулярность времени partition с дня на месяц. Временной диапазон partitions, которые необходимо объединить, - от `2024-01-01` до `2024-03-31`.

```
ALTER TABLE site_access1 PARTITION BY date_trunc('month', event_day)
BETWEEN '2024-01-01' AND '2024-03-31';
```

После слияния:

- Partitions уровня дня с `2024-01-01` по `2024-01-31` объединяются в partition уровня месяца `2024-01`.
- Partitions уровня дня с `2024-02-01` по `2024-02-29` объединяются в partition уровня месяца `2024-02`.
- Partitions уровня дня с `2024-03-01` по `2024-03-31` объединяются в partition уровня месяца `2024-03`.

Примечания по использованию

- Слияние поддерживается только для expression partitions на основе временной функции.
- Слияние partitions с несколькими столбцами partition не поддерживается.
- Параллельное выполнение слияния и операций Schema Change/DML не поддерживается.

Ограничения

- Начиная с версии v1.5.2, режим shared-data Selena поддерживает выражение временной функции. И начиная с версии v1.5.2, режим shared-data Selena дополнительно поддерживает выражение столбца.
- В настоящее время использование CTAS для создания таблиц, настроенных с expression partitioning, не поддерживается.
- В настоящее время использование Spark Load для загрузки данных в таблицы, использующие expression partitioning, не поддерживается.
- Когда оператор `ALTER TABLE <table_name> DROP PARTITION <partition_name>` используется для удаления partition, созданного с использованием выражения столбца, данные в partition удаляются напрямую и не могут быть восстановлены.
- Начиная с версий v1.5.2, Selena поддерживает резервное копирование и восстановление таблиц, созданных со стратегией expression partitioning.

Приложение

Поддерживаемые временные функции

Expression partitioning поддерживает следующие функции:

Временные функции:

- timediff
- datediff
- to_days
- years_add/sub
- quarters_add/sub
- months_add/sub
- weeks_add/sub
- date_add/sub
- days_add/sub

- hours_add/sub
- minutes_add/sub
- seconds_add/sub
- milliseconds_add/sub
- date_trunc
- date_format(YmdHiSf/YmdHisf)
- str2date(YmdHiSf/YmdHisf)
- str_to_date(YmdHiSf/YmdHisf)
- to_iso8601
- to_date
- unix_timestamp
- from_unixtime(YmdHiSf/YmdHisf)
- time_slice

Другие функции:

- add
- subtract
- cast

> **ПРИМЕЧАНИЕ**
>
> - Поддерживается комбинированное использование нескольких временных функций.
> - Системный часовой пояс по умолчанию используется для всех временных функций, перечисленных выше.
> - Формат значения временной функции, `YmdHiSf`, должен начинаться с наиболее грубой временной гранулярности, `%Y`. Форматы, которые начинаются с более тонкой временной гранулярности, например, `%m-%d`, не допускаются.
>
> **Пример**
>
> `PARTITION BY from_unixtime(cast(str as INT) + 3600, '%Y-%m-%d')`

2.4.6 List partitioning (Legacy)

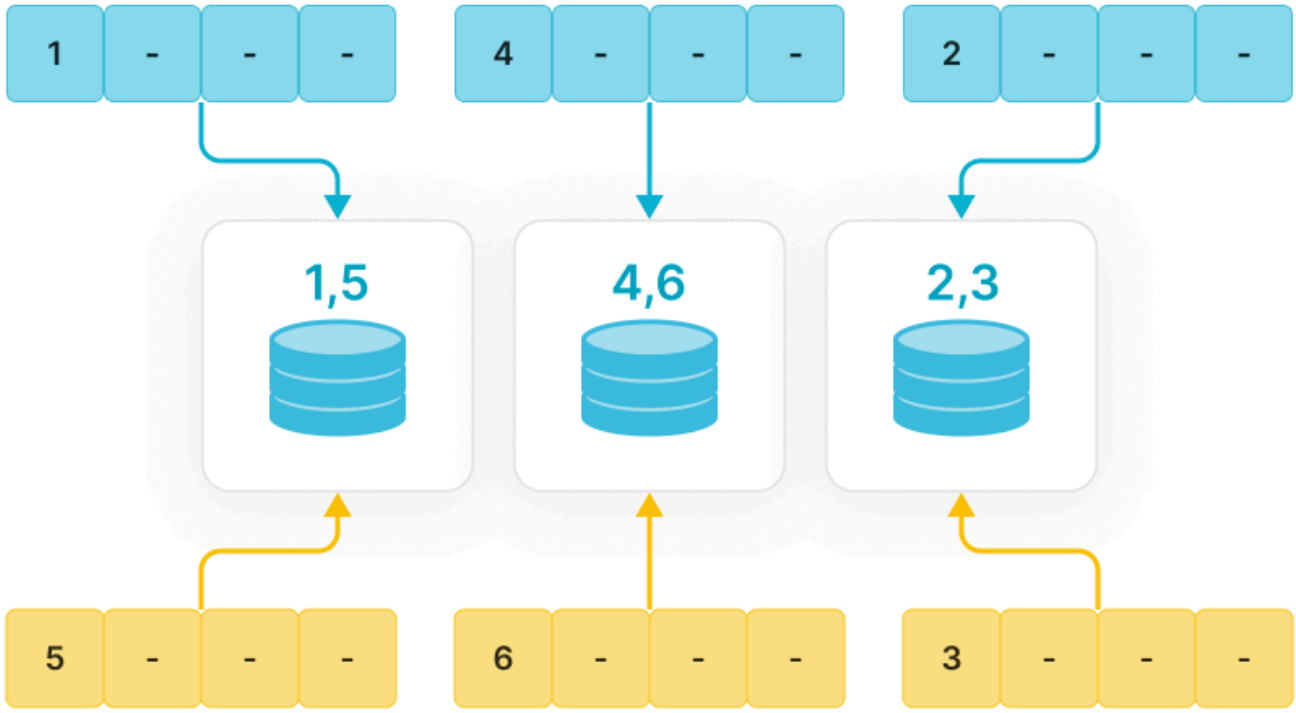
Начиная с версии v1.5.2, Selena поддерживает list partitioning. Данные разбиваются на основе заранее определенного списка значений для каждой partition, что может ускорить

запросы и упростить управление в соответствии с перечисляемыми значениями.

> **ПРИМЕЧАНИЕ**
>
> Обратите внимание, что начиная с версии v1.5.2, expression partitioning был дополнительно оптимизирован для унификации всех стратегий partitioning и поддержки более сложных решений. Он рекомендуется в большинстве случаев и заменит стратегию list partitioning в будущих релизах.

Введение

Вам необходимо явно указать список значений столбца в каждой partition. Эти значения не обязательно должны быть непрерывными, в отличие от непрерывного временного или числового диапазона, требуемого в Range Partitioning. Во время загрузки данных Selena будет хранить данные в соответствующей partition на основе соответствия между значениями столбца partitioning данных и заранее определенными значениями столбца для каждой partition.



List partitioning подходит для хранения данных, столбцы которых содержат небольшое количество enum значений, и вы часто запрашиваете и управляете данными на основе этих enum значений. Например, столбцы представляют географические местоположения, штаты и категории. Каждое значение в столбце представляет независимую категорию.

Разбивая данные на основе enum значений, вы можете улучшить производительность запросов и упростить управление данными.

List partitioning особенно полезен для сценариев, где partition должна включать несколько значений для каждого столбца partitioning. Например, если таблица включает столбец `City`, представляющий город, из которого человек, и вы часто запрашиваете и управляете данными по штатам и городам. При создании таблицы вы можете использовать столбец `City` в качестве столбца partitioning для List Partitioning и указать, что данные различных городов в пределах одного штата помещаются в одну partition, например `PARTITION pCalifornia VALUES IN ("Los Angeles", "San Francisco", "San Diego")`. Этот метод может ускорить запросы на основе штатов и городов, упрощая управление данными.

Если partition должна содержать только данные с одинаковым значением каждого столбца partitioning, рекомендуется использовать expression partitioning.

Сравнение между list partitioning и expression partitioning

Основное различие между list partitioning и expression partitioning (рекомендуется) заключается в том, что list partitioning требует вручную создавать partitions одну за другой. С другой стороны, expression partitioning может автоматически создавать partitions во время загрузки для упрощения partitioning. И в большинстве случаев expression partitioning может заменить list partitioning. Конкретное сравнение между ними показано в следующей таблице:

Метод partitioning	List partitioning	Expression partitioning
Синтаксис	`PARTITION BY LIST (partition_columns)( PARTITION <partition_name> VALUES IN (value_list) [, ...] )`	`PARTITION BY <partition_columns>`
Несколько значений для каждого столбца partitioning в partition	Поддерживается. Partition может хранить данные с различными значениями в каждом столбце partitioning. В следующем примере, когда загруженные данные содержат значения `Los Angeles`, `San Francisco` и `San Diego` в столбце `city`, все данные хранятся в одной partition `pCalifornia`. `PARTITION BY LIST (city) ( PARTITION pCalifornia VALUES IN ("L`	Не поддерживается. Partition хранит данные с одинаковым значением в столбце partitioning. Например, выражение `PARTITION BY (city)` используется в expression partitioning. Когда загруженные данные содержат значения `Los Angeles`, `San Francisco` и `San Diego` в столбце `city`, Selena автоматически создаст три partitions `pLosAngeles`, `pSanFrancisco` и `pSanDiego`. Три partitions соответственно

Метод partitioning	List partitioning	Expression partitioning
	`os Angeles","San Francisco","San Diego")  [, ...] )`	хранят данные со значениями `Los Angeles,` `San Francisco` и `San Diego` в столбце `city`.
Создание partitions перед загрузкой данных	Поддерживается. Partitions должны быть созданы при создании таблицы.	Не требуется. Partitions могут быть автоматически созданы во время загрузки данных.
Автоматическое создание List partitions во время загрузки данных	Не поддерживается. Если partition, соответствующая данным, не существует во время загрузки данных, возвращается ошибка.	Поддерживается. Если partition, соответствующая данным, не существует во время загрузки данных, Selena автоматически создает partition для хранения данных. Каждая partition может содержать только данные с одинаковым значением для столбца partitioning.
SHOW CREATE TABLE	Возвращает определение partitions в операторе CREATE TABLE.	После загрузки данных оператор возвращает результат с предложением partition, использованным в операторе CREATE TABLE, то есть `PARTITION BY (partition_columns)`. Однако возвращенный результат не показывает автоматически созданные partitions. Если вам нужно просмотреть автоматически созданные partitions, выполните `SHOW PARTITIONS FROM <table_name>`.

Использование

Синтаксис

```
PARTITION BY LIST (partition_columns) (
    PARTITION <partition_name> VALUES IN (value_list)
    [, ...]
)

partition_columns::=
    <column> [,<column> [, ...] ]

value_list ::=
    value_item [, value_item [, ...] ]

value_item ::=
    { <value> | ( <value> [, <value>, [, ...] ] ) }
```

Параметры

Параметры	Обязательно	Описание
`partition_columns`	ДА	Имена столбцов partitioning. Значения столбца partitioning могут быть строковыми (BINARY не поддерживается), датой или datetime, целочисленными и булевыми значениями. Начиная с v1.5.2, столбец partitioning допускает значения `NULL`.
`partition_name`	ДА	Имя partition. Рекомендуется устанавливать подходящие имена partitions на основе бизнес-сценария для разделения данных в разных partitions.
`value_list`	ДА	Список значений столбца partitioning в partition.

Примеры

Пример 1: Предположим, вы часто запрашиваете детали выставления счетов дата-центра на основе штатов или городов. При создании таблицы вы можете указать столбец partitioning как `city` и указать, что каждая partition хранит данные городов в пределах одного штата. Этот метод может ускорить запросы для конкретных штатов или городов и упростить управление данными.

```
CREATE TABLE t_recharge_detail1 (
    id bigint,
    user_id bigint,
    recharge_money decimal(32,2),
    city varchar(20) not null,
    dt varchar(20) not null
)
DUPLICATE KEY(id)
PARTITION BY LIST (city) (
   PARTITION pLos_Angeles VALUES IN ("Los Angeles"),
   PARTITION pSan_Francisco VALUES IN ("San Francisco")
)
DISTRIBUTED BY HASH(`id`);
```

Пример 2: Предположим, вы часто запрашиваете детали выставления счетов дата-центра на основе временных диапазонов и конкретных штатов или городов. При создании таблицы вы можете указать столбцы partitioning как `dt` и `city`. Таким образом, данные конкретных дат и конкретных штатов или городов хранятся в одной partition, улучшая скорость запросов и упрощая управление данными.

```
CREATE TABLE t_recharge_detail4 (
    id bigint,
    user_id bigint,
    recharge_money decimal(32,2),
    city varchar(20) not null,
    dt varchar(20) not null
) ENGINE=OLAP
DUPLICATE KEY(id)
PARTITION BY LIST (dt,city) (
   PARTITION p202204_California VALUES IN (
       ("2022-04-01", "Los Angeles"),
       ("2022-04-01", "San Francisco"),
       ("2022-04-02", "Los Angeles"),
       ("2022-04-02", "San Francisco")
    ),
   PARTITION p202204_Texas VALUES IN (
       ("2022-04-01", "Houston"),
       ("2022-04-01", "Dallas"),
       ("2022-04-02", "Houston"),
       ("2022-04-02", "Dallas")
   )
)
DISTRIBUTED BY HASH(`id`);
```

Ограничения

- List partitioning не поддерживает dynamic partitioning и создание нескольких partitions одновременно.
- В настоящее время режим shared-data Selena не поддерживает эту функцию.
- Когда оператор `ALTER TABLE <table_name> DROP PARTITION <partition_name>;` используется для удаления partition, созданной с использованием list partitioning, данные в partition непосредственно удаляются и не могут быть восстановлены.
- Начиная с версий v1.5.2, Selena поддерживает резервное копирование и восстановление таблиц, созданных со стратегией list partitioning.
- Начиная с версии v1.5.2, Selena поддерживает создание асинхронных материализованных представлений с базовыми таблицами, созданными со стратегией list partitioning.

2.4.7 Range partitioning (Устаревший)

Range partitioning подходит для хранения простых непрерывных данных, таких как данные временных рядов или непрерывные числовые данные.

На основе range partitioning вы можете создавать partitions с использованием стратегии динамического партиционирования, которая позволяет управлять временем жизни (TTL) partitions.

> **ПРИМЕЧАНИЕ**
>
> Обратите внимание, что начиная с версии v1.5.2, expression partitioning дополнительно оптимизирован для унификации всех стратегий партиционирования и поддерживает более сложные решения. Он рекомендуется в большинстве случаев и заменит стратегию range partitioning в будущих релизах.

Введение

Range partitioning подходит для часто запрашиваемых данных на основе непрерывных диапазонов дат/чисел. Кроме того, его можно применять в некоторых особых случаях, когда исторические данные необходимо партиционировать по месяцам, а недавние данные нужно партиционировать по дням.

Вам необходимо явно определить столбцы партиционирования данных и установить отношение сопоставления между partitions и диапазонами значений столбцов партиционирования. Во время загрузки данных Selena назначает данные соответствующим partitions на основе диапазонов, к которым относятся значения столбцов партиционирования данных.

Что касается типа данных столбцов партиционирования, до версии v1.5.2 range partitioning поддерживал только столбцы партиционирования типов date и integer. Начиная с версии v1.5.2, три специфические временные функции могут использоваться как столбцы partition. При явном определении отношения сопоставления между partitions и диапазонами значений столбцов партиционирования вам необходимо сначала использовать специфическую временную функцию для преобразования значений столбцов партиционирования из timestamps или strings в значения date, а затем разделить partitions на основе преобразованных значений date.

К СВЕДЕНИЮ

- Если значение столбца партиционирования является timestamp, вам необходимо использовать функцию from_unixtime или from_unixtime_ms для преобразования timestamp в значение date при разделении partitions. Когда используется функция from_unixtime, столбец партиционирования поддерживает только типы INT и BIGINT. Когда используется функция from_unixtime_ms, столбец партиционирования поддерживает только тип BIGINT.
- Если значение столбца партиционирования является строкой (тип STRING, VARCHAR или CHAR), вам необходимо использовать функцию str2date для преобразования строки в значение date при разделении partitions.

Использование

Синтаксис

```
PARTITION BY RANGE ( partition_columns | function_expression ) (
single_range_partition | multi_range_partitions )

partition_columns ::=
    <column> [, ...]

function_expression ::=
      from_unixtime(column)
    | from_unixtime_ms(column)
    | str2date(column)

single_range_partition ::=
    PARTITION <partition_name> VALUES partition_key_desc

partition_key_desc ::=
          LESS THAN { MAXVALUE | value_list }
        | value_range

-- value_range - это левозамкнутый, правооткрытый интервал. Пример:
`[202201, 202212)`.
-- Вы должны явно указать скобку `[` в полуоткрытом интервале.
value_range ::=
    [value_list, value_list)

value_list ::=
    ( <value> [, ...] )


multi_range_partitions ::=
        { PARTITIONS START ("<start_date_value>") END ("<end_date_value>")
EVERY ( INTERVAL <int_value> time_unit )
        | PARTITIONS START ("<start_integer_value>") END ("
<end_integer_value>") EVERY ( <int_value> ) } -- Значения столбцов
partition всё равно должны быть заключены в двойные кавычки, даже если
значения столбцов partition, указанные в START и END, являются целыми
числами. Однако значения интервалов в предложении EVERY не нужно заключать
в двойные кавычки.

time_unit ::=
    HOUR | DAY | WEEK | MONTH | YEAR
```

Параметры

Параметры	Описание
`partition_columns`	Имена столбцов партиционирования. Значения столбцов партиционирования могут быть строковыми (BINARY не поддерживается), date или datetime, или целыми числами.
`function_expression`	Функциональное выражение, которое преобразует столбец партиционирования в определенные типы данных. Поддерживаемые функции: from_unixtime, from_unixtime_ms и str2date. **ПРИМЕЧАНИЕ** Range partitioning поддерживает одно и только одно функциональное выражение.
`partition_name`	Имя partition. Рекомендуется устанавливать соответствующие имена partitions на основе бизнес-сценария для различения данных в разных partitions.

Пример

1. Создание partitions путем ручного определения диапазонов значений на основе столбца партиционирования типа date.

```
PARTITION BY RANGE(date_col)(
    PARTITION p1 VALUES LESS THAN ("2020-01-31"),
    PARTITION p2 VALUES LESS THAN ("2020-02-29"),
    PARTITION p3 VALUES LESS THAN ("2020-03-31")
)
```

2. Создание partitions путем ручного определения диапазонов значений на основе столбца партиционирования типа integer.

```
PARTITION BY RANGE (int_col) (
    PARTITION p1 VALUES LESS THAN ("20200131"),
    PARTITION p2 VALUES LESS THAN ("20200229"),
    PARTITION p3 VALUES LESS THAN ("20200331")
)
```

3. Создание нескольких partitions с одинаковым интервалом дат.

```
PARTITION BY RANGE (date_col) (
    START ("2021-01-01") END ("2021-01-04") EVERY (INTERVAL 1 DAY)
)
```

4. Создание нескольких partitions с различными интервалами дат.

```
PARTITION BY RANGE (date_col) (
    START ("2019-01-01") END ("2021-01-01") EVERY (INTERVAL 1 YEAR),
    START ("2021-01-01") END ("2021-05-01") EVERY (INTERVAL 1 MONTH),
    START ("2021-05-01") END ("2021-05-04") EVERY (INTERVAL 1 DAY)
)
```

5. Создание нескольких partitions с одинаковым целочисленным интервалом.

```
PARTITION BY RANGE (int_col) (
    START ("1") END ("10") EVERY (1)
)
```

6. Создание нескольких partitions с различными целочисленными интервалами.

```
PARTITION BY RANGE (int_col) (
    START ("1") END ("10") EVERY (1),
    START ("10") END ("100") EVERY (10)
)
```

7. Использование from_unixtime для преобразования столбца partition типа timestamp (string) в тип date.

```
PARTITION BY RANGE(from_unixtime(timestamp_col)) (
    PARTITION p1 VALUES LESS THAN ("2021-01-01"),
    PARTITION p2 VALUES LESS THAN ("2021-01-02"),
    PARTITION p3 VALUES LESS THAN ("2021-01-03")
)
```

8. Использование str2date для преобразования столбца partition типа string в тип date.

```
PARTITION BY RANGE(str2date(string_col, '%Y-%m-%d'))(
    PARTITION p1 VALUES LESS THAN ("2021-01-01"),
    PARTITION p2 VALUES LESS THAN ("2021-01-02"),
    PARTITION p3 VALUES LESS THAN ("2021-01-03")
)
```

Dynamic partitioning

Selena поддерживает динамическое партиционирование, которое может автоматически управлять временем жизни (TTL) partitions, например, партиционирование новых входных данных в таблицах и удаление устаревших partitions. Эта функция значительно снижает затраты на обслуживание.

Включение динамического партиционирования

Возьмем таблицу `site_access` в качестве примера. Для включения динамического партиционирования вам необходимо настроить параметр PROPERTIES. Информацию о элементах конфигурации см. в CREATE TABLE.

```
CREATE TABLE site_access(
    event_day DATE,
    site_id INT DEFAULT '10',
    city_code VARCHAR(100),
    user_name VARCHAR(32) DEFAULT '',
    pv BIGINT DEFAULT '0'
)
DUPLICATE KEY(event_day, site_id, city_code, user_name)
PARTITION BY RANGE(event_day)(
    PARTITION p20200321 VALUES LESS THAN ("2020-03-22"),
    PARTITION p20200322 VALUES LESS THAN ("2020-03-23"),
    PARTITION p20200323 VALUES LESS THAN ("2020-03-24"),
    PARTITION p20200324 VALUES LESS THAN ("2020-03-25")
)
DISTRIBUTED BY HASH(event_day, site_id)
PROPERTIES(
    "dynamic_partition.enable" = "true",
    "dynamic_partition.time_unit" = "DAY",
    "dynamic_partition.start" = "-3",
    "dynamic_partition.end" = "3",
    "dynamic_partition.prefix" = "p",
    "dynamic_partition.buckets" = "32",
    "dynamic_partition.history_partition_num" = "0"
);
```

`PROPERTIES`:

параметр	обязательный	описание
dynamic_partition.enable	Нет	Включает динамическое партиционирование. Допустимые значения: `TRUE` и `FALSE`. Значение по умолчанию: `TRUE`.
dynamic_partition.time_unit	Да	Временная гранулярность для динамически создаваемых partitions. Это обязательный параметр. Допустимые значения: `HOUR`, `DAY`, `WEEK`, `MONTH` и `YEAR`. Временная гранулярность определяет формат суффикса для динамически создаваемых partitions. • Если значение `HOUR`, столбец партиционирования может быть только типа DATETIME и не может быть типа DATE. Формат суффикса для динамически создаваемых partitions - yyyyMMddHH, например, `2020032101`. • Если значение `DAY`, формат суффикса для динамически создаваемых partitions - yyyyMMdd. Пример суффикса имени partition - `20200321`. • Если значение `WEEK`, формат суффикса для динамически создаваемых partitions - yyyy_ww, например `2020_13` для 13-й недели 2020 года. • Если значение `MONTH`, формат суффикса для динамически создаваемых partitions - yyyyMM, например `202003`. • Если значение `YEAR`, формат суффикса для динамически создаваемых partitions - yyyy, например `2020`.
dynamic_partition.time_zone	Нет	Часовой пояс для динамических partitions, который по умолчанию совпадает с часовым поясом системы.
dynamic_partition.start	Нет	Начальное смещение динамического партиционирования. Значение этого параметра должно быть отрицательным целым числом. Partitions до этого смещения будут удалены на основе текущего дня, недели или месяца, который определяется значением параметра `dynamic_partition.time_unit`. Значение по умолчанию: `Integer.MIN_VALUE`, а именно -2147483648, что означает, что исторические partitions не будут удаляться.
dynamic_partition.end	Да	Конечное смещение динамического партиционирования. Значение этого параметра должно быть положительным целым числом. Partitions от текущего дня, недели или месяца до конечного смещения будут созданы заранее.
dynamic_partition.prefix	Нет	Префикс, добавляемый к именам динамических partitions. Значение по умолчанию: `p`.
dynamic_partition.buckets	Нет	Количество buckets для каждого динамического partition. Значение по умолчанию совпадает с количеством buckets, определенным зарезервированным словом BUCKETS или автоматически установленным Selena.
dynamic_partition.history_p	Нет	Количество исторических partitions, создаваемых механизмом динамического партиционирования, со значением по умолчанию `0`. Когда значение больше 0, исторические partitions создаются заранее. Начиная с версии v1.5.2, Selena поддерживает этот параметр.

параметр	обязательный	описание
artition_num		
dynamic_partition.start_day_of_week	Нет	Когда `dynamic_partition.time_unit` равен `WEEK`, этот параметр используется для указания первого дня каждой недели. Допустимые значения: от `1` до `7`. `1` означает понедельник, а `7` означает воскресенье. Значение по умолчанию: `1`, что означает, что каждая неделя начинается с понедельника.
dynamic_partition.start_day_of_month	Нет	Когда `dynamic_partition.time_unit` равен `MONTH`, этот параметр используется для указания первого дня каждого месяца. Допустимые значения: от `1` до `28`. `1` означает 1-е число каждого месяца, а `28` означает 28-е число каждого месяца. Значение по умолчанию: `1`, что означает, что каждый месяц начинается с 1-го числа. Первый день не может быть 29-м, 30-м или 31-м.
dynamic_partition.replication_num	Нет	Количество реплик для tablets в динамически создаваемых partitions. Значение по умолчанию совпадает с количеством реплик, настроенным при создании таблицы.

> **ПРИМЕЧАНИЕ**
>
> Когда столбец partition имеет тип INT, его формат должен быть `yyyyMMdd`, независимо от гранулярности времени partition.

Конфигурация FE:

`dynamic_partition_check_interval_seconds`: интервал для планирования динамического партиционирования. Значение по умолчанию: 600 секунд, что означает, что ситуация с partitions проверяется каждые 10 минут, чтобы определить, соответствуют ли partitions условиям динамического партиционирования, указанным в `PROPERTIES`. Если нет, partitions будут создаваться и удаляться автоматически.

Просмотр динамических partitions

После включения динамических partitions для таблицы входные данные непрерывно и автоматически партиционируются. Вы можете просмотреть текущие partitions, используя следующий оператор. Например, если текущая дата - 2020-03-25, вы можете видеть только partitions в диапазоне времени от 2020-03-25 до 2020-03-28.

```
SHOW PARTITIONS FROM site_access;

[types: [DATE]; keys: [2020-03-25]; ..types: [DATE]; keys: [2020-03-26]; )
[types: [DATE]; keys: [2020-03-26]; ..types: [DATE]; keys: [2020-03-27]; )
[types: [DATE]; keys: [2020-03-27]; ..types: [DATE]; keys: [2020-03-28]; )
[types: [DATE]; keys: [2020-03-28]; ..types: [DATE]; keys: [2020-03-29]; )
```

Если вы хотите создать исторические partitions при создании таблицы, вам необходимо указать `dynamic_partition.history_partition_num` для определения количества исторических partitions, которые нужно создать. Например, если вы установите `dynamic_partition.history_partition_num` в `3` при создании таблицы и текущая дата - 2020-03-25, вы увидите только partitions в диапазоне времени от 2020-03-22 до 2020-03-28.

```
SHOW PARTITIONS FROM site_access;

[types: [DATE]; keys: [2020-03-22]; ..types: [DATE]; keys: [2020-03-23]; )
[types: [DATE]; keys: [2020-03-23]; ..types: [DATE]; keys: [2020-03-24]; )
[types: [DATE]; keys: [2020-03-24]; ..types: [DATE]; keys: [2020-03-25]; )
[types: [DATE]; keys: [2020-03-25]; ..types: [DATE]; keys: [2020-03-26]; )
[types: [DATE]; keys: [2020-03-26]; ..types: [DATE]; keys: [2020-03-27]; )
[types: [DATE]; keys: [2020-03-27]; ..types: [DATE]; keys: [2020-03-28]; )
[types: [DATE]; keys: [2020-03-28]; ..types: [DATE]; keys: [2020-03-29]; )
```

Изменение свойств динамического партиционирования

Вы можете использовать оператор ALTER TABLE для изменения свойств динамического партиционирования, например, отключения динамического партиционирования. Возьмем следующий оператор в качестве примера.

```
ALTER TABLE site_access
SET("dynamic_partition.enable"="false");
```

> **ПРИМЕЧАНИЕ**
>
> - Чтобы проверить свойства динамического партиционирования таблицы, выполните оператор SHOW CREATE TABLE.
> - Вы также можете использовать оператор ALTER TABLE для изменения других свойств таблицы.

2.4.8 Временная партиция

В этом разделе описывается, как использовать функцию временной партиции.

Вы можете создать временные партиции в партиционированной таблице, которая уже имеет определенные правила партиционирования, и определить новые стратегии распределения данных для этих временных партиций. Временные партиции могут служить в качестве временных носителей данных, когда вы атомарно перезаписываете данные в партиции или когда вы корректируете стратегии партиционирования и bucketing'a. Для временных партиций вы можете сбросить стратегии распределения данных, такие как диапазоны партиций, количество bucket'ов и свойства, такие как количество реплик и носитель хранения, чтобы соответствовать конкретным требованиям.

Вы можете использовать функцию временной партиции в следующих сценариях:

- Атомарная операция перезаписи

  Если вам нужно переписать данные в партиции, обеспечивая при этом возможность запроса данных во время процесса перезаписи, вы можете сначала создать временную партицию на основе оригинальной формальной партиции и загрузить новые данные во временную партицию. Затем вы можете использовать операцию замены для атомарной замены оригинальной формальной партиции временной партицией. Для операций атомарной перезаписи на непартиционированных таблицах см. ALTER TABLE - SWAP.

- Корректировка параллелизма запросов данных партиции

  Если вам нужно изменить количество bucket'ов для партиции, вы можете сначала создать временную партицию с тем же диапазоном партиции, что и оригинальная формальная партиция, и указать новое количество bucket'ов. Затем вы можете использовать команду `INSERT INTO` для загрузки данных оригинальной формальной

партиции во временную партицию. Наконец, вы можете использовать операцию замены для атомарной замены оригинальной формальной партиции временной партицией.

- Изменение правил партиционирования

  Если вы хотите изменить стратегию партиционирования, такую как слияние партиций или разделение большой партиции на несколько меньших партиций, вы можете сначала создать временные партиции с ожидаемыми объединенными или разделенными диапазонами. Затем вы можете использовать команду `INSERT INTO` для загрузки данных оригинальных формальных партиций во временные партиции. Наконец, вы можете использовать операцию замены для атомарной замены оригинальных формальных партиций временными партициями.

Создание временных партиций

Вы можете создать одну или несколько партиций за раз с помощью команды ALTER TABLE.

Синтаксис

Создание одной временной партиции

```
ALTER TABLE <table_name>
ADD TEMPORARY PARTITION <temporary_partition_name> VALUES [("value1"),
{MAXVALUE|("value2")})]
[(partition_desc)]
[DISTRIBUTED BY HASH(<bucket_key>)];
ALTER TABLE <table_name>
ADD TEMPORARY PARTITION <temporary_partition_name> VALUES LESS THAN
{MAXVALUE|(<"value">)}
[(partition_desc)]
[DISTRIBUTED BY HASH(<bucket_key>)];
```

Создание нескольких партиций за раз

```
ALTER TABLE <table_name>
ADD TEMPORARY PARTITIONS START ("value1") END ("value2") EVERY {(INTERVAL
<num> <time_unit>)|<num>}
[(partition_desc)]
[DISTRIBUTED BY HASH(<bucket_key>)];
```

Параметры

`partition_desc`: определяет количество bucket'ов и свойства для временных партиций, такие как количество реплик и носитель хранения.

Примеры

Создайте временную партицию `tp1` в таблице `site_access` и укажите ее диапазон как `[2020-01-01, 2020-02-01)`, используя синтаксис `VALUES [(...), (...)]`.

```
ALTER TABLE site_access
ADD TEMPORARY PARTITION tp1 VALUES [("2020-01-01"), ("2020-02-01"));
```

Создайте временную партицию `tp2` в таблице `site_access` и укажите ее верхнюю границу как `2020-03-01`, используя синтаксис `VALUES LESS THAN (...)`. Selena использует верхнюю границу предыдущей временной партиции в качестве нижней границы этой временной партиции, генерируя временную партицию с левозамкнутым и правооткрытым диапазоном `[2020-02-01, 2020-03-01)`.

```
ALTER TABLE site_access
ADD TEMPORARY PARTITION tp2 VALUES LESS THAN ("2020-03-01");
```

Создайте временную партицию `tp3` в таблице `site_access`, укажите ее верхнюю границу как `2020-04-01`, используя синтаксис `VALUES LESS THAN (...)`, и укажите количество реплик как `1`.

```
ALTER TABLE site_access
ADD TEMPORARY PARTITION tp3 VALUES LESS THAN ("2020-04-01")
 ("replication_num" = "1")
DISTRIBUTED BY HASH (site_id);
```

Создайте несколько партиций за раз в таблице `site_access`, используя синтаксис `START (...) END (...) EVERY (...)`, и укажите диапазон этих партиций как `[2020-04-01, 2021-01-01)` с гранулярностью партиции в один месяц.

```
ALTER TABLE site_access
ADD TEMPORARY PARTITIONS START ("2020-04-01") END ("2021-01-01") EVERY
(INTERVAL 1 MONTH);
```

Примечания по использованию

- Партиционирующая колонка для временной партиции должна быть такой же, как партиционирующая колонка для оригинальной формальной партиции, на основе которой вы создаете временную партицию, и не может быть изменена.
- Имя временной партиции не может совпадать с именем любой формальной партиции или другой временной партиции.
- Диапазоны всех временных партиций в таблице не могут перекрываться, но диапазоны временной партиции и формальной партиции могут перекрываться.

Просмотр временных партиций

Вы можете просмотреть временные партиции с помощью команды SHOW TEMPORARY PARTITIONS.

```
SHOW TEMPORARY PARTITIONS FROM [db_name.]table_name [WHERE] [ORDER BY]
[LIMIT]
```

Загрузка данных во временные партиции

Вы можете загрузить данные в одну или несколько временных партиций с помощью команды `INSERT INTO`, STREAM LOAD или BROKER LOAD.

Загрузка данных с помощью команды `INSERT INTO`

Пример:

```
INSERT INTO site_access TEMPORARY PARTITION (tp1) VALUES ("2020-01-
01",1,"ca","lily",4);
INSERT INTO site_access TEMPORARY PARTITION (tp2) SELECT * FROM
site_access_copy PARTITION p2;
INSERT INTO site_access TEMPORARY PARTITION (tp3, tp4,...) SELECT * FROM
site_access_copy PARTITION (p3, p4,...);
```

Подробный синтаксис и описание параметров см. в разделе INSERT INTO.

Загрузка данных с помощью STREAM LOAD

Пример:

```
curl --location-trusted -u root: -H "label:123" -H "Expect:100-continue" -H
"temporary_partitions: tp1, tp2, ..." -T testData \
    http://host:port/api/example_db/site_access/_stream_load
```

Подробный синтаксис и описание параметров см. в разделе STREAM LOAD.

Загрузка данных с помощью BROKER LOAD

Пример:

```
LOAD LABEL example_db.label1
(
    DATA INFILE("hdfs://hdfs_host:hdfs_port/user/selena/data/input/file")
    INTO TABLE my_table
    TEMPORARY PARTITION (tp1, tp2, ...)
    ...
)
WITH BROKER
(
    StorageCredentialParams
);
```

Обратите внимание, что `StorageCredentialParams` представляет группу параметров аутентификации, которые варьируются в зависимости от выбранного вами метода аутентификации. Подробный синтаксис и описание параметров см. в разделе Broker Load.

Загрузка данных с помощью ROUTINE LOAD

Пример:

```
CREATE ROUTINE LOAD example_db.site_access ON example_tbl
COLUMNS(col, col2,...),
TEMPORARY PARTITIONS(tp1, tp2, ...)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "ordertest"
);
```

Подробный синтаксис и описание параметров см. в разделе CREATE ROUTINE LOAD.

Запрос данных во временных партициях

Вы можете использовать оператор SELECT для запроса данных в указанных временных партициях.

```
SELECT * FROM
site_access TEMPORARY PARTITION (tp1);

SELECT * FROM
site_access TEMPORARY PARTITION (tp1, tp2, ...);

SELECT event_day,site_id,pv FROM
site_access TEMPORARY PARTITION (tp1, tp2, ...);
```

Вы можете использовать клаузу JOIN для запроса данных во временных партициях из двух таблиц.

```
SELECT * FROM
site_access TEMPORARY PARTITION (tp1, tp2, ...)
JOIN
site_access_copy TEMPORARY PARTITION (tp1, tp2, ...)
ON site_access.site_id=site_access1.site_id and
site_access.event_day=site_access1.event_day;
```

Замена оригинальных формальных партиций временными партициями

Вы можете использовать оператор ALTER TABLE для замены оригинальной формальной партиции временной партицией, тем самым создавая новую формальную партицию.

> **ПРИМЕЧАНИЕ**
>
> Оригинальная формальная партиция и временная партиция, с которыми вы работали в операторе ALTER TABLE, удаляются и не могут быть восстановлены.

Синтаксис

```
ALTER TABLE table_name REPLACE PARTITION (partition_name) WITH TEMPORARY
PARTITION (temporary_partition_name1, ...)
PROPERTIES ("key" = "value");
```

Параметры

- **strict_range**

  Значение по умолчанию: `true`.

  Когда этот параметр установлен в `true`, объединения диапазонов всех оригинальных формальных партиций должны быть точно такими же, как объединения диапазонов временных партиций, которые используются для замены. Когда этот параметр установлен в `false`, вам нужно только убедиться, что диапазоны новых формальных партиций не перекрываются с другими формальными партициями после замены.

  - Пример 1:

    В следующем примере объединения оригинальных формальных партиций `p1`, `p2` и `p3` такие же, как объединения временных партиций `tp1` и `tp2`, и вы можете использовать `tp1` и `tp2` для замены `p1`, `p2` и `p3`.

```
# Диапазоны оригинальных формальных партиций p1, p2 и p3 =>
Объединения этих диапазонов
[10, 20), [20, 30), [40, 50) => [10, 30), [40, 50)

# Диапазоны временных партиций tp1 и tp2 => Объединения этих
диапазонов
[10, 30), [40, 45), [45, 50) => [10, 30), [40, 50)
```

- Пример 2:

  В следующем примере объединение диапазонов оригинальной формальной партиции отличается от объединения диапазонов временных партиций. Если значение параметра `strict_range` установлено в `true`, временные партиции `tp1` и `tp2` не могут заменить оригинальную формальную партицию `p1`. Если значение установлено в `false`, и диапазоны [10, 30) и [40, 50) временных партиций не перекрываются с другими формальными партициями, временные партиции могут заменить оригинальную формальную партицию.

```
# Диапазон оригинальной формальной партиции p1 => Объединение
диапазона
[10, 50) => [10, 50)

# Диапазоны временных партиций tp1 и tp2 => Объединения этих
диапазонов
[10, 30), [40, 50) => [10, 30), [40, 50)
```

- **use_temp_partition_name**

  Значение по умолчанию: `false`.

  Если количество оригинальных формальных партиций совпадает с количеством временных партиций, используемых для замены, имена новых формальных партиций остаются неизменными после замены, когда этот параметр установлен в `false`. Когда этот параметр установлен в `true`, имена временных партиций используются в качестве имен новых формальных партиций после замены.

  В следующем примере имя партиции новой формальной партиции остается `p1` после замены, когда этот параметр установлен в `false`. Однако связанные с ней данные и свойства заменяются данными и свойствами временной партиции `tp1`. Когда этот

параметр установлен в `true`, имя партиции новой формальной партиции изменяется на `tp1` после замены. Оригинальная формальная партиция `p1` больше не существует.

```
ALTER TABLE tbl1 REPLACE PARTITION (p1) WITH TEMPORARY PARTITION (tp1);
```

Если количество формальных партиций, подлежащих замене, отличается от количества временных партиций, используемых для замены, и этот параметр остается значением по умолчанию `false`, значение `false` этого параметра является недействительным.

В следующем примере после замены имя новой формальной партиции изменяется на `tp1`, и оригинальные формальные партиции `p1` и `p2` больше не существуют.

```
ALTER TABLE site_access REPLACE PARTITION (p1, p2) WITH TEMPORARY
PARTITION (tp1);
```

Примеры

Замените оригинальную формальную партицию `p1` временной партицией `tp1`.

```
ALTER TABLE site_access REPLACE PARTITION (p1) WITH TEMPORARY PARTITION
(tp1);
```

Замените оригинальные формальные партиции `p2` и `p3` временными партициями `tp2` и `tp3`.

```
ALTER TABLE site_access REPLACE PARTITION (p2, p3) WITH TEMPORARY PARTITION
(tp2, tp3);
```

Замените оригинальные формальные партиции `p4` и `p5` временными партициями `tp4` и `tp5`, и укажите параметры `strict_range` как `false` и `use_temp_partition_name` как `true`.

```
ALTER TABLE site_access REPLACE PARTITION (p4, p5) WITH TEMPORARY PARTITION
(tp4, tp5)
PROPERTIES (
    "strict_range" = "false",
    "use_temp_partition_name" = "true"
);
```

Примечания по использованию

- Когда таблица имеет временные партиции, вы не можете использовать команду `ALTER` для выполнения операций Schema Change в таблице.
- При выполнении операций Schema Change в таблице вы не можете добавлять временные партиции в таблицу.

Удаление временных партиций

Используйте следующую команду для удаления временной партиции `tp1`.

```
ALTER TABLE site_access DROP TEMPORARY PARTITION tp1;
```

Обратите внимание на следующие ограничения:

- Если вы используете команду `DROP` для прямого удаления базы данных или таблицы, вы можете восстановить базу данных или таблицу в течение ограниченного периода времени с помощью команды `RECOVER`. Однако временные партиции не могут быть восстановлены.
- После использования команды `ALTER` для удаления формальной партиции вы можете восстановить ее в течение ограниченного периода времени с помощью команды `RECOVER`. Временные партиции не связаны с формальными партициями, поэтому операции с временными партициями не влияют на формальные партиции.
- После использования команды `ALTER` для удаления временной партиции вы не можете восстановить ее с помощью команды `RECOVER`.
- При использовании команды `TRUNCATE` для удаления данных в таблице временные партиции таблицы удаляются и не могут быть восстановлены.
- При использовании команды `TRUNCATE` для удаления данных в формальной партиции временные партиции не затрагиваются.
- Команда TRUNCATE не может использоваться для удаления данных во временной партиции.

2.4.9 Поддержка функций: Распределение данных

Этот документ описывает функции партиционирования и bucketing, поддерживаемые Selena.

Поддерживаемые типы таблиц

- **Bucketing**

  Hash Bucketing поддерживается во всех типах таблиц. Random Bucketing (начиная с версии v1.5.2) поддерживается **только в таблицах Duplicate Key**.

- **Partitioning**

  Expression Partitioning (начиная с версии v1.5.2), Range Partitioning и List Partitioning (начиная с версии v1.5.2) поддерживаются во всех типах таблиц.

Bucketing

Функция	Ключевой момент	Статус поддержки	Примечание
Стратегия Bucketing	Hash Bucketing	Да	
	Random Bucketing	Да (v1.5.2+)	Random Bucketing поддерживается **только в таблицах Duplicate Key**. Начиная с версии v1.5.2, Selena поддерживает динамическую настройку количества tablets для создания в соответствии с информацией о cluster и размером данных.
Тип данных Bucket Key	Date, Integer, String	Да	
Количество Bucket	Автоматическая установка количества buckets	Да (v1.5.2+)	Автоматически определяется количеством узлов BE или объемом данных крупнейшего исторического partition. Логика была отдельно оптимизирована для партиционированных таблиц и непартиционированных таблиц в более поздних версиях.
	Динамическое увеличение количества Bucket для Random Bucketing	Да (v1.5.2+)	

Partitioning

Функция	Ключевой момент	Статус поддержки	Примечание
Стратегия Partitioning	Expression Partitioning	Да (v1.5.2+)	• Включая Partitioning на основе выражения временной функции (начиная с v1.5.2) и Partitioning на основе выражения столбца (начиная с v1.5.2) • Поддерживаемые временные функции: date_trunc, time_slice
	Range Partitioning	Да (v1.5.2+)	Начиная с версии v1.5.2, три специфические временные функции могут использоваться для Partition Keys: from_unixtime, from_unixtime_ms, str2date, substr/substring.
	List Partitioning	Да (v1.5.2+)	
Тип данных Partition Key	Date, Integer, Boolean	Да	
	String	Да	• Только Expression Partitioning и List Partitioning поддерживают Partition Key типа String. • Range Partitioning не поддерживает Partition Key типа String. Вам необходимо использовать str2date для преобразования столбца в типы date.

Различия между стратегиями партиционирования

	Expression Partitioning		Range Partitioning	List Partitioning
	Partitioning на основе выражения временной функции	Partitioning на основе выражения столбца		
Тип данных	Date (DATE/DATETIME)	• String (кроме BINARY) • Date (DATE/DATETIME) • Integer и Boolean	• String (кроме BINARY) [1] • Date или timestamp [1] • Integer	• String (кроме BINARY) • Date (DATE/DATETIME) • Integer и Boolean

Поддержка нескольких Partition Keys	/ (Поддерживает только один Partition Key типа date)	Да	Да	Да
Поддержка Null значений для Partition Keys	Да	/ [2]	Да	/ [2]
Ручное создание partitions перед загрузкой данных	/ [3]	/ [3]	• Да, если partitions создаются вручную пакетно • Нет, если принята стратегия динамического партиционирования	Да
Автоматическое создание partitions во время загрузки данных	Да	Да	/	/

ПРИМЕЧАНИЕ

- [1]: Вам необходимо использовать from_unixtime, str2date или другие временные функции для преобразования столбца в типы date.
- [2]: Null значения будут поддерживаться в Partition Keys для List Partitioning начиная с версии v1.5.2.
- [3]: Partitions создаются автоматически.

Для подробного сравнения между List Partitioning и Expression Partitioning обратитесь к Сравнение между list partitioning и expression partitioning.

2.5 Сжатие данных

Selena поддерживает сжатие данных для хранения таблиц и индексов. Сжатие данных не только помогает экономить место для хранения, но и повышает производительность задач с интенсивным вводом-выводом, поскольку Selena может считывать меньше страниц с диска для каждого запроса. Обратите внимание, что для сжатия и распаковки данных требуются дополнительные ресурсы процессора.

2.5.1 Выбор алгоритма сжатия данных

Selena поддерживает четыре алгоритма сжатия данных: LZ4, Zstandard (или zstd), zlib и Snappy. Эти алгоритмы сжатия данных отличаются коэффициентом сжатия и производительностью сжатия/распаковки. Обычно коэффициенты сжатия этих алгоритмов ранжируются следующим образом: zlib > Zstandard > LZ4 > Snappy. Среди них zlib показывает относительно высокие коэффициенты сжатия. В результате высокого сжатия данных производительность загрузки и запросов к таблицам с алгоритмом сжатия zlib также снижается. LZ4 и Zstandard, в частности, имеют хорошо сбалансированные коэффициенты сжатия и производительность распаковки. Вы можете выбрать один из этих алгоритмов сжатия в соответствии с потребностями вашего бизнеса в меньшем объеме хранения или лучшей производительности. Мы рекомендуем LZ4 или Zstandard, если у вас нет особых требований к экономии места для хранения.

> **ПРИМЕЧАНИЕ**
>
> Различные типы данных могут влиять на коэффициент сжатия.

2.5.2 Указание алгоритма сжатия данных для таблицы

Вы можете указать алгоритм сжатия данных для таблицы только при создании таблицы, и вы не сможете изменить его впоследствии.

Следующий пример создает таблицу `data_compression` с алгоритмом Zstandard. Подробные инструкции см. в разделе CREATE TABLE.

```
CREATE TABLE `data_compression` (
  `id`      INT(11)     NOT NULL     COMMENT "",
  `name`    CHAR(200)   NULL         COMMENT ""
)
ENGINE=OLAP
UNIQUE KEY(`id`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`id`)
PROPERTIES (
"compression" = "ZSTD"
);
```

> **ПРИМЕЧАНИЕ**

> Если алгоритм сжатия данных не указан, Selena использует LZ4 по умолчанию.

Вы можете просмотреть алгоритм сжатия таблицы с помощью SHOW CREATE TABLE.

2.6 Indexes

2.6.1 Индексы

Индексы помогают быстро находить данные, соответствующие определённым условиям запроса. В частности, когда индекс построен на таблице на основе некоторых её столбцов, запросы, использующие эти столбцы в качестве условий, требуют сканирования только части данных таблицы для поиска данных, удовлетворяющих условиям. Таким образом повышается эффективность запросов. Selena предоставляет различные типы индексов, которые в основном делятся на следующие две категории:

- Selena предоставляет встроенные индексы, которые автоматически создаются системой, включая Prefix индексы, Ordinal индексы и ZoneMap индексы.
- Selena также позволяет пользователям создавать индексы вручную, включая Bitmap индексы и Bloom filter индексы.

Встроенные индексы

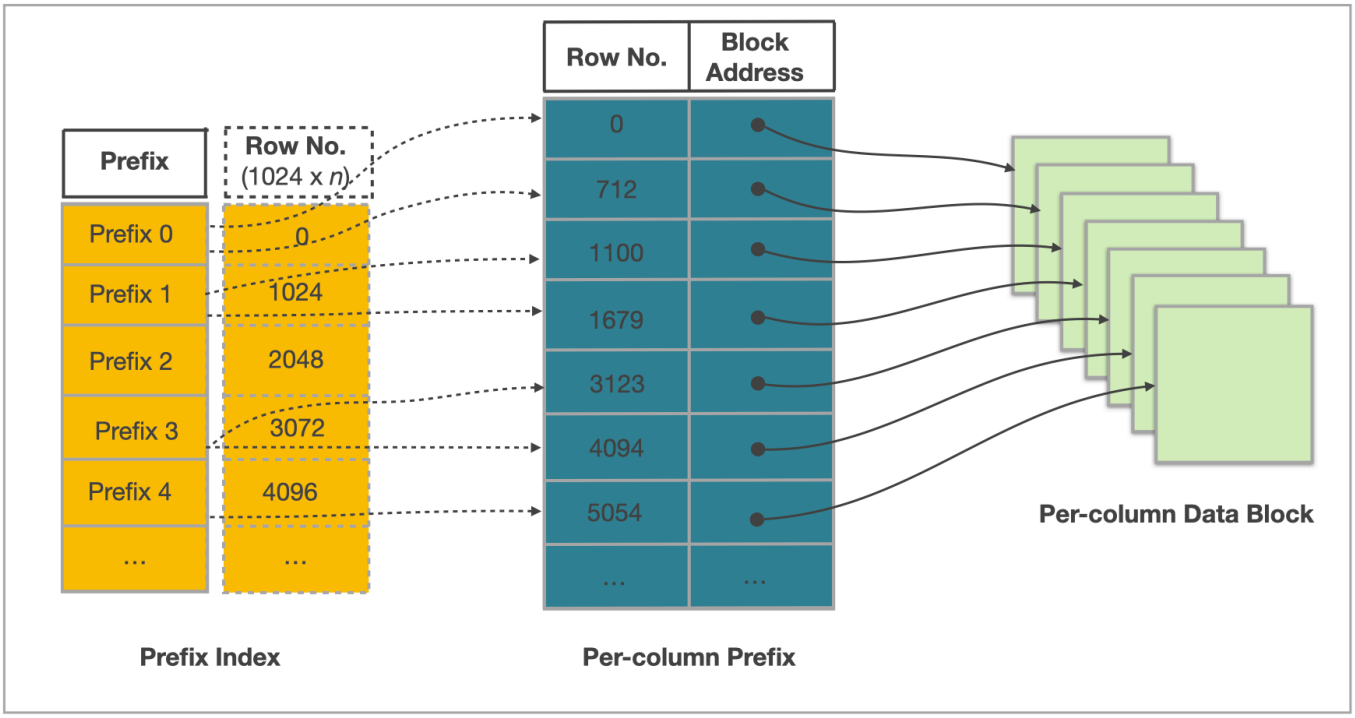
Prefix индексы

Prefix индекс автоматически генерируется во время записи данных. Точнее, когда данные записываются в Selena, они сортируются в соответствии с указанным ключом сортировки, и каждые 1024 строки данных включаются в один логический блок данных. Индексная запись, состоящая из значений столбцов ключа сортировки первой строки данных в этом логическом блоке, добавляется в таблицу Prefix индекса. Когда условие фильтрации в запросе совпадает с префиксом Prefix индекса, Prefix индекс может быстро найти данные, соответствующие условиям, и уменьшить объём сканируемых данных, что значительно улучшает производительность запросов.

Ordinal индексы

Selena фактически использует колоночное хранилище в базовом слое хранения. Данные каждого столбца хранятся на страницах данных, размер каждой страницы данных обычно составляет 64 * 1024 байта (data_page_size = 64 * 1024). Запись Ordinal индекса добавляется одновременно с генерацией страницы данных. Запись Ordinal индекса содержит информацию, такую как начальный номер строки страницы данных. Таким

образом, Ordinal индекс может определить физический адрес страницы данных столбца, используя номер строки.



ZoneMap индексы

ZoneMap индекс хранит статистику каждого блока данных, включающую Min (минимальное значение), Max (максимальное значение), HasNull (наличие null значений) и HasNotNull (наличие не-null значений). Во время запросов Selena может быстро оценить, можно ли отфильтровать эти блоки данных на основе их статистики, уменьшая объём сканируемых данных и повышая скорость запросов.

▸ Дополнительная информация

Индексы, созданные вручную

Если столбец в условии запроса не является префиксным полем, вы можете вручную создать индекс для этого столбца на основе характеристик данных и запросов для повышения эффективности запросов.

Bitmap индексы

Bitmap индекс подходит для запросов по столбцам с высокой кардинальностью или комбинациям нескольких столбцов с низкой кардинальностью. И bitmap индекс может

демонстрировать идеальную производительность фильтрации для таких запросов, отфильтровывая как минимум 999 из 1000 строк.

Bloom filter индексы

Bloom filter индекс подходит для столбцов с относительно высокой кардинальностью, таких как столбцы ID, но может иметь определённый уровень ложных срабатываний.

N-Gram bloom filter индексы

N-gram bloom filter индекс — это специальный тип Bloom filter индекса, обычно используемый для ускорения запросов `LIKE` или операций функций `ngram_search` и `ngram_search_case_insensitive`.

Полнотекстовые inverted индексы

Полнотекстовый inverted индекс может быстро находить строки данных, соответствующие ключевым словам, тем самым ускоряя полнотекстовый поиск.

Vector индексы

Vector индекс позволяет выполнять приблизительный поиск ближайших соседей (ANNS) с помощью Selena.

2.6.2 Prefix индексы

Укажите одну или несколько колонок для формирования sort key при создании таблицы. Строки данных в таблице будут отсортированы на основе sort key и затем сохранены на диске.

Во время записи данных Prefix индекс создается автоматически. После сортировки данных в соответствии с указанным sort key каждые 1024 строки данных включаются в один логический блок данных. Запись индекса, состоящая из значений колонок sort key первой строки данных в этом логическом блоке данных, добавляется в таблицу Prefix индекса.

С этими двумя уровнями структур сортировки запросы могут использовать бинарный поиск для быстрого пропуска данных, которые не соответствуют условиям запроса, а также могут избежать дополнительных операций сортировки во время запросов.

ПОДСКАЗКА

Prefix индекс является разреженным индексом, и его размер как минимум в 1024 раза меньше объема данных. Поэтому он обычно может быть полностью кэширован в памяти для ускорения производительности запросов.

Примечания по использованию

Начиная с версии 3.0, таблицы Primary Key поддерживают определение sort keys с использованием `ORDER BY`. Начиная с версии 3.3, таблицы Duplicate Key, таблицы Aggregate и таблицы Unique Key поддерживают определение sort keys с использованием `ORDER BY`.

- Данные в таблице Duplicate Key сортируются в соответствии с sort key `ORDER BY`. Sort key может быть комбинацией любых колонок.

  К СВЕДЕНИЮ

  Когда указаны и `ORDER BY`, и `DUPLICATE KEY`, `DUPLICATE KEY` не вступает в силу.

- Данные в таблице Aggregate сначала агрегируются в соответствии с aggregate key `AGGREGATE KEY`, затем сортируются в соответствии с sort key `ORDER BY`. Колонки в `ORDER BY` и `AGGREGATE KEY` должны быть одинаковыми, но порядок колонок не обязательно должен быть одинаковым.

- Данные в таблице Unique Key сначала заменяются в соответствии с unique key `UNIQUE KEY`, затем сортируются в соответствии с sort key `ORDER BY`. Колонки в `ORDER BY` и `UNIQUE KEY` должны быть одинаковыми, но порядок колонок не обязательно должен быть одинаковым.

- Данные в таблице Primary Key сначала заменяются в соответствии с primary key `PRIMARY KEY`, затем сортируются в соответствии с sort key `ORDER BY`.

Возьмем в качестве примера таблицу Duplicate Key. Sort key определяется как `uid` и `name` с использованием `ORDER BY`.

```
CREATE TABLE user_access (
    uid int,
    name varchar(64),
    age int,
    phone varchar(16),
    last_access datetime,
    credits double
)
ORDER BY (uid, name);
```

ПОДСКАЗКА

После создания таблицы вы можете использовать `SHOW CREATE TABLE <table_name>;`, чтобы просмотреть указанные колонки сортировки и порядок этих колонок в предложении `ORDER BY` из возвращенного результата.

Поскольку максимальная длина записи Prefix индекса составляет 36 байт, превышающая часть будет обрезана. Поэтому каждая запись в Prefix индексе этой таблицы - это uid (4 байта) + name (берутся только первые 32 байта), и prefix поля - это `uid` и `name`.

Примечание

- Количество prefix полей не может превышать 3, а максимальная длина записи Prefix индекса составляет 36 байт.
- В пределах prefix полей колонки типа CHAR, VARCHAR или STRING могут встречаться только один раз и должны находиться в конце.

  Возьмем в качестве примера следующую таблицу, где первые три колонки являются колонками sort key. Prefix поле этой таблицы - `name` (20 байт). Это связано с тем, что этот Prefix индекс начинается с колонки типа VARCHAR (`name`) и обрезается напрямую без включения дальнейших колонок, даже если запись Prefix индекса не достигает 36 байт в длину. Поэтому этот Prefix индекс содержит только поле `name`.

```
MySQL [example_db]> describe user_access2;
+-------------+-------------+------+-------+---------+-------+
| Field       | Type        | Null | Key   | Default | Extra |
+-------------+-------------+------+-------+---------+-------+
| name        | varchar(20) | YES  | true  | NULL    |       |
| uid         | int         | YES  | true  | NULL    |       |
| last_access | datetime    | YES  | true  | NULL    |       |
| age         | int         | YES  | false | NULL    |       |
| phone       | varchar(16) | YES  | false | NULL    |       |
| credits     | double      | YES  | false | NULL    |       |
+-------------+-------------+------+-------+---------+-------+
6 rows in set (0.00 sec)
```

- Если sort key указан в таблице с использованием `ORDER BY`, Prefix индекс формируется на основе sort key. Если sort key не указан с использованием `ORDER BY`, Prefix индекс формируется на основе Key колонок.

Как правильно спроектировать sort key для формирования Prefix индекса, который может ускорить запросы

Анализ запросов и данных в бизнес-сценариях помогает выбрать соответствующие колонки sort key и расположить их в правильном порядке для формирования Prefix индекса, что может значительно улучшить производительность запросов.

- Количество колонок sort key обычно составляет 3 и не рекомендуется превышать 4. Sort key со слишком большим количеством колонок не может улучшить производительность запросов, но увеличивает накладные расходы на сортировку при загрузке данных.
- Рекомендуется приоритезировать колонки для формирования sort key в порядке, указанном ниже:
    i. **Выберите колонки, которые часто используются в условиях фильтрации запросов, в качестве колонок sort key.** Если количество колонок sort key больше одной, расположите их в порядке убывания их частоты в условиях фильтрации запросов. Таким образом, если условия фильтрации запросов включают prefix Prefix индекса, производительность запроса может быть значительно улучшена. И если условия фильтрации включают весь prefix Prefix индекса, запрос может полностью использовать Prefix индекс. Конечно, если условия фильтрации включают prefix, хотя и не весь prefix, Prefix индекс все еще может оптимизировать запрос. Однако эффект Prefix индекса будет ослаблен,

если длина prefix, включенного в условия фильтрации, слишком мала. Опять же, возьмем таблицу Unique Key, чей sort key - `(uid,name)`, в качестве примера. Если условия фильтрации запроса включают весь prefix, например `select sum(credits) from user_access where uid = 123 and name = 'Jane Smith';`, запрос может полностью использовать Prefix индекс для повышения производительности. Если условия запроса включают только часть prefix, например `select sum(credits) from user_access where uid = 123;`, запрос также может воспользоваться Prefix индексом для повышения производительности. Однако если условия запроса не включают prefix, например `select sum(credits) from user_access where name = 'Jane Smith';`, запрос не может использовать Prefix индекс для ускорения.

- Если несколько колонок sort key имеют схожую частоту в качестве условий фильтрации запросов, вы можете измерить кардинальность этих колонок.

- Если кардинальность колонки высока, она может отфильтровать больше данных во время запроса. Если кардинальность слишком низкая, например для колонок булева типа, ее эффект фильтрации не идеален.

> **ПОДСКАЗКА**
>
> Однако, учитывая характеристики запросов в реальных бизнес-сценариях, обычно колонки с немного более низкой кардинальностью чаще используются в качестве условий запроса, чем колонки с высокой кардинальностью. Это связано с тем, что запросы, чья фильтрация часто основана на колонках с высокой кардинальностью, или даже в некоторых экстремальных сценариях, основана на колонках с ограничениями UNIQUE, больше похожи на точечные запросы в OLTP базах данных, а не на сложные аналитические запросы в OLAP базах данных.

- Также учитывайте факторы сжатия хранилища. Если разница в производительности запросов между порядком колонки с низкой кардинальностью и колонкой с высокой кардинальностью не очевидна, размещение колонки с низкой кардинальностью перед колонкой с высокой кардинальностью приведет к гораздо более высокой степени сжатия хранилища для отсортированной колонки с низкой кардинальностью. Поэтому рекомендуется размещать колонку с низкой кардинальностью впереди.

Соображения по определению колонок sort key при создании таблицы

- Типы данных колонок сортировки:
  - Колонки сортировки в таблицах Primary Key поддерживают числовые типы (включая целые числа, булевы значения), строки и типы date/datetime.
  - Колонки сортировки в таблицах Duplicate Key, таблицах Aggregate и таблицах Unique Key поддерживают числовые типы (включая целые числа, булевы значения, десятичные числа), строки и типы date/datetime.
- В таблицах Aggregate и таблицах Unique Key колонки сортировки должны быть определены перед другими колонками.

Можно ли изменить Prefix индекс?

Если характеристики запросов в бизнес-сценарии меняются, и колонки, отличные от prefix полей, часто используются в условиях фильтрации запросов, существующий Prefix индекс не может фильтровать данные, и производительность запросов может быть не идеальной.

Начиная с версии 3.0, sort keys таблиц Primary Key могут быть изменены. И начиная с версии 3.3, sort keys таблиц Duplicate Key, таблиц Aggregate и таблиц Unique Key могут быть изменены. Sort keys в таблицах Duplicate Key и таблицах Primary Key могут быть комбинацией любых колонок сортировки. Sort keys в таблицах Aggregate и таблицах Unique Key должны включать все key колонки, но порядок этих колонок не обязательно должен соответствовать key колонкам.

В качестве альтернативы, вы также можете создать синхронные материализованные представления на основе этой таблицы и выбрать другие колонки, часто используемые в качестве условных колонок, для формирования Prefix индекса, что может улучшить производительность для этих запросов. Но обратите внимание, что это увеличит место для хранения.

Как проверить, ускоряет ли Prefix индекс запросы

После выполнения запроса вы можете проверить, вступает ли в силу Prefix индекс, и просмотреть его эффект фильтрации по детальным метрикам, таким как `ShortKeyFilterRows`, в узле сканирования в Query Profile.

2.6.3 Bitmap индексы

В этой теме описывается, как создавать и управлять bitmap индексом, а также варианты использования.

Введение

Bitmap индекс - это специальный индекс базы данных, который использует bitmap, который представляет собой массив битов. Бит всегда находится в одном из двух значений: 0 и 1. Каждый бит в bitmap соответствует одной строке в таблице. Значение каждого бита зависит от значения соответствующей строки.

Bitmap индекс может помочь улучшить производительность запросов по заданному столбцу. Если условия фильтра запроса соответствуют prefix индексу, это может значительно улучшить эффективность запроса и быстро вернуть результаты. Однако таблица может иметь только один prefix индекс. Если условия фильтра запроса не включают prefix prefix индекса, bitmap индекс может быть создан для этого столбца для улучшения эффективности запроса.

Как разработать bitmap индекс для ускорения запросов

Основные соображения при выборе bitmap индекса - это **cardinality столбца и эффект фильтрации bitmap индекса на запросы**. Вопреки распространенному мнению, bitmap индексы в Selena больше подходят для **запросов к столбцам с высокой cardinality и запросов к комбинации нескольких столбцов с низкой cardinality. Кроме того, bitmap индексы должны эффективно фильтровать данные, потенциально отфильтровывая по крайней мере 999/1000 данных, тем самым уменьшая объем данных Page, которые необходимо прочитать.**

Для запросов к одному столбцу с низкой cardinality эффект фильтрации bitmap индекса плохой, поскольку необходимо прочитать слишком много строк, разбросанных по нескольким Pages.

> **К СВЕДЕНИЮ**
>
> Вам нужно учитывать стоимость загрузки данных при оценке эффекта фильтрации bitmap индексов на запросы. В Selena базовые данные организованы и загружаются по Pages (размер по умолчанию 64K). Стоимость загрузки данных включает время загрузки Pages с диска, распаковки Pages и декодирования.

Однако чрезмерно высокая cardinality также может вызвать проблемы, такие как **занятие большего дискового пространства**, и поскольку bitmap индексы должны создаваться во время загрузки данных, частая загрузка данных может **влиять на производительность загрузки**.

Кроме того, следует учитывать **накладные расходы на загрузку bitmap индексов во время запросов**. Во время запроса bitmap индексы загружаются по требованию, и чем больше значение `количество значений столбца, участвующих в условиях запроса/cardinality x bitmap индекс`, тем больше накладные расходы на загрузку bitmap индексов во время запросов.

Чтобы определить подходящую cardinality и условия запроса для bitmap индексов, рекомендуется обратиться к Тесту производительности bitmap индекса в этой теме для проведения тестов производительности. Вы можете использовать фактические бизнес-данные и запросы для **создания bitmap индексов на столбцах с различной cardinality, чтобы проанализировать эффект фильтрации bitmap индексов на запросы (по крайней мере, отфильтровывая 999/1000 данных), использование дискового пространства, влияние на производительность загрузки и накладные расходы на загрузку bitmap индексов во время запросов.**

Selena имеет встроенный механизм адаптивного выбора bitmap индексов. Если bitmap индекс не может ускорить запросы, например, если он не может отфильтровать много Pages, или накладные расходы на загрузку bitmap индексов во время запросов высоки, он не будет использоваться во время запроса, поэтому производительность запроса не будет существенно затронута.

Адаптивный выбор bitmap индексов

Selena может адаптивно выбирать, использовать ли bitmap индекс на основе cardinality столбца и условий запроса. Если bitmap индекс не эффективно фильтрует много Pages или накладные расходы на загрузку bitmap индексов во время запросов высоки, Selena не будет использовать bitmap индекс по умолчанию, чтобы избежать ухудшения производительности запроса.

Selena определяет, использовать ли bitmap индекс, на основе отношения количества значений, участвующих в условии запроса, к cardinality столбца. Как правило, чем меньше это отношение, тем лучше эффект фильтрации bitmap индекса. Таким образом, Selena использует `bitmap_max_filter_ratio/1000` в качестве порога. Когда `количество`

`значений в условии фильтра/cardinality столбца` меньше `bitmap_max_filter_ratio/1000`, bitmap индекс будет использоваться. Значение по умолчанию для `bitmap_max_filter_ratio` равно `1`.

Возьмем в качестве примера запрос на основе одного столбца, такой как `SELECT * FROM employees WHERE gender = 'male';`. Столбец `gender` в таблице `employees` имеет значения 'male' и 'female', поэтому cardinality равна 2 (два различных значения). Условие запроса включает одно значение, поэтому отношение составляет 1/2, что больше 1/1000. Следовательно, этот запрос не будет использовать bitmap индекс.

Возьмем другой запрос на основе комбинации нескольких столбцов в качестве примера, такой как `SELECT * FROM employees WHERE gender = 'male' AND city IN ('Beijing', 'Shanghai');`. Cardinality столбца `city` составляет 10 000, и условие запроса включает два значения, поэтому отношение рассчитывается как `(1*2)/(2*10000)`, что меньше 1/1000. Следовательно, этот запрос будет использовать bitmap индекс.

> **К СВЕДЕНИЮ**
>
> Диапазон значений для `bitmap_max_filter_ratio` составляет 1-1000. Если `bitmap_max_filter_ratio` установлено в `1000`, любой запрос к столбцу с bitmap индексом будет принудительно использовать bitmap индекс.

Преимущества

- Bitmap индексы могут быстро определить номера строк запрошенных значений столбцов, подходят для точечных запросов или запросов с небольшим диапазоном.
- Bitmap индексы могут оптимизировать многомерные запросы, включающие операции объединения и пересечения (операции `OR` и `AND`).

Рекомендации

Запросы, которые могут быть оптимизированы

Bitmap индексы подходят для оптимизации запросов на равенство `=`, запросов диапазона `[NOT] IN`, запросов `>`, `>=`, `<`, `<=` и запросов `IS NULL`. Они не подходят для оптимизации запросов `!=` и `[NOT] LIKE`.

Поддерживаемые столбцы и типы данных

Bitmap индексы могут быть созданы на всех столбцах в primary key и duplicate key таблицах, и на ключевых столбцах в aggregate и unique key таблицах. Bitmap индексы могут быть созданы на столбцах следующих типов данных:

- Типы дат: DATE, DATETIME.
- Числовые типы: TINYINT, SMALLINT, INT, BIGINT, LARGEINT, DECIMAL, BOOLEAN.
- Строковые типы: CHAR, STRING, VARCHAR.
- Другие типы: HLL.

Базовые операции

Создание индекса

- Создание bitmap индекса во время создания таблицы.

```
CREATE TABLE `lineorder_partial` (
  `lo_orderkey` int(11) NOT NULL COMMENT "",
  `lo_orderdate` int(11) NOT NULL COMMENT "",
  `lo_orderpriority` varchar(16) NOT NULL COMMENT "",
  `lo_quantity` int(11) NOT NULL COMMENT "",
  `lo_revenue` int(11) NOT NULL COMMENT "",
   INDEX lo_orderdate_index (lo_orderdate) USING BITMAP
) ENGINE=OLAP
DUPLICATE KEY(`lo_orderkey`)
DISTRIBUTED BY HASH(`lo_orderkey`) BUCKETS 1;
```

В этом примере bitmap индекс с именем `lo_orderdate_index` создается на столбце `lo_orderdate`. Требования к именованию bitmap индексов можно найти в Системные ограничения. Идентичные bitmap индексы не могут быть созданы в одной таблице.

Несколько bitmap индексов могут быть созданы для нескольких столбцов, разделенных запятыми (,).

> **ПРИМЕЧАНИЕ**
>
> Для дополнительных параметров создания таблицы обратитесь к CREATE TABLE.

- `CREATE INDEX` может использоваться для создания bitmap индекса после создания таблицы. Для подробных описаний параметров и примеров обратитесь к CREATE INDEX.

```
CREATE INDEX lo_quantity_index ON lineorder_partial (lo_quantity) USING
BITMAP;
```

Прогресс создания индекса

Создание bitmap индекса - это асинхронный процесс. После выполнения оператора создания индекса вы можете проверить прогресс создания индекса с помощью команды SHOW ALTER TABLE. Когда поле `State` в возвращаемом значении показывает `FINISHED`, индекс успешно создан.

```
SHOW ALTER TABLE COLUMN;
```

> **К СВЕДЕНИЮ**
>
> Каждая таблица может иметь только одну выполняющуюся задачу Schema Change одновременно. Вы не можете создать новый bitmap индекс, пока текущий bitmap индекс не будет создан.

Просмотр индекса

Просмотр всех bitmap индексов для указанной таблицы. Для подробных параметров и возвращаемых результатов обратитесь к SHOW INDEX.

```
SHOW INDEXES FROM lineorder_partial;
```

> **ПРИМЕЧАНИЕ**
>
> Создание bitmap индекса - это асинхронный процесс. Используя приведенный выше оператор, вы можете просматривать только индексы, которые успешно завершили создание.

Удаление индекса

Удаление bitmap индекса для указанной таблицы. Для подробных параметров и примеров обратитесь к DROP INDEX.

```
DROP INDEX lo_orderdate_index ON lineorder_partial;
```

Проверка того, ускоряет ли bitmap индекс запросы

Проверьте поле `BitmapIndexFilterRows` в Profile запроса. Для информации о просмотре Profile обратитесь к Анализ запросов.

Тест производительности bitmap индекса

Цель тестирования

Анализ эффекта фильтрации и других влияний, таких как использование диска, bitmap индексов на запросы с различной cardinality:

- Запрос к одному столбцу с низкой cardinality
- Запрос к комбинации нескольких столбцов с низкой cardinality
- Запрос к одному столбцу с высокой cardinality

Этот раздел также сравнивает производительность между всегда использующим bitmap индекс и адаптивным использованием bitmap индексов для проверки эффективности адаптивного выбора bitmap индексов Selena.

Создание таблицы и bitmap индекса

> **WARNING**
>
> Чтобы избежать кэширования данных Page, влияющего на производительность запроса, убедитесь, что конфигурационный параметр BE `disable_storage` установлен в `true`.

В этом разделе в качестве примера используется таблица `lineorder` (SSB 20G).

- Исходная таблица (без bitmap индексов) в качестве эталона

```
CREATE TABLE `lineorder_without_index` (
`lo_orderkey` int(11) NOT NULL COMMENT "",
`lo_linenumber` int(11) NOT NULL COMMENT "",
`lo_custkey` int(11) NOT NULL COMMENT "",
`lo_partkey` int(11) NOT NULL COMMENT "",
`lo_suppkey` int(11) NOT NULL COMMENT "",
`lo_orderdate` int(11) NOT NULL COMMENT "",
`lo_orderpriority` varchar(16) NOT NULL COMMENT "",
`lo_shippriority` int(11) NOT NULL COMMENT "",
`lo_quantity` int(11) NOT NULL COMMENT "",
`lo_extendedprice` int(11) NOT NULL COMMENT "",
`lo_ordtotalprice` int(11) NOT NULL COMMENT "",
`lo_discount` int(11) NOT NULL COMMENT "",
`lo_revenue` int(11) NOT NULL COMMENT "",
`lo_supplycost` int(11) NOT NULL COMMENT "",
`lo_tax` int(11) NOT NULL COMMENT "",
`lo_commitdate` int(11) NOT NULL COMMENT "",
`lo_shipmode` varchar(11) NOT NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`lo_orderkey`)
DISTRIBUTED BY HASH(`lo_orderkey`) BUCKETS 1;
```

- Таблица с bitmap индексами: Bitmap индексы созданы на основе `lo_shipmode`, `lo_quantity`, `lo_discount`, `lo_orderdate`, `lo_tax` и `lo_partkey`.

```
CREATE TABLE `lineorder_with_index` (
  `lo_orderkey` int(11) NOT NULL COMMENT "",
  `lo_linenumber` int(11) NOT NULL COMMENT "",
  `lo_custkey` int(11) NOT NULL COMMENT "",
  `lo_partkey` int(11) NOT NULL COMMENT "",
  `lo_suppkey` int(11) NOT NULL COMMENT "",
  `lo_orderdate` int(11) NOT NULL COMMENT "",
  `lo_orderpriority` varchar(16) NOT NULL COMMENT "",
  `lo_shippriority` int(11) NOT NULL COMMENT "",
  `lo_quantity` int(11) NOT NULL COMMENT "",
  `lo_extendedprice` int(11) NOT NULL COMMENT "",
  `lo_ordtotalprice` int(11) NOT NULL COMMENT "",
  `lo_discount` int(11) NOT NULL COMMENT "",
  `lo_revenue` int(11) NOT NULL COMMENT "",
  `lo_supplycost` int(11) NOT NULL COMMENT "",
  `lo_tax` int(11) NOT NULL COMMENT "",
  `lo_commitdate` int(11) NOT NULL COMMENT "",
  `lo_shipmode` varchar(11) NOT NULL COMMENT "",
  INDEX i_shipmode (`lo_shipmode`) USING BITMAP,
  INDEX i_quantity (`lo_quantity`) USING BITMAP,
  INDEX i_discount (`lo_discount`) USING BITMAP,
  INDEX i_orderdate (`lo_orderdate`) USING BITMAP,
  INDEX i_tax (`lo_tax`) USING BITMAP,
  INDEX i_partkey (`lo_partkey`) USING BITMAP
) ENGINE=OLAP
DUPLICATE KEY(`lo_orderkey`)
DISTRIBUTED BY HASH(`lo_orderkey`) BUCKETS 1;
```

Использование дискового пространства bitmap индексами

- `lo_shipmode`: Строковый тип, cardinality 7, занимает 130M
- `lo_quantity`: Целочисленный тип, cardinality 50, занимает 291M
- `lo_discount`: Целочисленный тип, cardinality 11, занимает 198M
- `lo_orderdate`: Целочисленный тип, cardinality 2406, занимает 191M
- `lo_tax`: Целочисленный тип, cardinality 9, занимает 160M
- `lo_partkey`: Целочисленный тип, cardinality 600 000, занимает 601M

Запрос 1: Запрос к одному столбцу с низкой cardinality

Запрос к таблице без bitmap индекса

Запрос:

```
SELECT count(1) FROM lineorder_without_index WHERE lo_shipmode="MAIL";
```

Анализ производительности запроса: Поскольку запрашиваемая таблица не имеет bitmap индекса, все pages, содержащие данные столбца `lo_shipmode`, должны быть прочитаны, а затем применяется фильтрация предикатов.

Общее время: Приблизительно 0,91 секунды, **из которых загрузка данных заняла 0,47 секунды**, декодирование словаря для оптимизации низкой cardinality заняло 0,31 секунды, а фильтрация предикатов заняла 0,23 секунды.

```
PullRowNum: 20.566M (20566493) // Количество строк в результирующем наборе.
CompressedBytesRead: 55.283 MB // Общий объем прочитанных данных.
RawRowsRead: 143.999M (143999468) // Количество прочитанных строк.
Поскольку bitmap индекса нет, все данные в этом столбце читаются.
ReadPagesNum: 8.795K (8795) // Количество прочитанных pages. Поскольку
bitmap индекса нет, все pages, содержащие данные для этого столбца,
читаются.
IOTaskExecTime: 914ms // Общее время сканирования данных.
    BlockFetch: 469ms // Время загрузки данных.
    DictDecode: 311.612ms // Время декодирования словаря для оптимизации
низкой cardinality.
    PredFilter: 23.182ms // Время фильтрации предикатов.
    PredFilterRows: 123.433M (123432975) // Количество отфильтрованных
строк.
```

Запрос к таблице с bitmap индексом

Принудительное использование bitmap индекса

> **К СВЕДЕНИЮ**
>
> Для принудительного использования bitmap индекса, согласно конфигурации Selena, необходимо установить `bitmap_max_filter_ratio=1000` в файле **be.conf** каждого узла BE, а затем перезапустить узлы BE.

Запрос:

```
SELECT count(1) FROM lineorder_with_index WHERE lo_shipmode="MAIL";
```

Анализ производительности запроса: Поскольку запрашиваемый столбец имеет низкую cardinality, bitmap индекс не фильтрует данные эффективно. Хотя bitmap индекс может быстро определить номера строк фактических данных, необходимо прочитать большое количество строк, разбросанных по нескольким pages. В результате он не может эффективно отфильтровать pages, которые необходимо прочитать. Более того, возникают дополнительные накладные расходы на загрузку bitmap индекса и использование bitmap индекса для фильтрации данных, что приводит к более длительному общему времени.

Общее время: 2,077 секунды, **из которых 0,93 секунды потрачено на загрузку данных и bitmap индекса**, 0,33 секунды на декодирование словаря для оптимизации низкой cardinality, 0,42 секунды на фильтрацию данных с помощью bitmap индекса и 0,17 секунды на фильтрацию данных с помощью ZoneMap Index.

```
PullRowNum: 20.566M (20566493) // Количество строк в результирующем наборе.
CompressedBytesRead: 72.472 MB // Общий объем прочитанных данных. Размер
bitmap индекса для этого столбца составляет 130MB, с 7 уникальными
значениями. Размер bitmap индекса для каждого значения составляет 18MB.
Размер данных Pages составляет 55MB = 73MB.
RawRowsRead: 20.566M (20566493) // Количество прочитанных строк. Фактически
прочитано 20 миллионов строк.
ReadPagesNum: 8.802K (8802) // Количество прочитанных Pages. Все Pages
читаются, поскольку 20 миллионов строк, отфильтрованных bitmap индексом,
случайно распределены по разным Pages, а Page является наименьшей единицей
чтения данных. Следовательно, bitmap индекс не фильтрует Pages.
IOTaskExecTime: 2s77ms // Общее время сканирования данных, больше, чем без
bitmap индекса.
    BlockFetch: 931.144ms // Время загрузки данных и bitmap индекса. По
сравнению с предыдущим запросом дополнительные 400ms были потрачены на
загрузку bitmap индекса (18MB).
    DictDecode: 329.696ms // Время декодирования словаря для оптимизации
низкой cardinality аналогично, поскольку количество выходных строк
одинаковое.
    BitmapIndexFilter: 419.308ms // Время фильтрации данных с помощью
bitmap индекса.
    BitmapIndexFilterRows: 123.433M (123432975) // Количество строк,
отфильтрованных bitmap индексом.
    ZoneMapIndexFiter: 171.580ms // Время фильтрации данных с помощью
ZoneMap Index.
```

Определение использования bitmap индекса на основе конфигурации по умолчанию Selena

Запрос:

```
SELECT count(1) FROM lineorder_with_index WHERE lo_shipmode="MAIL";
```

Анализ производительности запроса: Согласно конфигурации по умолчанию Selena, bitmap индекс используется только в том случае, если количество различных значений в столбце условия фильтра / cardinality столбца < `bitmap_max_filter_ratio/1000` (по умолчанию 1/1000). В этом случае значение больше 1/1000, поэтому bitmap индекс не используется, что приводит к производительности, аналогичной запросу к таблице без bitmap индекса.

```
PullRowNum: 20.566M (20566493) // Количество строк в результирующем наборе.
CompressedBytesRead: 55.283 MB // Общий объем прочитанных данных.
RawRowsRead: 143.999M (143999468) // Количество прочитанных строк.
Поскольку bitmap индекс не используется, все данные в этом столбце
читаются.
ReadPagesNum: 8.800K (8800) // Количество прочитанных Pages. Поскольку
bitmap индекс не используется, все pages, содержащие данные для этого
столбца, читаются.
IOTaskExecTime: 914.279ms // Общее время сканирования данных.
    BlockFetch: 475.890ms // Время загрузки данных.
    DictDecode: 312.019ms // Время декодирования словаря для оптимизации
низкой cardinality.
    PredFilterRows: 123.433M (123432975) // Количество строк,
отфильтрованных предикатом.
```

Запрос 2: Запрос к комбинации нескольких столбцов с низкой cardinality

Запрос к таблице без bitmap индекса

Запрос:

```
SELECT count(1)
FROM lineorder_without_index
WHERE lo_shipmode = "MAIL"
  AND lo_quantity = 10
  AND lo_discount = 9
  AND lo_tax = 8;
```

Анализ производительности запроса: Поскольку запрашиваемая таблица не имеет bitmap индекса, все Pages, содержащие данные столбцов `lo_shipmode`, `lo_quantity`, `lo_discount` и `lo_tax`, читаются, а затем применяется фильтрация предикатов.

Общее затраченное время: 1,76 секунды, **из которых 1,6 секунды потрачено на загрузку данных (4 столбца)**, и 0,1 секунды на фильтрацию предикатов.

```
PullRowNum: 4.092K (4092) // Количество строк в результирующем наборе.
CompressedBytesRead: 305.346 MB // Общий объем прочитанных данных. Данные 4
столбцов читаются, всего 305MB.
RawRowsRead: 143.999M (143999468) // Количество прочитанных строк.
Поскольку bitmap индекса нет, все данные этих 4 столбцов читаются.
ReadPagesNum: 35.168K (35168) // Количество прочитанных Pages. Поскольку
bitmap индекса нет, все Pages, содержащие данные этих 4 столбцов, читаются.
IOTaskExecTime: 1s761ms // Общее время сканирования данных.
    BlockFetch: 1s639ms // Время загрузки данных.
    PredFilter: 96.533ms // Время фильтрации предикатов с 4 условиями
фильтрации.
    PredFilterRows: 143.995M (143995376) // Количество строк,
отфильтрованных предикатами.
```

Запрос к таблице с bitmap индексом

Принудительное использование bitmap индекса

> **К СВЕДЕНИЮ**
>
> Для принудительного использования bitmap индекса, согласно конфигурации Selena, необходимо установить `bitmap_max_filter_ratio=1000` в файле **be.conf** каждого узла BE, затем узлы BE должны быть перезапущены.

Запрос:

```
SELECT count(1) FROM lineorder_with_index WHERE lo_shipmode="MAIL" AND
lo_quantity=10 AND lo_discount=9 AND lo_tax=8;
```

Анализ производительности запроса: Поскольку это комбинированный запрос на основе нескольких столбцов с низкой cardinality, bitmap индекс эффективен, отфильтровывая часть Pages и значительно сокращая время чтения данных.

Общее затраченное время: 0,68 секунды, **из которых 0,54 секунды потрачено на загрузку данных и bitmap индекса**, и 0,14 секунды на фильтрацию данных с помощью bitmap индекса.

```
PullRowNum: 4.092K (4092) // Количество строк в результирующем наборе.
CompressedBytesRead: 156.340 MB // Общий объем прочитанных данных. Bitmap
индекс эффективно фильтрует 2/3 данных. Из этих 156MB 60MB - это данные
индекса, а 90MB - фактические данные.
ReadPagesNum: 11.325K (11325) // Количество прочитанных Pages. Bitmap
индекс эффективно фильтрует 2/3 Pages.
IOTaskExecTime: 683.471ms // Общее время сканирования данных, значительно
меньше общего времени без bitmap индекса.
    BlockFetch: 537.421ms // Время загрузки данных и bitmap индекса. Хотя
дополнительное время тратится на загрузку bitmap индекса, это значительно
сокращает время загрузки данных.
    BitmapIndexFilter: 139.024ms // Время фильтрации данных с помощью
bitmap индекса.
    BitmapIndexFilterRows: 143.995M (143995376) // Количество строк,
отфильтрованных bitmap индексом.
```

Определение использования bitmap индекса на основе конфигурации по умолчанию Selena

Запрос:

```
SELECT count(1) FROM lineorder_with_index WHERE lo_shipmode="MAIL" AND
lo_quantity=10 AND lo_discount=9 AND lo_tax=8;
```

Анализ производительности запроса: Согласно конфигурации по умолчанию Selena, bitmap индекс используется, если количество различных значений в столбцах условия фильтра / cardinality столбца < `bitmap_max_filter_ratio/1000` (по умолчанию 1/1000). В

этом случае значение меньше 1/1000, поэтому bitmap индекс используется, что приводит к производительности, аналогичной принудительному использованию bitmap индекса.

Общее затраченное время: 0,67 секунды, **из которых 0,54 секунды потрачено на загрузку данных и bitmap индекса**, и 0,13 секунды на фильтрацию данных с помощью bitmap индекса.

```
PullRowNum: 4.092K (4092) // Количество строк в результирующем наборе.
CompressedBytesRead: 154.430 MB // Общий объем прочитанных данных.
ReadPagesNum: 11.209K (11209) // Количество прочитанных Pages.
IOTaskExecTime: 672.029ms // Общее время сканирования данных, значительно
меньше, чем без bitmap индекса.
    BlockFetch: 535.823ms // Время загрузки данных и bitmap индекса. Хотя
загрузка bitmap индекса занимает немного больше времени, это значительно
сокращает время загрузки данных.
    BitmapIndexFilter: 128.403ms // Время фильтрации данных с помощью
bitmap индекса.
    BitmapIndexFilterRows: 143.995M (143995376) // Количество строк,
отфильтрованных bitmap индексом.
```

Запрос 3: Запрос к одному столбцу с высокой cardinality

Запрос к таблице без bitmap индекса

Запрос:

```
select count(1) from lineorder_without_index where lo_partkey=10000;
```

Анализ производительности запроса: Поскольку запрашиваемая таблица не имеет bitmap индекса, pages, содержащие данные столбца `lo_partkey`, читаются, а затем применяется фильтрация предикатов.

Общее время: приблизительно 0,43 секунды, **включая 0,39 секунды на загрузку данных** и 0,02 секунды на фильтрацию предикатов.

```
PullRowNum: 255 // Количество строк в результирующем наборе.
CompressedBytesRead: 344.532 MB // Общий объем прочитанных данных.
RawRowsRead: 143.999M (143,999,468) // Количество прочитанных строк.
Поскольку bitmap индекса нет, все данные в этом столбце читаются.
ReadPagesNum: 8.791K (8,791) // Количество прочитанных pages. Поскольку
bitmap индекса нет, все данные этого столбца читаются.
IOTaskExecTime: 428.258ms // Общее время сканирования данных.
    BlockFetch: 392.434ms // Время загрузки данных.
    PredFilter: 20.807ms // Время фильтрации предикатов.
    PredFilterRows: 143.999M (143,999,213) // Количество строк,
отфильтрованных предикатами.
```

Запрос к таблице с bitmap индексом

Принудительное использование bitmap индекса

> **К СВЕДЕНИЮ**
>
> Для принудительного использования bitmap индекса, согласно конфигурации Selena, вы должны установить `bitmap_max_filter_ratio=1000` в файле **be.conf** на каждом узле BE, а затем перезапустить узлы BE.

Запрос:

```
SELECT count(1) FROM lineorder_with_index WHERE lo_partkey=10000;
```

Анализ производительности запроса: Поскольку запрашиваемый столбец имеет высокую cardinality, bitmap индекс эффективен, позволяя отфильтровать часть pages и значительно сокращая время чтения данных.

Общее время: 0,015 секунды, **включая 0,009 секунды на загрузку данных и bitmap индекса**, и 0,003 секунды на фильтрацию bitmap индекса.

```
PullRowNum: 255 // Количество строк в результирующем наборе.
CompressedBytesRead: 13.600 MB // Общий объем прочитанных данных.
Фильтрация Bitmap Index эффективна, отфильтровывая большой объем данных.
RawRowsRead: 255 // Количество прочитанных строк.
ReadPagesNum: 225 // Количество прочитанных pages. Bitmap индекс может
эффективно фильтровать данные, отфильтровывая большое количество pages.
IOTaskExecTime: 15.354ms // Общее время сканирования данных, значительно
меньше общего времени без Bitmap Index.
    BlockFetch: 9.530ms // Время загрузки данных и Bitmap Index. Хотя
дополнительное время тратится на загрузку Bitmap Index, это значительно
сокращает время загрузки данных.
    BitmapIndexFilter: 3.450ms // Время фильтрации данных с помощью Bitmap
Index.
    BitmapIndexFilterRows: 143.999M (143,999,213) // Количество строк,
отфильтрованных Bitmap Index.
```

Определение использования bitmap индекса на основе конфигурации по умолчанию Selena

Запрос:

```
SELECT count(1) FROM lineorder_with_index WHERE lo_partkey=10000;
```

Анализ производительности запроса: Согласно конфигурации по умолчанию Selena, Bitmap Index используется, когда количество различных значений/cardinality столбца < `bitmap_max_filter_ratio/1000` (по умолчанию 1/1000). Поскольку это условие выполнено, запрос использует Bitmap Index, и производительность аналогична той, когда Bitmap Index используется принудительно.

Общее время: 0,014 секунды, включая **0,008 секунды на загрузку данных и bitmap индекса**, и 0,003 секунды на фильтрацию Bitmap Index.

```
PullRowNum: 255 // Количество строк в результирующем наборе.
CompressedBytesRead: 13.600 MB // Общий объем прочитанных данных. Bitmap
индекс может эффективно фильтровать данные, отфильтровывая большой объем
данных.
RawRowsRead: 255 // Количество прочитанных строк.
ReadPagesNum: 225 // Количество прочитанных pages. Bitmap индекс может
эффективно фильтровать данные, отфильтровывая большое количество pages.
IOTaskExecTime: 14.387ms // Общее время сканирования данных, значительно
меньше общего времени без Bitmap Index.
    BitmapIndexFilter: 2.979ms // Время фильтрации данных с помощью Bitmap
Index.
    BitmapIndexFilterRows: 143.999M (143,999,213) // Количество строк,
отфильтрованных Bitmap Index.
    BlockFetch: 8.988ms // Время загрузки данных и Bitmap Index. Хотя
дополнительное время тратится на загрузку Bitmap Index, это значительно
сокращает время загрузки данных.
```

2.6.4 Bloom filter индексы

В этом разделе описывается создание и изменение bloom filter индексов, а также принципы их работы.

Bloom filter индекс - это пространственно-эффективная структура данных, которая используется для определения возможного наличия фильтруемых данных в файлах данных таблицы. Если bloom filter индекс определяет, что данные для фильтрации отсутствуют в определённом файле данных, Selena пропускает сканирование этого файла. Bloom filter индексы могут сократить время отклика, когда колонка (например, ID) имеет относительно высокую кардинальность.

Если запрос обращается к колонке sort key, Selena эффективно возвращает результат запроса, используя prefix индекс. Однако запись prefix индекса для блока данных не может превышать 36 байт в длину. Если вы хотите улучшить производительность запросов по колонке, которая не используется как sort key и имеет относительно высокую кардинальность, вы можете создать bloom filter индекс для этой колонки.

Как это работает

Например, вы создаёте bloom filter индекс на колонке `column1` таблицы `table1` и выполняете запрос типа `Select xxx from table1 where column1 = something;`. Тогда происходит следующее, когда Selena сканирует файлы данных `table1`.

- Если bloom filter индекс определяет, что файл данных не содержит данные для фильтрации, Selena пропускает файл данных для повышения производительности запроса.
- Если bloom filter индекс определяет, что файл данных может содержать данные для фильтрации, Selena читает файл данных, чтобы проверить, существуют ли данные. Обратите внимание, что bloom filter может точно сказать, что значение отсутствует, но не может с уверенностью сказать, что значение присутствует - только что оно может присутствовать. Использование bloom filter индекса для определения наличия значения может давать ложноположительные срабатывания, что означает, что bloom filter индекс определяет, что файл данных содержит данные для фильтрации, но на самом деле файл данных их не содержит.

Примечания по использованию

- Вы можете создавать bloom filter индексы для всех колонок таблицы Duplicate Key или Primary Key. Для таблиц Aggregate или Unique Key вы можете создавать bloom filter индексы только для key колонок.
- Bloom filter индексы могут быть созданы для колонок следующих типов данных:
  - Числовые типы: SMALLINT, INT, BIGINT и LARGEINT.
  - Строковые типы: CHAR, STRING и VARCHAR.
  - Типы дат: DATE и DATETIME.
- Bloom filter индексы могут улучшить производительность только тех запросов, которые содержат операторы `in` и `=`, например `Select xxx from table where x in {}` и `Select xxx from table where column = xxx`.
- Вы можете проверить, использует ли запрос bloom filter индексы, просмотрев поле `BloomFilterFilterRows` в профиле запроса.

Создание bloom filter индексов

Вы можете создать bloom filter индекс для колонки при создании таблицы, указав параметр `bloom_filter_columns` в `PROPERTIES`. Например, создание bloom filter индексов для колонок `k1` и `k2` в `table1`.

```
CREATE TABLE table1
(
    k1 BIGINT,
    k2 LARGEINT,
    v1 VARCHAR(2048) REPLACE,
    v2 SMALLINT DEFAULT "10"
)
ENGINE = olap
PRIMARY KEY(k1, k2)
DISTRIBUTED BY HASH (k1, k2)
PROPERTIES("bloom_filter_columns" = "k1,k2");
```

Вы можете создать bloom filter индексы для нескольких колонок одновременно, указав имена этих колонок. Обратите внимание, что имена колонок необходимо разделять запятыми (`,`). Описание других параметров оператора CREATE TABLE см. в CREATE TABLE.

Отображение bloom filter индексов

Например, следующий оператор отображает bloom filter индексы для `table1`. Описание выходных данных см. в SHOW CREATE TABLE.

```
SHOW CREATE TABLE table1;
```

Изменение bloom filter индексов

Вы можете добавлять, уменьшать и удалять bloom filter индексы с помощью оператора ALTER TABLE.

- Следующий оператор добавляет bloom filter индекс на колонку `v1`.

  ```
  ALTER TABLE table1 SET ("bloom_filter_columns" = "k1,k2,v1");
  ```

- Следующий оператор удаляет bloom filter индекс с колонки `k2`.

  ```
  ALTER TABLE table1 SET ("bloom_filter_columns" = "k1");
  ```

- Следующий оператор удаляет все bloom filter индексы для `table1`.

```
ALTER TABLE table1 SET ("bloom_filter_columns" = "");
```

> Примечание: Изменение индекса является асинхронной операцией. Вы можете просмотреть прогресс этой операции, выполнив SHOW ALTER TABLE. Одновременно можно выполнять только одну задачу изменения индекса для таблицы.

2.6.5 N-gram bloom filter индекс

> *Beta* **BETA FEATURE**
>
> Рекомендации по использованию бета-функций

N-gram bloom filter индекс - это специальный Bloom filter индекс, который обычно используется для ускорения LIKE запросов и скорости вычисления функций `ngram_search` и `ngram_search_case_insensitive`.

N-gram bloom filter индекс подходит только для колонок строкового типа (`STRING`, `CHAR` или `VARCHAR`). Разница между N-gram bloom filter индексом и Bloom filter индексом заключается в том, что N-gram bloom filter индекс сначала токенизирует строки, а затем записывает полученные подстроки в Bloom filter индекс. Например, значение определенной индексированной колонки - это строка `Technical`. Для традиционного Bloom filter индекса вся строка `Technical` будет непосредственно записана в индекс. Однако для N-gram bloom filter индекса с указанным `gram_num` равным `4`, строка `Technical` будет сначала токенизирована в следующие подстроки:

```
"Tech", "echn", "chni", "hnic", "nica", "ical"
```

И затем каждая из этих шести подстрок записывается в Bloom filter индекс.

Примечания по использованию

- Для таблицы Duplicate Key или Primary Key вы можете создавать N-gram bloom filter индексы для всех колонок (строковых типов). Для таблицы Aggregate или Unique Key вы можете создавать N-gram bloom filter индексы только для key колонок (строковых типов).
- N-gram bloom filter индексы могут быть созданы только для колонок строковых типов (CHAR, STRING и VARCHAR).

- Чтобы определить, использует ли запрос N-gram bloom filter индекс, вы можете проверить поле `BloomFilterFilterRows` в профиле запроса.
- Для одной колонки может быть создан только один тип индекса (Bloom Filter или N-gram Bloom Filter).

Основные операции

Создание N-gram bloom filter индекса

```
CREATE TABLE test.table1
(
    k1 CHAR(10),
    k2 CHAR(10),
    v1 INT SUM,
    INDEX index_name (k2) USING NGRAMBF ("gram_num" = "4",
                                         "bloom_filter_fpp" = "0.05")
COMMENT ''
)
ENGINE = olap
AGGREGATE KEY(k1, k2)
DISTRIBUTED BY HASH(k1);
```

Параметры, связанные с N-gram bloom filter индексом:

Параметр	Обязательный	Описание
`index_name`	Да	Имя индекса. Имена индексов должны быть уникальными в пределах таблицы.
`column_name`	Да	Имя колонки, для которой создается индекс. Можно указать только одно имя колонки. В примере выше это `k2`.
`gram_num`	Нет	Длина подстроки после токенизации строки в индексируемой колонке. Значение по умолчанию - 2.
`bloom_filter_fpp`	Нет	Вероятность ложноположительного срабатывания Bloom filter в диапазоне от 0.0001 до 0.05. Значение по умолчанию - 0.05. Меньшее значение обеспечивает лучшую фильтрацию, но требует больших накладных расходов на хранение.
`case_sensitive`	Нет	Чувствителен ли этот индекс к регистру. Значение по умолчанию - `true`.
`COMMENT`	Нет	Комментарий к индексу.

Описание других параметров, связанных с созданием таблицы, см. в CREATE TABLE.

Просмотр N-gram bloom filter индекса

Вы можете просмотреть все индексы таблицы, используя `SHOW CREATE TABLE` или `SHOW INDEX FROM` table. Поскольку создание индекса выполняется асинхронно, вы сможете увидеть соответствующий индекс только после успешного создания индекса.

```
SHOW CREATE TABLE table1;
SHOW INDEX FROM table1;
```

Изменение N-gram bloom filter индекса

Вы можете добавлять и удалять N-gram bloom filter индекс с помощью оператора ALTER TABLE.

- Выполните следующий оператор, чтобы добавить новый N-gram bloom filter индекс `new_index_name` для колонки `k1` в таблицу `table1`.

```
ALTER TABLE table1
ADD INDEX new_index_name(k1) USING NGRAMBF ("gram_num" = "4",
                                            "bloom_filter_fpp" =
"0.05") COMMENT '';
```

- Выполните следующий оператор, чтобы удалить N-gram bloom filter индекс `new_index_name` из таблицы `table1`.

```
ALTER TABLE table1 DROP INDEX new_index_name;
```

> **ПРИМЕЧАНИЕ**
>
> Изменение индекса является асинхронной операцией. Вы можете просмотреть прогресс этой операции, выполнив SHOW ALTER TABLE. Одновременно можно выполнять только одну задачу изменения индекса для таблицы.

Запросы, которые могут быть ускорены

`LIKE` запросы

N-gram bloom filter индекс может ускорить `LIKE` запросы, если `gram_num` достаточно мал (меньше длины строки, для которой выполняется запрос). В противном случае N-gram bloom filter индекс не может ускорить `LIKE` запросы.

Например, если `gram_num` равен `4`, и оператор запроса `SELECT * FROM table WHERE col1 LIKE "%abc"`, N-Gram Bloom filter индекс не ускорит этот запрос, потому что запрашиваемая строка `abc` имеет только три символа, что меньше значения `4` параметра `gram_num`. Если условие запроса `WHERE col1 LIKE "%abcd"` или `WHERE col1 LIKE "%abcde%"`, N-Gram Bloom filter индекс ускорит запрос.

ngram_search

Когда функция `ngram_search` используется в запросе, если колонка, указанная в функции, имеет N-gram bloom filter индекс, и `gram_num`, указанный в функции, соответствует `gram_num` N-gram bloom filter индекса, индекс автоматически отфильтрует данные с похожестью строк 0, значительно ускоряя процесс выполнения функции.

ngram_search_case_insensitive

Использование этой функции такое же, как у `ngram_search`, за исключением того, что эта функция чувствительна к регистру. Поэтому требуется, чтобы параметр `case_sensitive` был установлен в `false` при создании N-gram bloom filter индекса.

```
CREATE TABLE test.table1
(
    k1 CHAR(10),
    k2 CHAR(10),
    v1 INT SUM,
    INDEX index_name (k2) USING NGRAMBF ("gram_num" = "4",
                                         "bloom_filter_fpp" = "0.05",
                                         "case_sensitive" = "false")
COMMENT ''
)
ENGINE = olap
AGGREGATE KEY(k1, k2)
DISTRIBUTED BY HASH(k1);
```

Если индекс уже создан и его параметр `case_sensitive` установлен в `true`, вы можете изменить параметр этого индекса на `false`, используя:

```
ALTER TABLE table1
ADD INDEX new_index_name(k1) USING NGRAMBF ("gram_num" = "4",
                                            "bloom_filter_fpp" = "0.05",
                                            "case_sensitive" = "false")
COMMENT '';
```

2.6.6 Полнотекстовый inverted индекс

> *Beta* **BETA FEATURE**
>
> Рекомендации по использованию бета-функций

Начиная с версии 3.3.0, Selena поддерживает полнотекстовые inverted индексы, которые могут разбивать текст на более мелкие слова и создавать индексную запись для каждого слова, показывающую отношение отображения между словом и соответствующим номером строки в файле данных. Для полнотекстового поиска Selena запрашивает inverted индекс на основе ключевых слов поиска, быстро определяя строки данных, соответствующие ключевым словам.

Primary Key таблицы поддерживают полнотекстовые inverted индексы начиная с v2.0.0.

Полнотекстовый inverted индекс пока не поддерживается в shared-data кластерах.

Обзор

Selena хранит свои базовые данные в файлах данных, организованных по столбцам. Каждый файл данных содержит полнотекстовый inverted индекс на основе индексированных столбцов. Значения в индексированных столбцах токенизируются на отдельные слова. Каждое слово после токенизации обрабатывается как индексная запись, отображаемая на номер строки, в которой появляется слово. В настоящее время поддерживаются методы токенизации для английской токенизации, китайской токенизации, многоязычной токенизации и без токенизации.

Например, если строка данных содержит "hello world" и её номер строки 123, полнотекстовый inverted индекс строит индексные записи на основе этого результата токенизации и номера строки: hello->123, world->123.

Во время полнотекстового поиска Selena может находить индексные записи, содержащие ключевые слова поиска, используя полнотекстовые inverted индексы, а затем быстро находить номера строк, в которых появляются ключевые слова, значительно уменьшая количество строк данных, которые необходимо сканировать.

Базовые операции

Создание полнотекстового inverted индекса

Перед созданием полнотекстового inverted индекса необходимо включить параметр конфигурации FE `enable_experimental_gin`.

```
ADMIN SET FRONTEND CONFIG ("enable_experimental_gin" = "true");
```

> **ПРИМЕЧАНИЕ**
>
> При создании полнотекстового inverted индекса для таблицы необходимо отключить функцию `replicated_storage` для таблицы.
>
> - Для v2.0.0 и более поздних версий эта функция автоматически отключается при создании индекса.
> - Для версий ранее v2.0.0 необходимо вручную установить свойство таблицы `replicated_storage` в `false`.

Создание полнотекстового Inverted индекса при создании таблицы

Создание полнотекстового inverted индекса на столбце `v` с английской токенизацией.

```
CREATE TABLE `t` (
  `k` BIGINT NOT NULL COMMENT "",
  `v` STRING COMMENT "",
   INDEX idx (v) USING GIN("parser" = "english")
) ENGINE=OLAP
DUPLICATE KEY(`k`)
DISTRIBUTED BY HASH(`k`) BUCKETS 1
PROPERTIES (
"replicated_storage" = "false"
);
```

- Параметр `parser` указывает метод токенизации. Поддерживаемые значения и описания следующие:
  - `none` (по умолчанию): без токенизации. Вся строка данных в индексированном столбце обрабатывается как один индексный элемент при построении полнотекстового inverted индекса.
  - `english`: английская токенизация. Этот метод токенизации обычно разбивает на любом не-алфавитном символе. Также заглавные английские буквы преобразуются в строчные. Следовательно, ключевые слова в условиях запроса должны быть строчными английскими буквами, а не заглавными, чтобы использовать полнотекстовый inverted индекс для поиска строк данных.
  - `chinese`: китайская токенизация. Этот метод токенизации использует CJK Analyzer в CLucene для токенизации.
  - `standard`: многоязычная токенизация. Этот метод токенизации обеспечивает токенизацию на основе грамматики (основанную на алгоритме сегментации текста Unicode) и хорошо работает для большинства языков и случаев смешанных языков, таких как китайский и английский. Например, этот метод токенизации может различать китайский и английский, когда эти два языка сосуществуют. После токенизации английского он преобразует заглавные английские буквы в строчные. Следовательно, ключевые слова в условиях запроса должны быть строчными английскими буквами, а не заглавными, чтобы использовать полнотекстовый inverted индекс для поиска строк данных.
- Тип данных индексированного столбца должен быть CHAR, VARCHAR или STRING.

Добавление полнотекстового inverted индекса после создания таблицы

После создания таблицы вы можете добавить полнотекстовый inverted индекс, используя `ALTER TABLE ADD INDEX` или `CREATE INDEX`.

```
ALTER TABLE t ADD INDEX idx (v) USING GIN('parser' = 'english');
CREATE INDEX idx ON t (v) USING GIN('parser' = 'english');
```

Управление полнотекстовым inverted индексом

Просмотр полнотекстового inverted индекса

Выполните `SHOW CREATE TABLE` для просмотра полнотекстовых inverted индексов.

```
MySQL [example_db]> SHOW CREATE TABLE t\G
```

Удаление полнотекстового inverted индекса

Выполните `ALTER TABLE ADD INDEX` или `DROP INDEX` для удаления полнотекстовых inverted индексов.

```
DROP INDEX idx on t;
ALTER TABLE t DROP index idx;
```

Ускорение запросов с помощью полнотекстового inverted индекса

После создания полнотекстового inverted индекса необходимо убедиться, что системная переменная `enable_gin_filter` включена, чтобы inverted индекс мог ускорять запросы. Также необходимо учитывать, токенизированы ли значения столбцов индекса, чтобы определить, какие запросы могут быть ускорены.

Поддерживаемые запросы при токенизированном индексированном столбце

Если полнотекстовый inverted индекс токенизирует индексированные столбцы, то есть `'parser' = 'standard|english|chinese'`, для фильтрации данных с использованием полнотекстовых inverted индексов поддерживается только предикат `MATCH` или `MATCH_ANY`, и формат должен быть `<col_name> (NOT) MATCH '%keyword%'` или `<col_name> (NOT) MATCH_ANY 'keyword1, keyword2'`. `keyword` должно быть строковым литералом и не поддерживает выражения.

1. Создайте таблицу и вставьте несколько строк тестовых данных.

```
CREATE TABLE `t` (
    `id1` bigint(20) NOT NULL COMMENT "",
    `value` varchar(255) NOT NULL COMMENT "",
    INDEX gin_english (`value`) USING GIN ("parser" = "english")
COMMENT 'english index'
)
DUPLICATE KEY(`id1`)
DISTRIBUTED BY HASH(`id1`)
PROPERTIES (
"replicated_storage" = "false"
);


INSERT INTO t VALUES
    (1, "selena is a database

1"),
    (2, "selena is a data warehouse");
```

2. Используйте предикат `MATCH` для запросов.

- Запрос строк данных, столбец `value` которых содержит ключевое слово `selena`.

```
MySQL [example_db]> SELECT * FROM t WHERE t.value MATCH "selena";
```

- Получите строки данных, столбец `value` которых содержит ключевое слово, начинающееся с `data`.

```
MySQL [example_db]> SELECT * FROM t WHERE t.value MATCH "data%";
```

3. Используйте предикат `MATCH_ANY` для запросов.

- Запрос строк данных, столбец `value` которых содержит ключевое слово `database` или `data`.

```
MySQL [example_db]> SELECT * FROM t WHERE t.value MATCH_ANY "database
data";
```

Примечания:

- Во время запросов ключевые слова могут быть сопоставлены нечётко с использованием `%`, в формате `%keyword%`. Однако ключевое слово должно содержать часть слова. Например, если ключевое слово `selena` , оно не может соответствовать слову `selena`, потому что содержит пробелы.

```
MySQL [example_db]> SELECT * FROM t WHERE t.value MATCH "star%";
+------+-------------------------------+
| id1  | value                         |
+------+-------------------------------+
|    1 | selena is a database1      |
|    2 | selena is a data warehouse |
+------+-------------------------------+
2 rows in set (0.02 sec)

MySQL [example_db]> SELECT * FROM t WHERE t.value MATCH "selena ";
Empty set (0.02 sec)
```

- Если английская или многоязычная токенизация используется для построения полнотекстового inverted индекса, заглавные английские слова преобразуются в строчные при фактическом хранении полнотекстового inverted индекса. Следовательно, во время запросов ключевые слова должны быть строчными, а не заглавными, чтобы использовать полнотекстовый inverted индекс для поиска строк данных.

```
MySQL [example_db]> INSERT INTO t VALUES (3, "Selena is the BEST");

MySQL [example_db]> SELECT * FROM t;
+------+-------------------------------+
| id1  | value                         |
+------+-------------------------------+
|    1 | selena is a database          |
|    2 | selena is a data warehouse    |
|    3 | Selena is the BEST            |
+------+-------------------------------+
3 rows in set (0.02 sec)

MySQL [example_db]> SELECT * FROM t WHERE t.value MATCH "BEST"; --
Ключевое слово в верхнем регистре
Empty set (0.02 sec) -- Возвращает пустой набор результатов

MySQL [example_db]> SELECT * FROM t WHERE t.value MATCH "best"; --
Ключевое слово в нижнем регистре
+------+-----------------------+
| id1  | value                 |
+------+-----------------------+
|    3 | Selena is the BEST | -- Может найти строки данных,
соответствующие условию
+------+-----------------------+
1 row in set (0.01 sec)
```

- Предикат `MATCH` или `MATCH_ANY` в условиях запроса должен использоваться как pushdown предикат, поэтому он должен быть в предложении WHERE и выполняться для индексированного столбца.

  Возьмём следующую таблицу и тестовые данные в качестве примера:

```
CREATE TABLE `t_match` (
    `id1` bigint(20) NOT NULL COMMENT "",
    `value` varchar(255) NOT NULL COMMENT "",
    `value_test` varchar(255) NOT NULL COMMENT "",
    INDEX gin_english (`value`) USING GIN("parser" = "english") COMMENT
'english index'
)
ENGINE=OLAP
DUPLICATE KEY(`id1`)
DISTRIBUTED BY HASH (`id1`) BUCKETS 1
PROPERTIES (
"replicated_storage" = "false"
);

INSERT INTO t_match VALUES (1, "test", "test");
```

Следующие операторы запроса не соответствуют требованиям:

- Поскольку предикат `MATCH` в операторе запроса не находится в предложении WHERE, он не может быть отправлен вниз, что приводит к ошибке запроса.

  ```
  MySQL [test]> SELECT value MATCH "test" FROM t_match;
  ERROR 1064 (HY000): Match can only be used as a pushdown predicate
  on a column with GIN in a single query.
  ```

- Поскольку столбец `value_test`, для которого выполняется предикат `MATCH` в операторе запроса, не является индексированным столбцом, запрос не выполняется.

  ```
  MySQL [test]> SELECT * FROM t_match WHERE value_test match "test";
  ERROR 1064 (HY000): Match can only be used as a pushdown predicate
  on a column with GIN in a single query.
  ```

Поддерживаемые запросы при нетокенизированном индексированном столбце

Если полнотекстовый inverted индекс не токенизирует индексированный столбец, то есть `'parser' = 'none'`, все pushdown предикаты в условиях запроса, перечисленные ниже, могут использоваться для фильтрации данных с использованием полнотекстового inverted индекса:

- Предикаты выражений: (NOT) LIKE, (NOT) MATCH, (NOT) MATCH_ANY

> **ПРИМЕЧАНИЕ**
>
> - В этом случае `MATCH` семантически эквивалентен `LIKE`.
> - `MATCH` и `LIKE` поддерживают только формат `(NOT) <col_name> MATCH|LIKE '%keyword%'`. `keyword` должно быть строковым литералом и не поддерживает выражения. Обратите внимание, что если `LIKE` не соответствует этому формату, даже если запрос может выполняться нормально, он деградирует до запроса, который не использует полнотекстовый inverted индекс для фильтрации данных.

- Регулярные предикаты: `==`, `!=`, `<=`, `>=`, `NOT IN`, `IN`, `IS NOT NULL`, `NOT NULL`

Проверка, ускоряется ли запрос полнотекстовым inverted индексом

После выполнения запроса вы можете просмотреть детальные метрики `GinFilterRows` и `GinFilter` в узле сканирования Query Profile, чтобы увидеть количество отфильтрованных строк и время фильтрации с использованием полнотекстового inverted индекса.

2.6.7 Vector индекс

> *Beta* **BETA FEATURE**
>
> Рекомендации по использованию бета-функций

Эта тема представляет функцию vector индекса Selena и как выполнять приблизительный поиск ближайших соседей (ANNS) с её помощью.

Функция vector индекса поддерживается только в shared-nothing кластерах версии v1.5.2 или более поздней.

Обзор

В настоящее время Selena поддерживает векторную индексацию на уровне файлов Segment. Индекс отображает каждый элемент поиска на его ID строки в файлах Segment, позволяя быстро извлекать данные путём прямого определения соответствующих строк данных без вычислений векторных расстояний методом грубой силы. Система теперь

предоставляет два типа векторной индексации: Inverted File with Product Quantization (IVFPQ) и Hierarchical Navigable Small World (HNSW), каждый со своей организационной структурой.

Inverted File with Product Quantization (IVFPQ)

Inverted File with Product Quantization (IVFPQ) — это метод приблизительного поиска ближайших соседей в крупномасштабных многомерных векторах, обычно используемый в задачах векторного поиска в глубоком обучении и машинном обучении. IVFPQ состоит из двух основных компонентов: inverted file и product quantization.

- **Inverted File**: Этот компонент является методом индексации. Он разделяет набор данных на несколько кластеров (или ячеек Вороного), каждый с центроидом (начальной точкой), и назначает каждую точку данных (вектор) её ближайшему центру кластера. Для векторного поиска IVFPQ нужно только искать в ближайшем кластере, значительно уменьшая область и сложность поиска.
- **Product Quantization**: Этот компонент является методом сжатия данных. Он разбивает многомерные векторы на подвекторы и квантует каждый подвектор для отображения его на ближайшую точку в предопределённом наборе, уменьшая затраты на хранение и вычисления при сохранении высокой точности.

Комбинируя inverted files и product quantization, IVFPQ обеспечивает эффективный приблизительный поиск ближайших соседей в больших многомерных наборах данных.

Hierarchical Navigable Small World (HNSW)

Hierarchical Navigable Small World (HNSW) — это алгоритм на основе графов для поиска ближайших соседей в многомерных пространствах, также широко используемый в задачах векторного поиска.

HNSW строит иерархическую структуру графов, в которой каждый уровень представляет собой navigable small world (NSW) граф. В графе каждая вершина представляет точку данных, а рёбра указывают на сходство между вершинами. Верхние уровни графа содержат меньше вершин с более разреженными соединениями для быстрого глобального поиска, в то время как нижние уровни включают все вершины с более плотными соединениями для точного локального поиска.

Для векторного поиска HNSW сначала выполняет поиск на верхнем уровне, быстро определяя приблизительную область ближайших соседей, затем перемещается вниз по

уровням, чтобы найти точного ближайшего соседа на нижнем уровне.

HNSW предлагает как эффективность, так и точность, делая его адаптируемым к различным данным и распределениям запросов.

Сравнение IVFPQ и HNSW

- **Коэффициент сжатия данных**: IVFPQ имеет более высокий коэффициент сжатия (около 1:0.15). Расчёт индексации обеспечивает только предварительные результаты сортировки после грубой ранжировки, потому что PQ сжимает векторы. Требуется дополнительная тонкая ранжировка для окончательных результатов сортировки, что приводит к более высоким вычислениям и задержке. HNSW имеет более низкий коэффициент сжатия (около 1:0.8) и обеспечивает точную ранжировку без дополнительной обработки, что приводит к более низким вычислительным затратам и задержке и более высоким затратам на хранение.
- **Настройка Recall**: Оба индекса поддерживают настройку уровня recall через регулировку параметров, но IVFPQ влечёт более высокие вычислительные затраты при аналогичных уровнях recall.
- **Стратегия кэширования**: IVFPQ позволяет балансировать затраты на память и задержку вычислений путём регулировки пропорции кэша блоков индекса, в то время как HNSW в настоящее время поддерживает только полное кэширование файла.

Использование

Каждая таблица поддерживает только один vector индекс.

Предварительные условия

Перед созданием vector индексов необходимо включить его, установив параметр конфигурации FE `enable_experimental_vector` в `true`.

Выполните следующий оператор для динамического включения:

```
ADMIN SET FRONTEND CONFIG ("enable_experimental_vector" = "true");
```

Чтобы включить его постоянно, необходимо добавить `enable_experimental_vector = true` в файл конфигурации FE `fe.conf` и перезапустить FE.

Создание vector индекса

Это руководство создаёт vector индексы при создании таблиц. Вы также можете добавить vector индексы к существующей таблице. См. Добавление vector индекса для подробных инструкций.

- Следующий пример создаёт HNSW vector индекс `hnsw_vector` на столбце `vector` в таблице `hnsw`.

```
CREATE TABLE hnsw (
    id     BIGINT(20)   NOT NULL COMMENT "",
    vector ARRAY<FLOAT> NOT NULL COMMENT "",
    INDEX hnsw_vector (vector) USING VECTOR (
        "index_type" = "hnsw",
        "dim"="5",
        "metric_type" = "l2_distance",
        "is_vector_normed" = "false",
        "M" = "16",
        "efconstruction" = "40"
    )
) ENGINE=OLAP
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1;
```

- Следующий пример создаёт IVFPQ vector индекс `ivfpq_vector` на столбце `vector` в таблице `ivfpq`.

```
CREATE TABLE ivfpq (
    id     BIGINT(20)   NOT NULL COMMENT "",
    vector ARRAY<FLOAT> NOT NULL COMMENT "",
    INDEX ivfpq_vector (vector) USING VECTOR (
        "index_type" = "ivfpq",
        "dim"="5",
        "metric_type" = "l2_distance",
        "is_vector_normed" = "false",
        "nbits" = "16",
        "nlist" = "40"
    )
) ENGINE=OLAP
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1;
```

Параметры построения индекса

USING VECTOR

- **По умолчанию**: N/A
- **Обязательно**: Да
- **Описание**: Создаёт vector индекс.

index_type

- **По умолчанию**: N/A
- **Обязательно**: Да
- **Описание**: Тип vector индекса. Допустимые значения: `hnsw` и `ivfpq`.

dim

- **По умолчанию**: N/A
- **Обязательно**: Да
- **Описание**: Размерность индекса. После построения индекса векторы, не соответствующие требованиям размерности, будут отклонены при загрузке в базовый столбец. Должно быть целым числом больше или равным `1`.

metric_type

- **По умолчанию**: N/A
- **Обязательно**: Да
- **Описание**: Тип метрики (функция измерения) vector индекса. Допустимые значения:
  - `l2_distance`: Евклидово расстояние. Чем меньше значение, тем выше сходство.
  - `cosine_similarity`: Косинусное сходство. Чем больше значение, тем выше сходство.

is_vector_normed

- **По умолчанию**: false
- **Обязательно**: Нет
- **Описание**: Нормализованы ли векторы. Допустимые значения `true` и `false`. Действует только когда `metric_type` равен `cosine_similarity`. Если векторы нормализованы, значение вычисленных расстояний будет в пределах [-1, 1]. Векторы должны удовлетворять условию, что сумма квадратов равна `1`, иначе возвращается ошибка.

M

- **По умолчанию**: 16

- **Обязательно**: Нет
- **Описание**: Специфический параметр HNSW. Количество двунаправленных соединений, создаваемых для каждого нового элемента во время построения графа. Должно быть целым числом больше или равным `2`. Значение `M` напрямую влияет на эффективность и точность построения и поиска графа. Во время построения графа каждая вершина попытается установить соединения с её ближайшими `M` вершинами. Если вершина уже имеет `M` соединений, но находит более близкую вершину, самое дальнее соединение будет удалено, и новое соединение будет установлено с более близкой вершиной. Векторный поиск начнётся с точки входа и найдёт ближайшего соседа вдоль вершин, соединённых с ней. Следовательно, чем больше значение `M`, тем больше область поиска для каждой вершины, тем выше эффективность поиска, но тем выше стоимость построения и хранения графа.

efconstruction

- **По умолчанию**: 40
- **Обязательно**: Нет
- **Описание**: Специфический параметр HNSW. Размер списка кандидатов, содержащего ближайших соседей. Должно быть целым числом больше или равным `1`. Используется для контроля глубины поиска во время процесса построения графа. В частности, `efconstruction` определяет размер списка поиска (также известного как список кандидатов) для каждой вершины во время процесса построения графа. Этот список кандидатов используется для хранения кандидатов соседей текущей вершины, а размер списка равен `efconstruction`. Чем больше значение `efconstruction`, тем больше кандидатов рассматривается как соседи вершины во время процесса построения графа, и, как результат, лучше качество (например, лучшая связность) графа, но также выше потребление времени и вычислительная сложность построения графа.

nbits

- **По умолчанию**: 16
- **Обязательно**: Нет
- **Описание**: Специфический параметр IVFPQ. Точность Product Quantization (PQ). Должно быть кратным `8`. С IVFPQ каждый вектор разделяется на несколько подвекторов, и затем каждый подвектор квантуется. `Nbits` определяет точность квантования, то есть на сколько двоичных цифр квантуется каждый подвектор. Чем больше значение `nbits`, тем выше точность квантования, но тем выше затраты на хранение и вычисления.

nlist

- **По умолчанию**: 16
- **Обязательно**: Нет
- **Описание**: Специфический параметр IVFPQ. Количество кластеров или inverted списков. Должно быть целым числом больше или равным `1`. С IVFPQ набор данных разделяется на кластеры, и центроид каждого кластера соответствует inverted списку. Векторный поиск сначала найдёт центроид кластера, ближайший к точке данных, и затем извлечёт ближайших соседей в соответствующем inverted списке. Следовательно, значение `nlist` будет влиять на точность и эффективность поиска. Чем больше значение `nlist`, тем мельче гранулярность кластеризации, таким образом, выше точность поиска, но также выше сложность поиска.

M_IVFPQ

- **Обязательно**: Да
- **Описание**: Специфический параметр IVFPQ. Количество подвекторов, на которые будет разделён исходный вектор. IVFPQ индекс разделит вектор размерности `dim` на `M_IVFPQ` подвекторов равной длины. Следовательно, должно быть делителем значения `dim`.

Добавление vector индекса

Вы также можете добавить vector индексы к существующей таблице, используя CREATE INDEX или ALTER TABLE ADD INDEX.

Пример:

```
CREATE INDEX ivfpq_vector
ON ivfpq (vector)
USING VECTOR (
    "index_type" = "ivfpq",
    "metric_type" = "l2_distance",
    "is_vector_normed" = "false",
    "dim"="5",
    "nlist" = "256",
    "nbits"="10"
);

ALTER TABLE ivfpq
ADD INDEX ivfpq_vector (vector)
USING VECTOR (
    "index_type" = "ivfpq",
    "metric_type" = "l2_distance",
    "is_vector_normed" = "false",
    "dim"="5",
    "nlist" = "256",
    "nbits"="10"
);
```

Управление vector индексами

Просмотр vector индекса

Вы можете просмотреть определение vector индекса с помощью оператора SHOW CREATE TABLE:

Пример:

```
mysql> SHOW CREATE TABLE hnsw \G
*************************** 1. row ***************************
       Table: hnsw
Create Table: CREATE TABLE hnsw (
  id bigint(20) NOT NULL COMMENT "",
  vector array<float> NOT NULL COMMENT "",
  INDEX index_vector (vector) USING VECTOR("dim" = "5", "efconstruction" =
"40", "index_type" = "hnsw", "is_vector_normed" = "false", "M" = "512",
"metric_type" = "l2_distance") COMMENT ''
) ENGINE=OLAP
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1
PROPERTIES (
"compression" = "LZ4",
"fast_schema_evolution" = "true",
"replicated_storage" = "false",
"replication_num" = "3"
);
1 row in set (0.00 sec)
```

Удаление vector индекса

Вы можете удалить vector индексы, используя DROP INDEX или ALTER TABLE DROP INDEX.

```
DROP INDEX ivfpq_vector ON ivfpq;
ALTER TABLE ivfpq DROP INDEX ivfpq_vector;
```

Выполнение ANNS с vector индексами

Перед запуском векторного поиска убедитесь, что параметр конфигурации FE `enable_experimental_vector` установлен в `true`.

Требования для запросов на основе vector индекса

```
SELECT *, <vector_index_distance_func>(v1, [1,2,3]) as dis
FROM table_name
WHERE <vector_index_distance_func>(v1, [1,2,3]) <= 10
ORDER BY <vector_index_distance_func>(v1, [1,2,3])
LIMIT 10
```

Чтобы использовать vector индекс в запросах, все следующие требования должны быть выполнены:

- **Требования ORDER BY:**
  - Формат предложения ORDER BY: Предложение `ORDER BY` должно следовать формату `ORDER BY <vector_index_distance_func>(vector_column, constant_array)` без включения дополнительных столбцов ORDER BY.
    - Требования к имени функции для `<vector_index_distance_func>`:
      - Если `metric_type` равен `l2_distance`, имя функции должно быть `approx_l2_distance`.
      - Если `metric_type` равен `cosine_similarity`, имя функции должно быть `approx_cosine_similarity`.
    - Требования к параметрам для `<vector_index_distance_func>`:
      - Один из столбцов `constant_array` должен быть константным `ARRAY<FLOAT>` с размерностью, соответствующей `dim` vector индекса.
      - Другой столбец `vector_column` должен быть столбцом, соответствующим vector индексу.
  - Требования к направлению ORDER:
    - Если `metric_type` равен `l2_distance`, порядок должен быть `ASC`.
    - Если `metric_type` равен `cosine_similarity`, порядок должен быть `DESC`.
  - Требуется предложение `LIMIT N`.
- **Требования к предикатам:**
  - Все предикаты должны быть выражениями `<vector_index_distance_func>`, объединёнными с использованием `AND` и операторов сравнения (`>` или `<`). Направление оператора сравнения должно совпадать с порядком `ASC`/`DESC`. В частности:
  - Требование 1:
    - Если `metric_type` равен `l2_distance`: `col_ref <= constant`.
    - Если `metric_type` равен `cosine_similarity`: `col_ref >= constant`.
    - Здесь `col_ref` относится к результату `<vector_index_distance_func>(vector_column, constant_array)` и может быть приведён к типам `FLOAT` или `DOUBLE`, например:
      - `approx_l2_distance(v1, [1,2,3])`
      - `CAST(approx_l2_distance(v1, [1,2,3]) AS FLOAT)`
      - `CAST(approx_l2_distance(v1, [1,2,3]) AS DOUBLE)`
  - Требование 2:

- Предикат должен использовать `AND`, где каждый дочерний предикат соответствует Требованию 1.

Подготовка

Создайте таблицы с vector индексами и вставьте векторные данные в них:

```
CREATE TABLE test_hnsw (
    id     BIGINT(20)   NOT NULL COMMENT "",
    vector ARRAY<FLOAT> NOT NULL COMMENT "",
    INDEX index_vector (vector) USING VECTOR (
        "index_type" = "hnsw",
        "metric_type" = "l2_distance",
        "is_vector_normed" = "false",
        "M" = "512",
        "dim"="5")
) ENGINE=OLAP
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1;

INSERT INTO test_hnsw VALUES
    (1, [1,2,3,4,5]),
    (2, [4,5,6,7,8]);

CREATE TABLE test_ivfpq (
    id     BIGINT(20)   NOT NULL COMMENT "",
    vector ARRAY<FLOAT> NOT NULL COMMENT "",
    INDEX index_vector (vector) USING VECTOR (
        "index_type" = "ivfpq",
        "metric_type" = "l2_distance",
        "is_vector_normed" = "false",
        "nlist" = "256",
        "nbits"="8",
        "dim"="5",
        "M_IVFPQ"="1")
) ENGINE=OLAP
DUPLICATE KEY(id)
DISTRIBUTED BY HASH(id) BUCKETS 1;

INSERT INTO test_ivfpq VALUES
    (1, [1,2,3,4,5]),
    (2, [4,5,6,7,8]);
```

Выполнение векторного поиска

Приблизительный поиск

Приблизительные поиски будут использовать vector индекс и тем самым ускорят процесс поиска.

Следующий пример ищет топ-1 приблизительного ближайшего соседа вектора `[1,1,1,1,1]`.

```
SELECT id, approx_l2_distance([1,1,1,1,1], vector)
FROM test_hnsw
ORDER BY approx_l2_distance([1,1,1,1,1], vector)
LIMIT 1;
```

Совместный скалярно-векторный поиск

Вы можете комбинировать скалярный поиск с векторным поиском.

```
SELECT id, approx_l2_distance([1,1,1,1,1], vector)
FROM test_hnsw
WHERE id = 1
ORDER BY approx_l2_distance([1,1,1,1,1], vector)
LIMIT 1;
```

Поиск по диапазону

Вы можете выполнять поиск по диапазону векторных данных.

Следующий пример отправляет условие `score < 40` вниз к индексу и фильтрует векторы по диапазону `score`.

```
SELECT * FROM (
    SELECT id, approx_l2_distance([1,1,1,1,1], vector) score
    FROM test_hnsw
) a
WHERE score < 40
ORDER BY score
LIMIT 1;
```

Точный расчёт

Точные расчёты проигнорируют vector индекс и будут напрямую вычислять расстояния между векторами для точных результатов.

```
SELECT id, l2_distance([1,1,1,1,1], vector)
FROM test_hnsw WHERE id = 1
ORDER BY l2_distance([1,1,1,1,1], vector)
LIMIT 1;
```

> **ПРИМЕЧАНИЕ**
>
> Различные функции метрик расстояния определяют "сходство" по-разному. Для `l2_distance` меньшие значения указывают на более высокое сходство; для `cosine_similarity` большие значения указывают на более высокое сходство. Следовательно, при вычислении `topN` направление сортировки (ORDER BY) должно соответствовать направлению сходства метрики. Используйте `ORDER BY ASC LIMIT x` для `l2_distance` и `ORDER BY DESC LIMIT x` для `cosine_similarity`.

Тонкая настройка параметров индекса для поиска

Настройка параметров важна в векторном поиске, поскольку влияет как на производительность, так и на точность. Рекомендуется настраивать параметры поиска на небольшом наборе данных и переходить к большим наборам данных только когда достигнуты ожидаемые recall и задержка.

Параметры поиска передаются через подсказки в операторах SQL.

Для HNSW индекса

Перед продолжением убедитесь, что столбец вектора построен с HNSW индексом.

```
SELECT
    /*+ SET_VAR (ann_params='{efsearch=256}') */
    id, approx_l2_distance([1,1,1,1,1], vector)
FROM test_hnsw
WHERE id = 1
ORDER BY approx_l2_distance([1,1,1,1,1], vector)
LIMIT 1;
```

Параметр:

efsearch

- **По умолчанию**: 16
- **Обязательно**: Нет
- **Описание**: Параметр, контролирующий компромисс точности и скорости. Во время поиска иерархической структуры графа этот параметр контролирует размер списка кандидатов во время поиска. Чем больше значение `efsearch`, тем выше точность, но тем ниже скорость.

Для IVFPQ индекса

Перед продолжением убедитесь, что столбец вектора построен с IVFPQ индексом.

```
SELECT
    /*+ SET_VAR
(ann_params='{nprobe=256,max_codes=0,scan_table_threshold=0,polysemous_ht=0,r
*/
    id, approx_l2_distance([1,1,1,1,1], vector)
FROM test_ivfpq
ORDER BY approx_l2_distance([1,1,1,1,1], vector)
LIMIT 1;
```

Параметр:

nprobe

- **По умолчанию**: 1
- **Обязательно**: Нет
- **Описание**: Количество inverted списков, проверяемых во время поиска. Чем больше значение `nprobe`, тем выше точность, но тем ниже скорость.

max_codes

- **По умолчанию**: 0
- **Обязательно**: Нет
- **Описание**: Максимальное количество кодов, проверяемых на inverted список. Этот параметр также будет влиять на точность и скорость.

scan_table_threshold

- **По умолчанию**: 0
- **Обязательно**: Нет

- **Описание**: Параметр, контролирующий polysemous hashing. Когда расстояние Хэмминга между хешем элемента и хешем вектора для поиска ниже этого порога, элемент добавляется в список кандидатов.

polysemous_ht

- **По умолчанию**: 0
- **Обязательно**: Нет
- **Описание**: Параметр, контролирующий polysemous hashing. Когда расстояние Хэмминга между хешем элемента и хешем вектора для поиска ниже этого порога, элемент напрямую добавляется в результаты.

range_search_confidence

- **По умолчанию**: 0.1
- **Обязательно**: Нет
- **Описание**: Уверенность приблизительных поисков по диапазону. Диапазон значений: [0, 1]. Установка в `1` даёт наиболее точные результаты.

Расчёт приблизительного recall

Вы можете рассчитать приблизительный recall, пересекая элементы `topK` из поиска методом грубой силы с элементами из приблизительного поиска: `Recall = TP / (TP + FN)`.

```
-- Приблизительный поиск
SELECT id
FROM test_hnsw
ORDER BY approx_l2_distance([1,1,1,1,1], vector)
LIMIT 5;
8
9
7
5
1

-- Поиск методом грубой силы
SELECT id
FROM test_hnsw
ORDER BY l2_distance([1,1,1,1,1], vector)
LIMIT 5;
8
9
5
7
10
```

В предыдущем примере приблизительный поиск возвращает 8, 9, 7 и 5. Однако правильный результат — 8, 9, 5, 7 и 10. В этом случае recall составляет 4/5=80%.

Проверка, работают ли vector индексы

Выполните EXPLAIN для оператора запроса. Если свойства `OlapScanNode` показывают `VECTORINDEX: ON`, это указывает, что vector индекс был применён к приблизительным векторным поискам.

Пример:

```
> EXPLAIN SELECT id FROM t_test_vector_table ORDER BY
approx_l2_distance([1,1,1,1,1], vector) LIMIT 5;

+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
+
| Explain String
|
+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
+
| PLAN FRAGMENT 0
|
|  OUTPUT EXPRS:1: id
|
|   PARTITION: UNPARTITIONED
|
|
|
|   RESULT SINK
|
|
|
|   4:Project
|
|   |  <slot 1> : 1: id
|
|   |  limit: 5
|
|   |
|
|   3:MERGING-EXCHANGE
|
|      limit: 5
|
|
|
| PLAN FRAGMENT 1
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|
|
```

```
|   STREAM DATA SINK
|
|     EXCHANGE ID: 03
|
|     UNPARTITIONED
|
|
|
|   2:TOP-N
|
|   |  order by: <slot 3> 3: approx_l2_distance ASC
|
|   |  offset: 0
|
|   |  limit: 5
|
|   |
|
|   1:Project
|
|   |  <slot 1> : 1: id
|
|   |  <slot 3> : 4: __vector_approx_l2_distance
|
|   |
|
|   0:OlapScanNode
|
|      TABLE: t_test_vector_table
|
|      VECTORINDEX: ON
|
|           IVFPQ: OFF, Distance Column: <4:__vector_approx_l2_distance>,
LimitK: 5, Order: ASC, Query Vector: [1, 1, 1, 1, 1], Predicate Range: -1.0
|
|      PREAGGREGATION: ON
|
|      partitions=1/1
|
|      rollup: t_test_vector_table
|
|      tabletRatio=1/1
|
|      tabletList=11302
|
|      cardinality=2
```

```
|
|      avgRowSize=4.0
|
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
+
```

Ограничения

- Каждая таблица поддерживает только один vector индекс.

2.7 Гибридное строчно-столбцовое хранилище

> *Beta* **BETA FEATURE**
>
> Рекомендации по использованию бета-функций

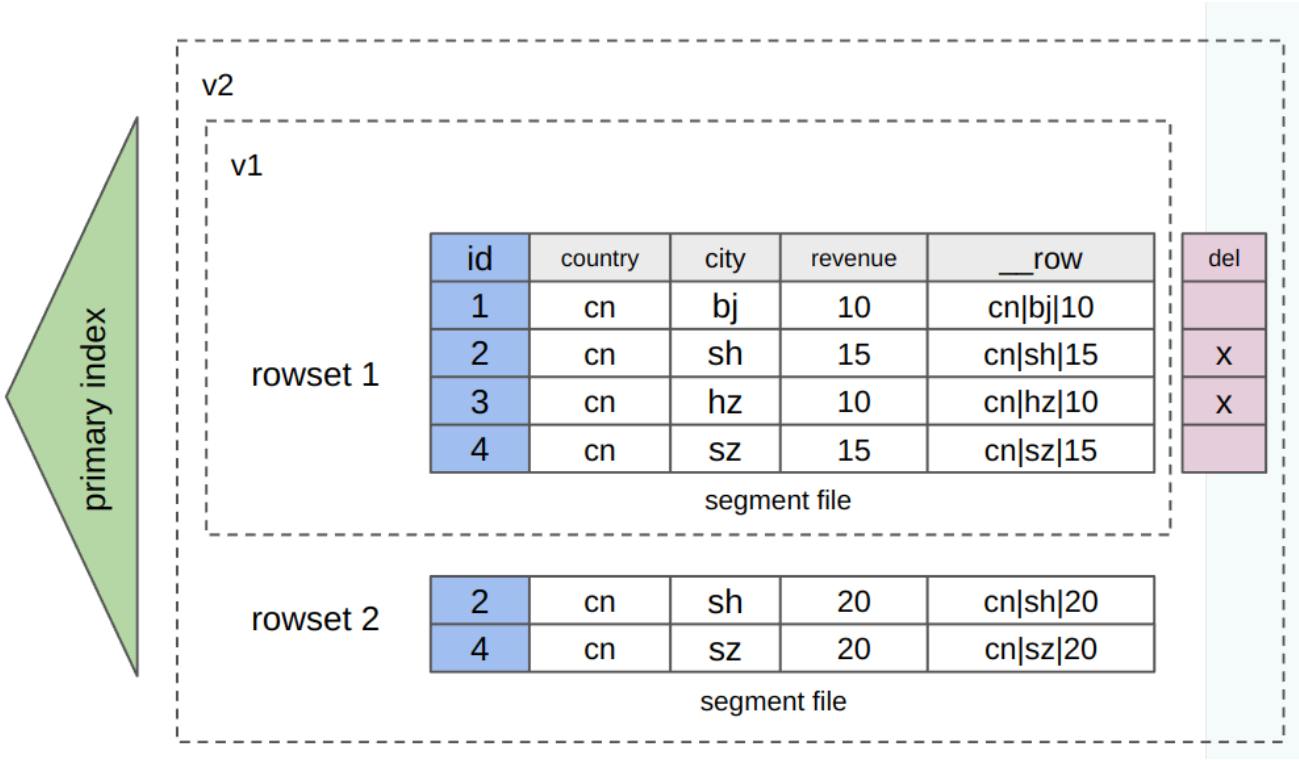
Как OLAP база данных, Selena изначально хранит данные в столбцовом формате, что может повысить производительность сложных запросов, таких как агрегатные запросы. Начиная с версии v1.5.2, Selena также поддерживает хранение данных в гибридном строчно-столбцовом формате, где данные хранятся как построчно, так и по столбцам. Это гибридное строчно-столбцовое хранилище хорошо подходит для различных сценариев, таких как высокая конкурентность точечных запросов с низкой задержкой на основе primary key и частичные обновления столбцов, при этом обеспечивая эффективные аналитические возможности, сопоставимые со столбцовым хранилищем. Кроме того, гибридное строчно-столбцовое хранилище поддерживает prepared statements, что повышает производительность и безопасность запросов.

2.7.1 Сравнение между столбцовым хранилищем и гибридным строчно-столбцовым хранилищем

Гибридное строчно-столбцовое хранилище

- Метод хранения: Данные хранятся как построчно, так и по столбцам. Проще говоря, таблица, использующая гибридное строчно-столбцовое хранилище, содержит дополнительный скрытый столбец бинарного типа `__row`. Когда данные записываются в таблицу, все значения из столбцов значений каждой задействованной строки кодируются и записываются в столбец `__row` (как показано ниже). Поскольку

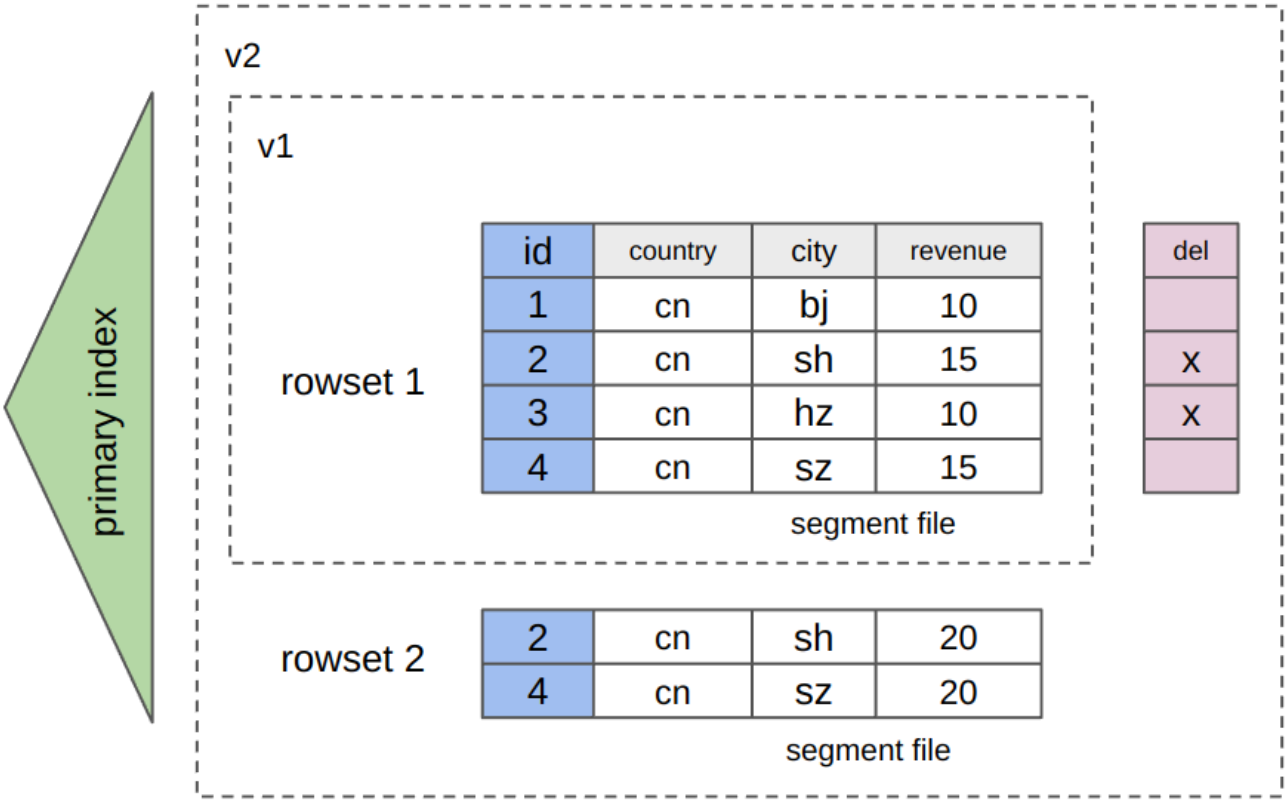
данные хранятся как построчно, так и по столбцам, возникают дополнительные затраты на хранение.



- Сценарии: поддерживает пользовательские сценарии как построчного, так и постолбцового хранения, но влечет дополнительные затраты на хранение.
    - Пользовательские сценарии построчного хранения:
        - Высококонкурентные точечные запросы на основе primary keys.
        - Запросы к большинству полей из таблиц, состоящих из небольшого количества полей.
        - Частичные обновления столбцов (более конкретно, несколько столбцов и небольшое количество строк данных требуется обновить)
    - Пользовательские сценарии постолбцового хранения: Сложный анализ данных.

Столбцовое хранилище

- Метод хранения: Данные хранятся по столбцам.



- Сценарии: Сложный анализ данных.
  - Сложные запросы и анализ массивных наборов данных, такие как агрегатный анализ и запросы с объединением нескольких таблиц.
  - Таблицы состоят из множества полей (например, широкие таблицы), но запросы к этим таблицам включают только несколько столбцов.

2.7.2 Базовое использование

Создание таблицы, использующей гибридное строчно-столбцовое хранилище

1. Включите конфигурационный параметр FE `enable_experimental_rowstore`.

```
ADMIN SET FRONTEND CONFIG ("enable_experimental_rowstore" = "true");
```

2. Укажите `"STORE_TYPE" = "column_with_row"` в `PROPERTIES` при создании таблицы.

> **ПРИМЕЧАНИЕ**
>
> - Таблица должна быть таблицей Primary Key.
> - Длина столбца `__row` не может превышать 1 МБ.
> - Начиная с v1.5.2, Selena расширяет поддержку следующих типов столбцов: BITMAP, HLL, JSON, ARRAY, MAP и STRUCT.

```
CREATE TABLE users (
  id bigint not null,
  country string,
  city string,
  revenue bigint
)
PRIMARY KEY (id)
DISTRIBUTED by HASH (id)
PROPERTIES ("store_type" = "column_with_row");
```

Вставка, удаление и обновление данных

Аналогично таблице, использующей столбцовое хранилище, вы можете вставлять, удалять и обновлять данные в таблице, использующей гибридное строчно-столбцовое хранилище, путем загрузки данных и операторов DML. В этом разделе демонстрируется, как выполнять операторы DML на вышеуказанной таблице с гибридным строчно-столбцовым хранилищем.

1. Вставка строки данных.

   i.
   ```
   INSERT INTO users (id, country, city, revenue)
   VALUES
     (1, 'USA', 'New York', 5000),
     (2, 'UK', 'London', 4500),
     (3, 'France', 'Paris', 6000),
     (4, 'Germany', 'Berlin', 4000),
     (5, 'Japan', 'Tokyo', 7000),
     (6, 'Australia', 'Sydney', 7500);
   ```

2. Удаление строки данных.

```
DELETE FROM users WHERE id = 6;
```

3. Обновление строки данных.

```
UPDATE users SET revenue = 6500 WHERE id = 4;
```

Запрос данных

В этом разделе используется точечный запрос в качестве примера. Точечные запросы используют короткий путь, напрямую запрашивая данные в строчном хранилище, что может улучшить производительность запросов.

В следующем примере по-прежнему используется вышеуказанная таблица с гибридным строчно-столбцовым хранилищем. После создания таблицы и операций модификации данных, упомянутых выше, таблица хранит данные следующим образом:

```
MySQL [example_db]> SELECT * FROM users ORDER BY id;
+------+---------+----------+---------+
| id   | country | city     | revenue |
+------+---------+----------+---------+
|    1 | USA     | New York |    5000 |
|    2 | UK      | London   |    4500 |
|    3 | France  | Paris    |    6000 |
|    4 | Germany | Berlin   |    6500 |
|    5 | Japan   | Tokyo    |    7000 |
+------+---------+----------+---------+
5 rows in set (0.03 sec)
```

1. Убедитесь, что система включает короткий путь для запросов. После включения короткого пути для запросов, запросы, соответствующие определенным критериям (для оценки, является ли запрос точечным запросом), используют короткий путь для сканирования данных в строчном хранилище.

   ```
   SHOW VARIABLES LIKE '%enable_short_circuit%';
   ```

   Если короткий путь для запросов не включен, выполните команду `SET enable_short_circuit = true;`, чтобы установить переменную

`enable_short_circuit` в `true`.

2. Запрос данных. Если запрос соответствует критериям, что условные столбцы в предложении WHERE включают все столбцы primary key, и операторы в предложении WHERE являются `=` или `IN`, запрос использует короткий путь.

> **ПРИМЕЧАНИЕ**
>
> Условные столбцы в предложении WHERE могут включать дополнительные столбцы помимо всех столбцов primary key.

```
SELECT * FROM users WHERE id=1;
```

3. Проверьте план запроса, чтобы убедиться, может ли запрос использовать короткий путь. Если план запроса включает `Short Circuit Scan: true`, запрос может использовать короткий путь.

```
MySQL [example_db]> EXPLAIN SELECT * FROM users WHERE id=1;
+-----------------------------------------------------------+
| Explain String                                            |
+-----------------------------------------------------------+
| PLAN FRAGMENT 0                                           |
|  OUTPUT EXPRS:1: id | 2: country | 3: city | 4: revenue |
|   PARTITION: RANDOM                                       |
|                                                           |
|   RESULT SINK                                             |
|                                                           |
|   0:OlapScanNode                                          |
|      TABLE: users                                         |
|      PREAGGREGATION: OFF. Reason: null                    |
|      PREDICATES: 1: id = 1                                |
|      partitions=1/1                                       |
|      rollup: users                                        |
|      tabletRatio=1/6                                      |
|      tabletList=10184                                     |
|      cardinality=-1                                       |
|      avgRowSize=0.0                                       |
|      Short Circuit Scan: true                             | -- Запрос
может использовать короткий путь.
+-----------------------------------------------------------+
17 rows in set (0.00 sec)
```

Использование prepared statements

Вы можете использовать prepared statements для запроса данных в таблицах с гибридным строчно-столбцовым хранилищем.

```
-- Подготовка операторов для выполнения.
PREPARE select_all_stmt FROM 'SELECT * FROM users';
PREPARE select_by_id_stmt FROM 'SELECT * FROM users WHERE id = ?';

-- Объявление переменных в операторах.
SET @id1 = 1, @id2 = 2;

-- Использование объявленных переменных для выполнения операторов.
-- Запрос данных с ID 1 или 2 отдельно.
EXECUTE select_by_id_stmt USING @id1;
EXECUTE select_by_id_stmt USING @id2;
```

2.7.3 Ограничения

- В настоящее время shared-data кластеры Selena не поддерживают гибридное строчно-столбцовое хранилище.
- Начиная с v1.5.2, таблица с гибридным строчно-столбцовым хранилищем может быть изменена с использованием ALTER TABLE.
- Короткий путь для запросов в настоящее время подходит только для запросов, которые происходят после запланированной пакетной загрузки данных. Поскольку может возникнуть взаимное исключение индексов, когда короткий путь для запросов происходит на этапе применения процесса записи данных, запись данных может блокировать короткий путь для запросов, влияя на время отклика точечных запросов во время записи данных.
- Гибридное строчно-столбцовое хранилище может значительно увеличить потребление хранилища. Это связано с тем, что данные хранятся как в строчном, так и в столбцовом форматах, и коэффициент сжатия данных строчного хранилища может быть не таким высоким, как у столбцового хранилища.
- Гибридное строчно-столбцовое хранилище может увеличить время и потребление ресурсов во время загрузки данных.
- Таблицы с гибридным строчно-столбцовым хранилищем могут быть жизнеспособным решением для онлайн-сервисов, но производительность этого типа таблиц может не конкурировать со зрелыми OLTP базами данных.
- Таблицы с гибридным строчно-столбцовым хранилищем не поддерживают функции, которые зависят от столбцового хранилища, такие как частичные обновления в режиме столбцов.
- Таблицы с гибридным строчно-столбцовым хранилищем должны быть таблицами Primary Key.

3 Загрузка данных

3.1 Обзор загрузки

3.1.1 Варианты загрузки

Загрузка данных — это процесс очистки и преобразования необработанных данных из различных источников данных в соответствии с вашими бизнес-требованиями и загрузки результирующих данных в Selena для облегчения анализа.

Selena предоставляет множество вариантов загрузки данных:

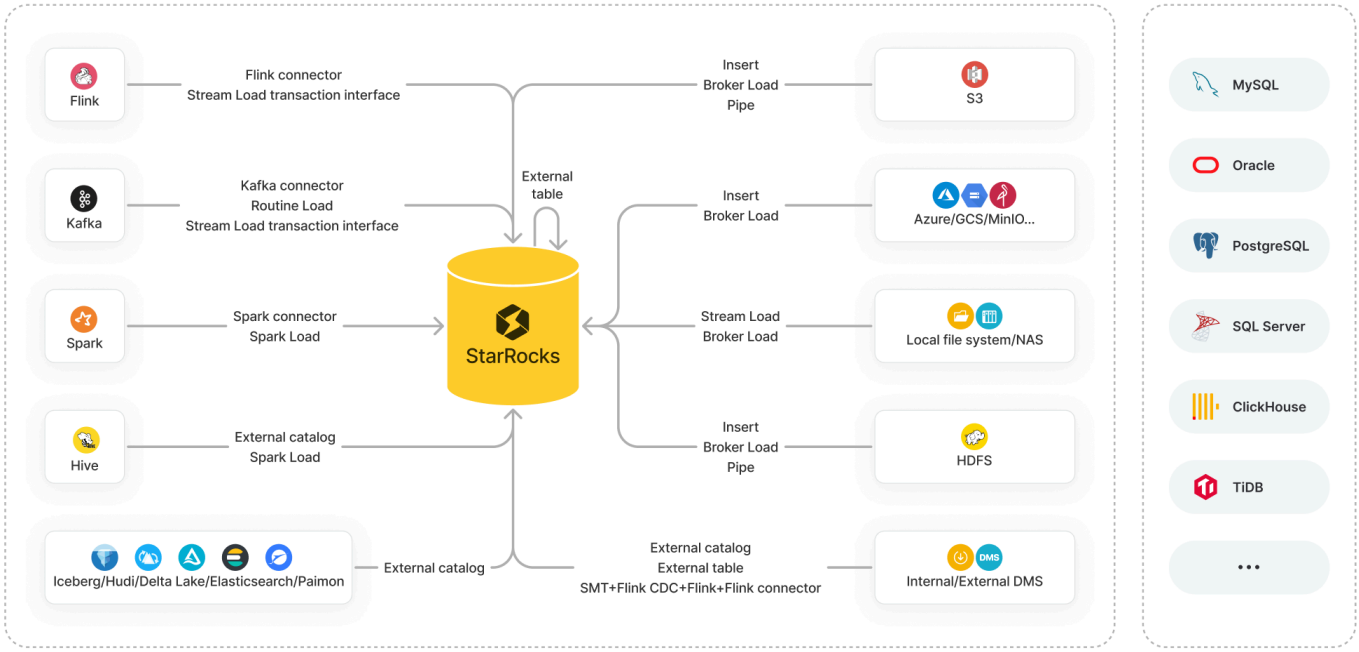
- Методы загрузки: Insert, Stream Load, Broker Load, Pipe, Routine Load и Spark Load
- Инструменты экосистемы: Selena Connector for Apache Kafka® (сокращённо Kafka connector), Selena Connector for Apache Spark™ (сокращённо Spark connector), Selena Connector for Apache Flink® (сокращённо Flink connector) и другие инструменты, такие как SMT, DataX, CloudCanal и Kettle Connector
- API: Stream Load transaction interface

Каждый из этих вариантов имеет свои преимущества и поддерживает свой набор систем источников данных для извлечения данных.

В этом разделе представлен обзор этих вариантов, а также сравнения между ними, чтобы помочь вам определить вариант загрузки по вашему выбору на основе источника данных, бизнес-сценария, объёма данных, формата файла данных и частоты загрузки.

Введение в варианты загрузки

В этом разделе в основном описываются характеристики и бизнес-сценарии вариантов загрузки, доступных в Selena.



> **ПРИМЕЧАНИЕ**
>
> В следующих разделах "пакетная" или "пакетная загрузка" относится к загрузке большого объёма данных из указанного источника за один раз в Selena, тогда как "потоковая" или "потоковая передача" относится к непрерывной загрузке данных в реальном времени.

Методы загрузки

Insert

Бизнес-сценарий:

- INSERT INTO VALUES: Добавление небольших объёмов данных во внутреннюю таблицу.
- INSERT INTO SELECT:
  - INSERT INTO SELECT FROM `<table_name>`: Добавление в таблицу результата запроса к внутренней или внешней таблице.
  - INSERT INTO SELECT FROM FILES(): Добавление в таблицу результата запроса к файлам данных в удалённом хранилище.

> **ПРИМЕЧАНИЕ**
>
> Для AWS S3 эта функция поддерживается начиная с версии v1.5.2. Для HDFS, Microsoft Azure Storage, Google GCS и S3-совместимого хранилища (такого как MinIO) эта функция поддерживается начиная с версии v1.5.2.

Формат файла:

- INSERT INTO VALUES: SQL
- INSERT INTO SELECT:
  - INSERT INTO SELECT FROM `<table_name>`: Таблицы Selena
  - INSERT INTO SELECT FROM FILES(): Parquet и ORC

Объём данных: Не фиксирован (Объём данных зависит от размера памяти.)

Stream Load

Бизнес-сценарий: Пакетная загрузка данных из локальной файловой системы.

Формат файла: CSV и JSON

Объём данных: 10 ГБ или менее

Broker Load

Бизнес-сценарий:

- Пакетная загрузка данных из HDFS или облачного хранилища, такого как AWS S3, Microsoft Azure Storage, Google GCS и S3-совместимое хранилище (такое как MinIO).
- Пакетная загрузка данных из локальной файловой системы или NAS.

Формат файла: CSV, Parquet, ORC и JSON (поддерживается начиная с v1.5.2)

Объём данных: От десятков ГБ до сотен ГБ

Pipe

Бизнес-сценарий: Пакетная или потоковая загрузка данных из HDFS или AWS S3.

> **ПРИМЕЧАНИЕ**
>
> Этот метод загрузки поддерживается начиная с версии v1.5.2.

Формат файла: Parquet и ORC

Объём данных: От 100 ГБ до 1 ТБ или более

Routine Load

Бизнес-сценарий: Потоковая загрузка данных из Kafka.

Формат файла: CSV, JSON и Avro (поддерживается начиная с v1.5.2)

Объём данных: От МБ до ГБ данных в виде мини-пакетов

Spark Load

Бизнес-сценарий: Пакетная загрузка данных таблиц Apache Hive™, хранящихся в HDFS, с использованием cluster Spark.

Формат файла: CSV, Parquet (поддерживается начиная с v1.5.2) и ORC (поддерживается начиная с v1.5.2)

Объём данных: От десятков ГБ до ТБ

Инструменты экосистемы

Kafka connector

Бизнес-сценарий: Потоковая загрузка данных из Kafka.

Spark connector

Бизнес-сценарий: Пакетная загрузка данных из Spark.

Flink connector

Бизнес-сценарий: Потоковая загрузка данных из Flink.

SMT

Бизнес-сценарий: Загрузка данных из источников данных, таких как MySQL, PostgreSQL, SQL Server, Oracle, Hive, ClickHouse и TiDB через Flink.

DataX

Бизнес-сценарий: Синхронизация данных между различными гетерогенными источниками данных, включая реляционные базы данных (например, MySQL и Oracle), HDFS и Hive.

CloudCanal

Бизнес-сценарий: Миграция или синхронизация данных из исходных баз данных (например, MySQL, Oracle и PostgreSQL) в Selena.

Kettle Connector

Бизнес-сценарий: Интеграция с Kettle. Объединяя мощные возможности обработки и преобразования данных Kettle с высокопроизводительными возможностями хранения и анализа данных Selena, можно достичь более гибких и эффективных рабочих процессов обработки данных.

API

Stream Load transaction interface

Бизнес-сценарий: Реализация двухфазной фиксации (2PC) для транзакций, которые выполняются для загрузки данных из внешних систем, таких как Flink и Kafka, при одновременном повышении производительности высококонкурентных потоковых загрузок. Эта функция поддерживается начиная с версии v1.5.2.

Формат файла: CSV и JSON

Объём данных: 10 ГБ или менее

Выбор вариантов загрузки

В этом разделе перечислены варианты загрузки, доступные для распространённых источников данных, что поможет вам выбрать вариант, который лучше всего соответствует вашей ситуации.

Объектное хранилище

Источник данных	Доступные варианты загрузки
AWS S3	• (Пакетная) INSERT INTO SELECT FROM FILES() (поддерживается начиная с v1.5.2) • (Пакетная) Broker Load • (Пакетная или потоковая) Pipe (поддерживается начиная с v1.5.2) См. Загрузка данных из AWS S3.
Microsoft Azure Storage	• (Пакетная) INSERT INTO SELECT FROM FILES() (поддерживается начиная с v1.5.2) • (Пакетная) Broker Load См. Загрузка данных из Microsoft Azure Storage.
Google GCS	• (Пакетная) INSERT INTO SELECT FROM FILES() (поддерживается начиная с v1.5.2) • (Пакетная) Broker Load См. Загрузка данных из GCS.
S3-совместимое хранилище (такое как MinIO)	• (Пакетная) INSERT INTO SELECT FROM FILES() (поддерживается начиная с v1.5.2) • (Пакетная) Broker Load См. Загрузка данных из MinIO.

Локальная файловая система (включая NAS)

Источник данных	Доступные варианты загрузки
Локальная файловая система (включая NAS)	• (Пакетная) Stream Load • (Пакетная) Broker Load См. Загрузка данных из локальной файловой системы.

HDFS

Источник данных	Доступные варианты загрузки
HDFS	• (Пакетная) INSERT INTO SELECT FROM FILES() (поддерживается начиная с v1.5.2) • (Пакетная) Broker Load • (Пакетная или потоковая) Pipe (поддерживается начиная с v1.5.2) См. Загрузка данных из HDFS.

Flink, Kafka и Spark

Источник данных	Доступные варианты загрузки
Apache Flink®	• Flink connector • Stream Load transaction interface
Apache Kafka®	• (Потоковая) Kafka connector • (Потоковая) Routine Load • Stream Load transaction interface **ПРИМЕЧАНИЕ** Если исходные данные требуют соединений нескольких таблиц и операций извлечения, преобразования и загрузки (ETL), вы можете использовать Flink для чтения и предварительной обработки данных, а затем использовать Flink connector для загрузки данных в Selena.
Apache Spark™	• Spark connector • Spark Load

Озёра данных

Источник данных	Доступные варианты загрузки
Apache Hive™	• (Пакетная) Создайте Hive catalog, а затем используйте INSERT INTO SELECT FROM `<table_name>`. • (Пакетная) Spark Load.
Apache Iceberg	(Пакетная) Создайте Iceberg catalog, а затем используйте INSERT INTO SELECT FROM `<table_name>`.
Apache Hudi	(Пакетная) Создайте Hudi catalog, а затем используйте INSERT INTO SELECT FROM `<table_name>`.
Delta Lake	(Пакетная) Создайте Delta Lake catalog, а затем используйте INSERT INTO SELECT FROM `<table_name>`.
Elasticsearch	(Пакетная) Создайте Elasticsearch catalog, а затем используйте INSERT INTO SELECT FROM `<table_name>`.
Apache Paimon	(Пакетная) Создайте Paimon catalog, а затем используйте INSERT INTO SELECT FROM `<table_name>`.

Обратите внимание, что Selena предоставляет unified catalog начиная с версии v1.5.2, чтобы помочь вам работать с таблицами из источников данных Hive, Iceberg, Hudi и Delta Lake как с единым источником данных без загрузки.

Внутренние и внешние базы данных

Источник данных	Доступные варианты загрузки
Selena	(Пакетная) Создайте внешнюю таблицу Selena, а затем используйте INSERT INTO VALUES для вставки нескольких записей данных или INSERT INTO SELECT FROM `<table_name>` для вставки данных таблицы. **ПРИМЕЧАНИЕ** Внешние таблицы Selena поддерживают только запись данных. Они не поддерживают чтение данных.
MySQL	• (Пакетная) Создайте JDBC catalog (рекомендуется) или внешнюю таблицу MySQL, а затем используйте INSERT INTO SELECT FROM `<table_name>`. • (Потоковая) Используйте SMT, Flink CDC connector, Flink и Flink connector.
Другие базы данных, такие как Oracle, PostgreSQL, SQL Server, ClickHouse и TiDB	• (Пакетная) Создайте JDBC catalog (рекомендуется) или внешнюю таблицу JDBC, а затем используйте INSERT INTO SELECT FROM `<table_name>`. • (Потоковая) Используйте SMT, Flink CDC connector, Flink и Flink connector.

3.1.2 Концепции загрузки

В этой теме представлены общие концепции и информация о загрузке данных.

Привилегии

Вы можете загружать данные в таблицы Selena только как пользователь, имеющий привилегию INSERT на эти таблицы Selena. Если у вас нет привилегии INSERT, следуйте инструкциям в разделе GRANT, чтобы предоставить привилегию INSERT пользователю, которого вы используете для подключения к вашему cluster Selena. Синтаксис: `GRANT INSERT ON TABLE <table_name> IN DATABASE <database_name> TO { ROLE <role_name> | USER <user_identity>}`.

Маркировка

Вы можете загружать данные в Selena, запуская задания загрузки. Каждое задание загрузки имеет уникальную метку, которая указывается пользователем или автоматически генерируется Selena для идентификации задания. Каждая метка может использоваться только для одного задания загрузки. После завершения задания загрузки его метку нельзя повторно использовать для других заданий загрузки. Только метки неудавшихся заданий загрузки могут быть использованы повторно.

Атомарность

Все методы загрузки, предоставляемые Selena, гарантируют атомарность. Атомарность означает, что квалифицированные данные в рамках задания загрузки должны быть либо полностью успешно загружены, либо ни одни квалифицированные данные не загружаются успешно. Никогда не бывает так, что часть квалифицированных данных загружена, а другая — нет. Обратите внимание, что квалифицированные данные не включают данные, отфильтрованные из-за проблем с качеством, таких как ошибки преобразования типов данных.

Протоколы

Selena поддерживает два протокола связи, которые можно использовать для отправки заданий загрузки: MySQL и HTTP. Из всех методов загрузки, поддерживаемых Selena, только Stream Load использует HTTP, тогда как все остальные используют MySQL.

Типы данных

Selena поддерживает загрузку данных всех типов. Вам нужно обратить внимание только на ограничения при загрузке нескольких конкретных типов данных. Для получения дополнительной информации см. Типы данных.

Строгий режим

Строгий режим — это необязательное свойство, которое можно настроить для загрузки данных. Он влияет на поведение загрузки и окончательные загруженные данные. Подробнее см. Строгий режим.

Режимы загрузки

Selena поддерживает два режима загрузки: режим синхронной загрузки и режим асинхронной загрузки.

> **ПРИМЕЧАНИЕ**
>
> Если вы загружаете данные с помощью внешних программ, вы должны выбрать режим загрузки, который лучше всего соответствует вашим бизнес-требованиям, прежде чем выбрать метод загрузки.

Синхронная загрузка

В режиме синхронной загрузки после отправки задания загрузки Selena синхронно выполняет задание для загрузки данных и возвращает результат задания после его завершения. Вы можете проверить, успешно ли выполнено задание, на основе результата задания.

Selena предоставляет два метода загрузки, поддерживающих синхронную загрузку: Stream Load и INSERT.

Процесс синхронной загрузки выглядит следующим образом:

1. Создайте задание загрузки.
2. Просмотрите результат задания, возвращённый Selena.
3. Проверьте, успешно ли выполнено задание, на основе результата задания. Если результат задания указывает на сбой загрузки, вы можете повторить попытку выполнения задания.

Асинхронная загрузка

В режиме асинхронной загрузки после отправки задания загрузки Selena немедленно возвращает результат создания задания.

- Если результат указывает на успешное создание задания, Selena асинхронно выполняет задание. Однако это не означает, что данные были успешно загружены. Вы должны использовать операторы или команды для проверки состояния задания. Затем вы можете определить, успешно ли загружены данные, на основе состояния задания.
- Если результат указывает на сбой создания задания, вы можете определить, нужно ли повторить попытку выполнения задания, на основе информации о сбое.

© ООО «Платформа Данных Селена», 2026

ПОДСКАЗКА

Вы можете установить различные кворумы записи для таблиц, то есть сколько replica должны вернуть успех загрузки, прежде чем Selena определит, что задание загрузки успешно. Вы можете указать кворум записи, добавив свойство `write_quorum` при CREATE TABLE, или добавить это свойство к существующей таблице с помощью ALTER TABLE.

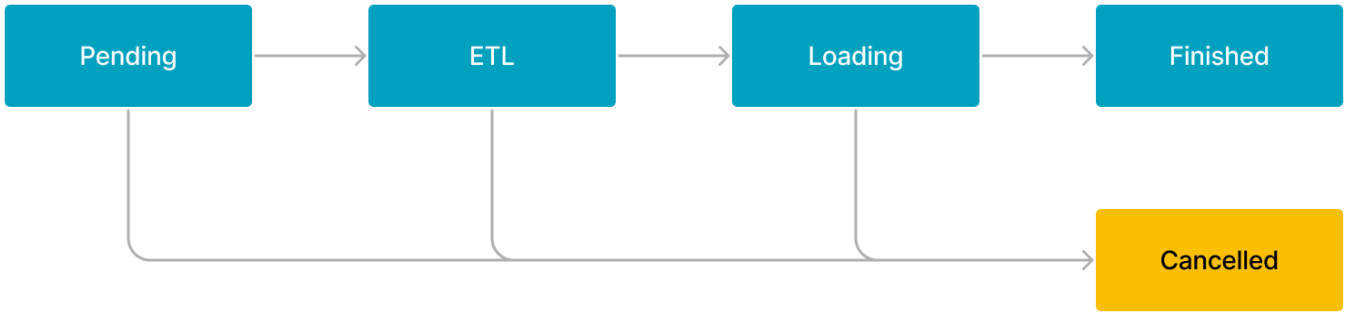
Selena предоставляет четыре метода загрузки, поддерживающих асинхронную загрузку: Broker Load, Pipe, Routine Load и Spark Load.

Процесс асинхронной загрузки выглядит следующим образом:

1. Создайте задание загрузки.
2. Просмотрите результат создания задания, возвращённый Selena, и определите, было ли задание успешно создано.
   - Если создание задания успешно, перейдите к шагу 3.
   - Если создание задания не удалось, вернитесь к шагу 1.
3. Используйте операторы или команды для проверки состояния задания, пока состояние задания не покажет **FINISHED** или **CANCELLED**.

Рабочий процесс Broker Load или Spark Load

Рабочий процесс задания Broker Load или Spark Load состоит из пяти этапов, как показано на следующем рисунке.



Рабочий процесс описан следующим образом:

1. **PENDING**

   Задание находится в очереди в ожидании планирования FE.

2. **ETL**

   FE предварительно обрабатывает данные, включая очистку, партиционирование, сортировку и агрегацию.

   Только задание Spark Load имеет этап ETL. Задание Broker Load пропускает этот этап.

3. **LOADING**

   FE очищает и трансформирует данные, а затем отправляет данные на BE или CN. После загрузки всех данных данные находятся в очереди в ожидании вступления в силу. В это время состояние задания остаётся **LOADING**.

4. **FINISHED**

   Когда загрузка завершается и все данные вступают в силу, состояние задания становится **FINISHED**. В это время данные можно запрашивать. **FINISHED** — это конечное состояние задания.

5. **CANCELLED**

   До того, как состояние задания станет **FINISHED**, вы можете отменить задание в любое время. Кроме того, Selena может автоматически отменить задание в случае ошибок загрузки. После отмены задания его состояние становится **CANCELLED**, и все обновления данных, сделанные до отмены, откатываются. **CANCELLED** также является конечным состоянием задания.

Рабочий процесс Pipe

Рабочий процесс задания Pipe описан следующим образом:

1. Задание отправляется на FE из клиента MySQL.

2. FE разделяет файлы данных, хранящиеся по указанному пути, на основе их количества или размера, разбивая задание на меньшие последовательные задачи.

Задачи попадают в очередь, ожидая планирования, после их создания.

3. FE получает задачи из очереди и вызывает оператор INSERT INTO SELECT FROM FILES для выполнения каждой задачи.

4. Загрузка данных завершается:

   - Если для задания при создании указано `"AUTO_INGEST" = "FALSE"`, задание завершается после загрузки данных из всех файлов данных, хранящихся по указанному пути.

   - Если для задания при создании указано `"AUTO_INGEST" = "TRUE"`, FE продолжит отслеживать изменения в файлах данных и автоматически загружать новые или обновлённые данные из файлов данных в целевую таблицу Selena.

Рабочий процесс Routine Load

Рабочий процесс задания Routine Load описан следующим образом:

1. Задание отправляется на FE из клиента MySQL.

2. FE разделяет задание на несколько задач. Каждая задача предназначена для загрузки данных из нескольких партиций.

3. FE распределяет задачи на указанные BE или CN.

4. BE или CN выполняют задачи и отчитываются FE после их завершения.

5. FE генерирует последующие задачи, повторяет неудавшиеся задачи, если таковые имеются, или приостанавливает планирование задач на основе отчётов от BE.

3.1.3 Рекомендации

В этой теме описаны некоторые системные ограничения и конфигурации, которые необходимо учитывать перед запуском загрузки данных.

Ограничения памяти

Selena предоставляет параметры для ограничения использования памяти для каждого задания загрузки, тем самым уменьшая потребление памяти, особенно в сценариях с высокой параллельностью. Однако не указывайте слишком низкий лимит использования

памяти. Если лимит использования памяти слишком низкий, данные могут часто сбрасываться из памяти на диск, потому что использование памяти для заданий загрузки достигает указанного лимита. Мы рекомендуем указывать подходящий лимит использования памяти в зависимости от вашего бизнес-сценария.

Параметры, используемые для ограничения использования памяти, различаются для каждого метода загрузки. Для получения дополнительной информации см. Stream Load, Broker Load, Routine Load, Spark Load и INSERT. Обратите внимание, что задание загрузки обычно выполняется на нескольких BE или CN. Поэтому параметры ограничивают использование памяти каждого задания загрузки на каждом задействованном BE или CN, а не общее использование памяти задания загрузки на всех задействованных BE или CN.

Selena также предоставляет параметры для ограничения общего использования памяти всеми заданиями загрузки, выполняющимися на каждом отдельном BE или CN. Для получения дополнительной информации см. раздел "Системные конфигурации" ниже.

Системные конфигурации

В этом разделе описаны некоторые конфигурации параметров, применимые ко всем методам загрузки, предоставляемым Selena.

Конфигурации FE

Вы можете настроить следующие параметры в файле конфигурации **fe.conf** каждого FE:

- `max_load_timeout_second` и `min_load_timeout_second`

  Эти параметры указывают максимальный и минимальный периоды тайм-аута каждого задания загрузки. Периоды тайм-аута измеряются в секундах. Максимальный период тайм-аута по умолчанию составляет 3 дня, а минимальный период тайм-аута по умолчанию — 1 секунду. Указанные максимальный и минимальный периоды тайм-аута должны находиться в диапазоне от 1 секунды до 3 дней. Эти параметры действительны как для синхронных, так и для асинхронных заданий загрузки.

- `desired_max_waiting_jobs`

  Этот параметр указывает максимальное количество заданий загрузки, которые могут удерживаться в очереди ожидания. Значение по умолчанию — **1024** (100 в версии v1.5.2 и ранее, и 1024 в версии v1.5.2 и позже). Когда количество заданий загрузки в

состоянии **PENDING** на FE достигает указанного максимального числа, FE отклоняет новые запросы загрузки. Этот параметр действителен только для асинхронных заданий загрузки.

- `max_running_txn_num_per_db`

  Этот параметр указывает максимальное количество текущих транзакций загрузки, разрешённых в каждой базе данных вашего cluster Selena. Задание загрузки может содержать одну или несколько транзакций. Значение по умолчанию — **100**. Когда количество транзакций загрузки, выполняющихся в базе данных, достигает указанного максимального числа, последующие отправленные задания загрузки не планируются. В этой ситуации, если вы отправляете синхронное задание загрузки, задание отклоняется. Если вы отправляете асинхронное задание загрузки, задание удерживается в очереди ожидания.

  > **ПРИМЕЧАНИЕ**
  >
  > Selena подсчитывает все задания загрузки вместе и не различает синхронные и асинхронные задания загрузки.

- `label_keep_max_second`

  Этот параметр указывает период хранения записей истории для заданий загрузки, которые завершились и находятся в состоянии **FINISHED** или **CANCELLED**. Период хранения по умолчанию составляет 3 дня. Этот параметр действителен как для синхронных, так и для асинхронных заданий загрузки.

Конфигурации BE/CN

Вы можете настроить следующие параметры в файле конфигурации **be.conf** каждого BE или файле конфигурации **cn.conf** каждого CN:

- `write_buffer_size`

  Этот параметр указывает максимальный размер блока памяти. Размер по умолчанию — 100 МБ. Загруженные данные сначала записываются в блок памяти на BE или CN. Когда объём загруженных данных достигает указанного максимального размера блока памяти, данные сбрасываются на диск. Вы должны указать подходящий максимальный размер блока памяти в зависимости от вашего бизнес-сценария.

- Если максимальный размер блока памяти слишком мал, на BE или CN может быть создано большое количество небольших файлов. В этом случае производительность запросов снижается. Вы можете увеличить максимальный размер блока памяти, чтобы уменьшить количество создаваемых файлов.
- Если максимальный размер блока памяти слишком велик, удалённые вызовы процедур (RPC) могут истечь по таймауту. В этом случае вы можете настроить значение этого параметра в зависимости от ваших бизнес-потребностей.

- `streaming_load_rpc_max_alive_time_sec`

  Период ожидания тайм-аута для каждого процесса Writer. Значение по умолчанию — 1200 секунд. Во время процесса загрузки данных Selena запускает процесс Writer для получения данных и записи данных в каждый tablet. Если процесс Writer не получает никаких данных в течение указанного периода ожидания тайм-аута, Selena останавливает процесс Writer. Когда ваш cluster Selena обрабатывает данные на низких скоростях, процесс Writer может не получить следующую партию данных в течение длительного периода времени и поэтому сообщает об ошибке "TabletWriter add batch with unknown id". В этом случае вы можете увеличить значение этого параметра.

- `load_process_max_memory_limit_bytes` и `load_process_max_memory_limit_percent`

  Эти параметры указывают максимальный объём памяти, который может быть использован для всех заданий загрузки на каждом отдельном BE или CN. Selena определяет меньшее потребление памяти между значениями двух параметров как окончательное разрешённое потребление памяти.

  - `load_process_max_memory_limit_bytes`: указывает максимальный размер памяти. Максимальный размер памяти по умолчанию — 100 ГБ.

  - `load_process_max_memory_limit_percent`: указывает максимальное использование памяти. Значение по умолчанию — 30%. Этот параметр отличается от параметра `mem_limit`. Параметр `mem_limit` указывает общее максимальное использование памяти вашего cluster Selena, и значение по умолчанию составляет 90% x 90%.

    Если ёмкость памяти машины, на которой находится BE или CN, равна M, максимальный объём памяти, который может быть использован для заданий

загрузки, рассчитывается следующим образом: `M x 90% x 90% x 30%`.

Конфигурации системных переменных

Вы можете настроить следующую системную переменную:

- `insert_timeout`

  Длительность тайм-аута INSERT. Единица измерения: секунды. Диапазон значений: от `1` до `259200`. Значение по умолчанию: `14400`. Эта переменная будет действовать на все операции, связанные с заданиями INSERT (например, UPDATE, DELETE, CTAS, обновление материализованного представления, сбор статистики и PIPE) в текущем соединении.

3.1.4 Поддержка функций: Загрузка и выгрузка данных

В этом документе описаны функции различных методов загрузки и выгрузки данных, поддерживаемых Selena.

Формат файла

Форматы файлов для загрузки

<table>
<tr><th rowspan="2"></th><th rowspan="2">Источник данных</th><th colspan="7">Формат файла</th></tr>
<tr><th>CSV</th><th>JSON [3]</th><th>Parquet</th><th>ORC</th><th>Avro</th><th>ProtoBuf</th><th>Thrift</th></tr>
<tr><td>Stream Load</td><td>Локальные файловые системы, приложения, коннекторы</td><td>Да</td><td>Да</td><td>Планируется</td><td>Планируется</td><td colspan="3">Планируется</td></tr>
<tr><td>INSERT from FILES</td><td rowspan="2">HDFS, S3, OSS, Azure, GCS, NFS(NAS) [5]</td><td>Да (v1.5.2+)</td><td>Планируется</td><td>Да (v1.5.2+)</td><td>Да (v1.5.2+)</td><td>Да (v1.5.2+)</td><td colspan="2">Планируется</td></tr>
<tr><td>Broker Load</td><td>Да</td><td>Да (v1.5.2+)</td><td>Да</td><td>Да</td><td colspan="3">Планируется</td></tr>
<tr><td>Routine Load</td><td>Kafka</td><td>Да</td><td>Да</td><td>Планируется</td><td>Планируется</td><td>Да (v1.5.2+) [1]</td><td>Планируется</td><td>Планируется</td></tr>
<tr><td>Spark Load</td><td></td><td>Да</td><td>Планируется</td><td>Да</td><td>Да</td><td colspan="3">Планируется</td></tr>
<tr><td>Коннекторы</td><td>Flink, Spark</td><td>Да</td><td>Да</td><td>Планируется</td><td>Планируется</td><td colspan="3">Планируется</td></tr>
</table>

Kafka Connector [2]	Kafka	Да (v1.5.2+)	Планируется	Планируется	Да (v1.5.2+)	Планируется
PIPE [4]	Соответствует INSERT from FILES					

> **ПРИМЕЧАНИЕ**
>
> [1], [2]: Требуется Schema Registry.
>
> [3]: JSON поддерживает различные форматы CDC. Подробнее о форматах JSON CDC, поддерживаемых Selena, см. в разделе Форматы JSON CDC.
>
> [4]: В настоящее время для загрузки с помощью PIPE поддерживается только INSERT from FILES.
>
> [5]: Вам нужно примонтировать устройство NAS как NFS в одну и ту же директорию каждого узла BE или CN для доступа к файлам в NFS через протокол `file://`.

Форматы JSON CDC

	Stream Load	Routine Load	Broker Load	INSERT from FILES	Kafka Connector [1]
Debezium	Планируется	Планируется	Планируется	Планируется	Да (v1.5.2+)
Canal	Планируется				
Maxwell					

> **ПРИМЕЧАНИЕ**
>
> [1]: При загрузке данных в формате Debezium CDC в таблицы Primary Key в Selena необходимо настроить параметр `transforms`.

Форматы файлов для выгрузки

	Цель		Формат файла			
	Формат таблицы	Удалённое хранилище	CSV	JSON	Parquet	ORC
INSERT INTO FILES	Н/Д	HDFS, S3, OSS, Azure, GCS, NFS(NAS) [3]	Да (v1.5.2+)	Планируется	Да (v1.5.2+)	Да (v1.5.2+)
INSERT INTO Catalog	Hive	HDFS, S3, OSS, Azure, GCS	Да (v1.5.2+)	Планируется	Да (v1.5.2+)	Да (v1.5.2+)

	Iceberg	HDFS, S3, OSS, Azure, GCS	Планируется	Планируется	Да (v1.5.2+)	Планируется
	Hudi/Delta		Планируется			
EXPORT	Н/Д	HDFS, S3, OSS, Azure, GCS	Да [1]	Планируется	Планируется	Планируется
PIPE	Планируется [2]					

ПРИМЕЧАНИЕ

[1]: Поддерживается настройка процесса Broker.

[2]: В настоящее время выгрузка данных с помощью PIPE не поддерживается.

[3]: Вам нужно примонтировать устройство NAS как NFS в одну и ту же директорию каждого узла BE или CN для доступа к файлам в NFS через протокол `file://`.

Параметры, связанные с форматом файла

Параметры формата файла для загрузки

Формат файла	Параметр	Метод загрузки				
		Stream Load	INSERT from FILES	Broker Load	Routine Load	Spark Load
CSV	column_separator	Да	Да (v1.5.2+)	Да [1]		
	row_delimiter	Да		Да [2] (v1.5.2+)	Да [3] (v1.5.2+)	Планируется
	enclose	Да (v1.5.2+)		Да (v1.5.2+)	Да (v1.5.2+)	Планируется
	escape					
	skip_header				Планируется	
	trim_space				Да (v1.5.2+)	
JSON	jsonpaths	Да	Планируется	Да (v1.5.2+)	Да	Планируется
	strip_outer_array					
	json_root					
	ignore_json_size				Планируется	

> **ПРИМЕЧАНИЕ**
>
> [1]: Соответствующий параметр — `COLUMNS TERMINATED BY`.
>
> [2]: Соответствующий параметр — `ROWS TERMINATED BY`.
>
> [3]: Соответствующий параметр — `ROWS TERMINATED BY`.

Параметры формата файла для выгрузки

Формат файла	Параметр	Метод выгрузки	
		INSERT INTO FILES	EXPORT
CSV	column_separator	Да (v1.5.2+)	Да
	line_delimiter [1]		

> **ПРИМЕЧАНИЕ**
>
> [1]: Соответствующий параметр при загрузке данных — `row_delimiter`.

Форматы сжатия

Форматы сжатия для загрузки

Формат файла	Формат сжатия	Метод загрузки				
		Stream Load	Broker Load	INSERT from FILES	Routine Load	Spark Load
CSV	• deflate • bzip2 • gzip • lz4_frame • zstd	Да [1]	Да [2]	Планируется	Планируется	Планируется
JSON		Да (v1.5.2+) [3]	Планируется	Н/Д	Планируется	Н/Д
Parquet	• gzip • lz4 • snappy • zlib • zstd	Н/Д	Да [4]		Планируется	Да [4]
ORC						

> **ПРИМЕЧАНИЕ**
>
> [1]: В настоящее время только при загрузке файлов CSV с помощью Stream Load можно указать формат сжатия с помощью `format=gzip`, что указывает на файлы CSV, сжатые gzip. Форматы `deflate` и `bzip2` также поддерживаются.
>
> [2]: Broker Load не поддерживает указание формата сжатия файлов CSV с помощью параметра `format`. Broker Load определяет формат сжатия по расширению файла. Расширение файлов, сжатых gzip, — `.gz`, а файлов, сжатых zstd, — `.zst`. Кроме того, другие параметры, связанные с `format`, такие как `trim_space` и `enclose`, также не поддерживаются.
>
> [3]: Поддерживается указание формата сжатия с помощью `compression = gzip`.
>
> [4]: Поддерживается библиотекой Arrow. Настройка параметра `compression` не требуется.

Форматы сжатия для выгрузки

Формат файла	Формат сжатия	Метод выгрузки				
		INSERT INTO FILES	INSERT INTO Catalog			EXPORT
			Hive	Iceberg	Hudi/Delta	
CSV	• deflate • bzip2 • gzip • lz4_frame • zstd	Планируется	Планируется	Планируется	Планируется	Планируется
JSON	Н/Д	Н/Д	Н/Д	Н/Д	Н/Д	Н/Д
Parquet	• gzip • lz4 • snappy • zstd	Да (v1.5.2+)	Да (v1.5.2+)	Да (v1.5.2+)	Планируется	Н/Д
ORC						

Учётные данные

Загрузка — Аутентификация

Аутентификация	Метод загрузки

	Stream Load	INSERT from FILES	Broker Load	Routine Load	External Catalog
Единичный Kerberos	Н/Д	Да (v1.5.2+)	Да [1] (версии ранее v1.5.2)	Да [2] (v1.5.2+)	Да
Kerberos Ticket Granting Ticket (TGT)	Н/Д	Планируется			Да (v1.5.2+)
Единый KDC, множественный Kerberos	Н/Д				
Базовая аутентификация доступа (пара ключей доступа, IAM Role)	Н/Д	Да (HDFS и S3-совместимое объектное хранилище)		Да [3]	Да

ПРИМЕЧАНИЕ

[1]: Для HDFS Selena поддерживает как простую аутентификацию, так и аутентификацию Kerberos.

[2]: Когда протокол безопасности установлен на `sasl_plaintext` или `sasl_ssl`, поддерживаются как аутентификация SASL, так и GSSAPI (Kerberos).

[3]: Когда протокол безопасности установлен на `sasl_plaintext` или `sasl_ssl`, поддерживаются как аутентификация SASL, так и PLAIN.

Выгрузка — Аутентификация

	INSERT INTO FILES	EXPORT
Единичный Kerberos	Планируется	Планируется

Загрузка — Другие параметры и функции

Параметр и функция	Метод загрузки						
	Stream Load	INSERT from FILES	INSERT from SELECT/VALUES	Broker Load	PIPE	Routine Load	Spark Load
partial_update	Да (v1.5.2+)	Да [1] (v1.5.2+)		Да (v1.5.2+)	Н/Д	Да (v1.5.2+)	Планируется
partial_update_mode	Да (v1.5.2+)	Планируется		Да (v1.5.2+)	Н/Д	Планируется	Планируется
COLUMNS FROM PATH	Н/Д	Да (v1.5.2+)	Н/Д	Да	Н/Д	Н/Д	Да
timezone или сессионная переменная time_zone [2]	Да [3]	Да [4]	Да [4]	Да [4]	Планируется	Да [4]	Планируется

Точность времени — микросекунды	Да	Да	Да	Да (v1.5.2+)	Планируется	Да	Да

> **ПРИМЕЧАНИЕ**
>
> [1]: Начиная с v1.5.2, Selena поддерживает частичные обновления в режиме Row для INSERT INTO путём указания списка столбцов.
>
> [2]: Установка часового пояса с помощью параметра или сессионной переменной влияет на результаты, возвращаемые функциями strftime(), alignment_timestamp() и from_unixtime().
>
> [3]: Поддерживается только параметр `timezone`.
>
> [4]: Поддерживается только сессионная переменная `time_zone`.

Выгрузка — Другие параметры и функции

Параметр и функция	INSERT INTO FILES	EXPORT
target_max_file_size	Да (v1.5.2+)	Планируется
single		
Partitioned_by		
Сессионная переменная time_zone	Планируется	
Точность времени — микросекунды	Планируется	Планируется

3.1.5 Устранение неполадок при загрузке данных

Это руководство предназначено для помощи администраторам баз данных и инженерам по эксплуатации в мониторинге состояния заданий загрузки данных через SQL-интерфейсы — без использования внешних систем мониторинга. Оно также содержит рекомендации по выявлению узких мест производительности и устранению аномалий во время операций загрузки.

Терминология

Задание загрузки: Непрерывный процесс загрузки данных, такой как **Routine Load Job** или **Pipe Job**.

Задача загрузки: Однократный процесс загрузки данных, обычно соответствующий одной транзакции загрузки. Примеры включают **Broker Load**, **Stream Load**, **Spark Load** и **INSERT INTO**. Задания Routine Load и Pipe непрерывно генерируют задачи для выполнения приёма данных.

Наблюдение за заданиями загрузки

Существует два способа наблюдения за заданиями загрузки:

- Использование SQL-операторов **SHOW ROUTINE LOAD** и **SHOW PIPES**.
- Использование системных представлений **information_schema.routine_load_jobs** и **information_schema.pipes**.

Наблюдение за задачами загрузки

Задачи загрузки также можно отслеживать двумя способами:

- Использование SQL-операторов **SHOW LOAD** и **SHOW ROUTINE LOAD TASK**.
- Использование системных представлений **information_schema.loads** и **statistics.loads_history**.

SQL-операторы

Операторы **SHOW** отображают как текущие, так и недавно завершённые задачи загрузки для текущей базы данных, предоставляя быстрый обзор состояния задач. Полученная информация является подмножеством системного представления **statistics.loads_history**.

Операторы SHOW LOAD возвращают информацию о задачах Broker Load, Insert Into и Spark Load, а операторы SHOW ROUTINE LOAD TASK возвращают информацию о задачах Routine Load.

Системные представления

information_schema.loads

Системное представление **information_schema.loads** хранит информацию о недавних задачах загрузки, включая активные и недавно завершённые. Selena периодически синхронизирует данные в системную таблицу **statistics.loads_history** для постоянного хранения.

information_schema.loads предоставляет следующие поля:

Поле	Описание
ID	Глобально уникальный идентификатор.
LABEL	Метка задания загрузки.
PROFILE_ID	ID профиля, который можно проанализировать с помощью `ANALYZE PROFILE`.
DB_NAME	База данных, к которой принадлежит целевая таблица.
TABLE_NAME	Целевая таблица.
USER	Пользователь, инициировавший задание загрузки.
WAREHOUSE	Warehouse, к которому принадлежит задание загрузки.
STATE	Состояние задания загрузки. Допустимые значения: • `PENDING` / `BEGIN`: Задание загрузки создано. • `QUEUEING` / `BEFORE_LOAD`: Задание загрузки в очереди на планирование. • `LOADING`: Задание загрузки выполняется. • `PREPARING`: Транзакция находится в процессе предварительной фиксации. • `PREPARED`: Транзакция предварительно зафиксирована. • `COMMITED`: Транзакция зафиксирована. • `FINISHED`: Задание загрузки выполнено успешно. • `CANCELLED`: Задание загрузки не выполнено.
PROGRESS	Прогресс этапов ETL и LOADING задания загрузки.
TYPE	Тип задания загрузки. Для Broker Load возвращаемое значение — `BROKER`. Для INSERT возвращаемое значение — `INSERT`. Для Stream Load возвращаемое значение — `STREAM`. Для Routine Load возвращаемое значение — `ROUTINE`.
PRIORITY	Приоритет задания загрузки. Допустимые значения: `HIGHEST`, `HIGH`, `NORMAL`, `LOW` и `LOWEST`.
SCAN_ROWS	Количество сканированных строк данных.
SCAN_BYTES	Количество сканированных байт.
FILTERED_ROWS	Количество строк данных, отфильтрованных из-за недостаточного качества данных.
UNSELECTED_ROWS	Количество строк данных, отфильтрованных из-за условий, указанных в предложении WHERE.

Поле	Описание
SINK_ROWS	Количество загруженных строк данных.
RUNTIME_DETAILS	Метаданные времени выполнения загрузки. Подробнее см. в разделе RUNTIME_DETAILS.
CREATE_TIME	Время создания задания загрузки. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.
LOAD_START_TIME	Время начала этапа LOADING задания загрузки. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.
LOAD_COMMIT_TIME	Время фиксации транзакции загрузки. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.
LOAD_FINISH_TIME	Время окончания этапа LOADING задания загрузки. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.
PROPERTIES	Статические свойства задания загрузки. Подробнее см. в разделе PROPERTIES.
ERROR_MSG	Сообщение об ошибке задания загрузки. Если задание загрузки не столкнулось с ошибками, возвращается `NULL`.
TRACKING_SQL	SQL-оператор для запроса журнала отслеживания задания загрузки. SQL-оператор возвращается только при наличии некачественных строк данных в задании загрузки. Если задание загрузки не содержит некачественных строк данных, возвращается `NULL`.
REJECTED_RECORD_PATH	Путь для доступа ко всем некачественным строкам данных, отфильтрованным в задании загрузки. Количество записанных некачественных строк данных определяется параметром `log_rejected_record_num`, настроенным в задании загрузки. Вы можете использовать команду `wget` для доступа к этому пути. Если задание загрузки не содержит некачественных строк данных, возвращается `NULL`.

RUNTIME_DETAILS

- Универсальные метрики:

Метрика	Описание
load_id	Глобально уникальный ID плана выполнения загрузки.
txn_id	ID транзакции загрузки.

- Специфичные метрики для Broker Load, INSERT INTO и Spark Load:

Метрика	Описание
etl_info	Детали ETL. Это поле действительно только для заданий Spark Load. Для других типов заданий загрузки значение будет пустым.

Метрика	Описание
etl_start_time	Время начала этапа ETL задания загрузки. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.
etl_start_time	Время окончания этапа ETL задания загрузки. Формат: `yyyy-MM-dd HH:mm:ss`. Пример: `2023-07-24 14:58:58`.
unfinished_backends	Список BE с незавершённым выполнением.
backends	Список BE, участвующих в выполнении.
file_num	Количество прочитанных файлов.
file_size	Общий размер прочитанных файлов.
task_num	Количество подзадач.

- Специфичные метрики для Routine Load:

Метрика	Описание
schedule_interval	Интервал планирования Routine Load.
wait_slot_time	Время ожидания слотов выполнения задачей Routine Load.
check_offset_time	Время проверки информации об offset при планировании задачи Routine Load.
consume_time	Время, потраченное задачей Routine Load на чтение данных из upstream.
plan_time	Время генерации плана выполнения.
commit_publish_time	Время выполнения COMMIT RPC.

- Специфичные метрики для Stream Load:

Метрика	Описание
timeout	Тайм-аут для задач загрузки.
begin_txn_ms	Время начала транзакции.
plan_time_ms	Время генерации плана выполнения.
receive_data_time_ms	Время получения данных.
commit_publish_time_ms	Время выполнения COMMIT RPC.
client_ip	IP-адрес клиента.

PROPERTIES

- Специфичные свойства для Broker Load, INSERT INTO и Spark Load:

Свойство	Описание
timeout	Тайм-аут для задач загрузки.
max_filter_ratio	Максимальная доля строк данных, отфильтрованных из-за недостаточного качества данных.

- Специфичные свойства для Routine Load:

Свойство	Описание
job_name	Имя задания Routine Load.
task_num	Количество подзадач, фактически выполняемых параллельно.
timeout	Тайм-аут для задач загрузки.

statistics.loads_history

Системное представление **statistics.loads_history** хранит записи загрузки за последние три месяца по умолчанию. DBA могут настроить период хранения, изменив `partition_ttl` представления. **statistics.loads_history** имеет такую же схему, как **information_schema.loads**.

Выявление проблем производительности загрузки с помощью Load Profiles

Load Profile записывает детали выполнения всех рабочих узлов, участвующих в загрузке данных. Он помогает быстро выявить узкие места производительности в cluster Selena.

Включение Load Profiles

Selena предоставляет несколько методов включения Load Profiles в зависимости от типа загрузки:

Для Broker Load и INSERT INTO

Включите Load Profiles для Broker Load и INSERT INTO на уровне сессии:

```
SET enable_profile = true;
```

По умолчанию профили автоматически включаются для длительных заданий (более 300 секунд). Вы можете настроить этот порог:

```
SET big_query_profile_threshold = 60s;
```

> **ПРИМЕЧАНИЕ**
>
> Когда `big_query_profile_threshold` установлен в значение по умолчанию `0`, поведение по умолчанию — отключение профилирования для запросов. Однако для задач загрузки профили автоматически записываются для задач с временем выполнения более 300 секунд.

Selena также поддерживает **Runtime Profiles**, которые периодически (каждые 30 секунд) сообщают метрики выполнения длительных заданий загрузки. Вы можете настроить интервал отчётов:

```
SET runtime_profile_report_interval = 60;
```

> **ПРИМЕЧАНИЕ**
>
> `runtime_profile_report_interval` указывает только минимальный интервал отчётов для задач загрузки. Фактический интервал отчётов динамически корректируется и может превышать это значение.

Для Stream Load и Routine Load

Включите Load Profiles для Stream Load и Routine Load на уровне таблицы:

```
ALTER TABLE <table_name> SET ("enable_load_profile" = "true");
```

Stream Load обычно имеет высокий QPS, поэтому Selena позволяет производить выборку для сбора Load Profile, чтобы избежать снижения производительности из-за обширного профилирования. Вы можете настроить интервал сбора, установив параметр FE `load_profile_collect_interval_second`. Эта настройка применяется только к Load Profiles, включённым через свойства таблицы. Значение по умолчанию — `0`.

```
ADMIN SET FRONTEND CONFIG ("load_profile_collect_interval_second"="30");
```

Selena также позволяет собирать профили только для заданий загрузки, превышающих определённый временной порог. Вы можете настроить этот порог с помощью параметра FE `stream_load_profile_collect_threshold_second`. Значение по умолчанию — `0`.

```
ADMIN SET FRONTEND CONFIG
("stream_load_profile_collect_threshold_second"="10");
```

Анализ Load Profiles

Структура Load Profiles идентична структуре Query Profiles. Подробные инструкции см. в разделе Рецепты оптимизации запросов.

Вы можете анализировать Load Profiles, выполнив ANALYZE PROFILE. Подробные инструкции см. в разделе Анализ текстовых профилей.

Профили предоставляют подробные метрики операторов. Ключевые компоненты включают оператор `OlapTableSink` и оператор `LoadChannel`.

Оператор OlapTableSink

Метрика	Описание
IndexNum	Количество синхронных материализованных представлений целевой таблицы.
ReplicatedStorage	Включена ли репликация с одним лидером.
TxnID	ID транзакции загрузки.
RowsRead	Количество строк данных, прочитанных от upstream-оператора.
RowsFiltered	Количество строк данных, отфильтрованных из-за недостаточного качества данных.
RowsReturned	Количество загруженных строк данных.
RpcClientSideTime	Общее время записи данных RPC со стороны клиента.
RpcServerSideTime	Общее время записи данных RPC со стороны сервера.
PrepareDataTime	Время преобразования формата данных и проверки качества данных.
SendDataTime	Локальное время отправки данных, включая сериализацию, сжатие и запись в очередь отправки.

ПОДСКАЗКА

- Значительное расхождение между максимальным и минимальным значениями `PushChunkNum` в `OLAP_TABLE_SINK` указывает на перекос данных в upstream-операторе, что может вызвать узкие места производительности записи.
- `RpcClientSideTime` равно сумме `RpcServerSideTime`, времени сетевой передачи и времени обработки RPC-фреймворка. Если разница между `RpcClientSideTime` и `RpcServerSideTime` значительна, рассмотрите возможность включения сжатия данных для сокращения времени передачи.
- Если `RpcServerSideTime` составляет значительную часть потраченного времени, дальнейший анализ можно провести с использованием профиля `LoadChannel`.

Оператор LoadChannel

Метрика	Описание
Address	IP-адрес или FQDN узла BE.
LoadMemoryLimit	Лимит памяти для загрузки.
PeakMemoryUsage	Пиковое использование памяти для загрузки.
OpenCount	Количество открытий канала, отражающее общую конкурентность sink.
OpenTime	Общее время открытия канала.
AddChunkCount	Количество загружаемых chunk, то есть количество вызовов `TabletsChannel::add_chunk`.
AddRowNum	Количество загруженных строк данных.
AddChunkTime	Общее время загрузки chunk, то есть общее время выполнения `TabletsChannel::add_chunk`.
WaitFlushTime	Общее время ожидания сброса MemTable функцией `TabletsChannel::add_chunk`.
WaitWriterTime	Общее время ожидания выполнения Async Delta Writer функцией `TabletsChannel::add_chunk`.
WaitReplicaTime	Общее время ожидания синхронизации от replica функцией `TabletsChannel::add_chunk`.
PrimaryTabletsNum	Количество первичных tablet.
SecondaryTabletsNum	Количество вторичных tablet.

ПОДСКАЗКА

Если `WaitFlushTime` занимает длительное время, это может указывать на недостаточное количество ресурсов для потока flush. Рассмотрите возможность настройки параметра BE `flush_thread_num_per_store`.

Лучшие практики

Диагностика узкого места производительности Broker Load

1. Загрузите данные с помощью Broker Load:

```
LOAD LABEL click_bench.hits_1713874468
(
    DATA INFILE ("s3://test-data/benchmark_data/query_data/click_bench/hi
    INTO TABLE hits COLUMNS TERMINATED BY "\t"
(WatchID,JavaEnable,Title,GoodEvent,EventTime,EventDate,CounterID,ClientI
)
WITH BROKER
(
            "aws.s3.access_key" = "<iam_user_access_key>",
            "aws.s3.secret_key" = "<iam_user_secret_key>",
            "aws.s3.region" = "<aws_s3_region>"
)
```

2. Используйте **SHOW PROFILELIST** для получения списка runtime-профилей.

```
MySQL [click_bench]> SHOW PROFILELIST;
+--------------------------------------+---------------------+---------
-+---------+-------------------------------------------------------------
-----------------------------------------------------------------------
+
| QueryId                              | StartTime           | Time
| State   | Statement
|
+--------------------------------------+---------------------+---------
-+---------+-------------------------------------------------------------
-----------------------------------------------------------------------
+
| 3df61627-f82b-4776-b16a-6810279a79a3 | 2024-04-23 20:28:26 | 11s850ms
| Running | LOAD LABEL click_bench.hits_1713875306 (DATA INFILE
("s3://test-data/benchmark_data/query_data/click_bench/hits.tbl*" ... |
+--------------------------------------+---------------------+---------
-+---------+-------------------------------------------------------------
-----------------------------------------------------------------------
+
1 row in set (0.00 sec)
```

3. Используйте **ANALYZE PROFILE** для просмотра Runtime Profile.

```
MySQL [click_bench]> ANALYZE PROFILE FROM '3df61627-f82b-4776-b16a-
6810279a79a3';
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
----------------+
| Explain String
|
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
----------------+
| Summary
|
|     Attention: The transaction of the statement will be aborted, and
no data will be actually inserted!!!                                  |
|     Attention: Profile is not identical!!!
|
|     QueryId: 3df61627-f82b-4776-b16a-6810279a79a3
|
|     Version: default_profile-70fe819
|
|     State: Running
|
|     Legend: ⏳ for blocked; ▫ for running; ▫ for finished
|
|     TotalTime: 31s832ms
|
|         ExecutionTime: 30s1ms [Scan: 28s885ms (96.28%), Network: 0ns
(0.00%), ResultDeliverTime: 7s613ms (25.38%), ScheduleTime: 145.701ms
(0.49%)] |
|         FrontendProfileMergeTime: 3.838ms
|
|     QueryPeakMemoryUsage: 141.367 MB, QueryAllocatedMemoryUsage:
82.422 GB
|
|     Top Most Time-consuming Nodes:
|
|         1. FILE_SCAN (id=0)  ▫ : 28s902ms (85.43%)
|
|         2. OLAP_TABLE_SINK ▫ : 4s930ms (14.57%)
|
|     Top Most Memory-consuming Nodes:
|
|     Progress (finished operator/all operator): 0.00%
|
|     NonDefaultVariables:
```

```
|
|         big_query_profile_threshold: 0s -> 60s
|
|         enable_adaptive_sink_dop: false -> true
|
|         enable_profile: false -> true
|
|         sql_mode_v2: 32 -> 34
|
|         use_compute_nodes: -1 -> 0
|
| Fragment 0
|
| |   BackendNum: 3
|
| |   InstancePeakMemoryUsage: 128.541 MB,
InstanceAllocatedMemoryUsage: 82.422 GB
|
| |   PrepareTime: 2.304ms
|
| └──OLAP_TABLE_SINK
|
|     |   TotalTime: 4s930ms (14.57%) [CPUTime: 4s930ms]
|
|     |   OutputRows: 14.823M (14823424)
|
|     |   PartitionType: RANDOM
|
|     |   Table: hits
|
|     └──FILE_SCAN (id=0)  ▫
|
|           Estimates: [row: ?, cpu: ?, memory: ?, network: ?, cost: ?]
|
|           TotalTime: 28s902ms (85.43%) [CPUTime: 17.038ms, ScanTime:
28s885ms]
|
|           OutputRows: 14.823M (14823424)
|
|           Progress (processed rows/total rows): ?
|
|           Detail Timers: [ScanTime = IOTaskExecTime + IOTaskWaitTime]
|
|               IOTaskExecTime: 25s612ms [min=19s376ms, max=28s804ms]
|
|               IOTaskWaitTime: 63.192ms [min=20.946ms, max=91.668ms]
```

```
|
|
|
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
----------------+
40 rows in set (0.04 sec)
```

Профиль показывает, что раздел `FILE_SCAN` занял почти 29 секунд, что составляет примерно 90% от общих 32 секунд. Это указывает на то, что чтение данных из объектного хранилища в настоящее время является узким местом в процессе загрузки.

Диагностика производительности Stream Load

1. Включите Load Profile для целевой таблицы.

```
mysql> ALTER TABLE duplicate_200_column_sCH
SET('enable_load_profile'='true');
Query OK, 0 rows affected (0.00 sec)
```

2. Используйте **SHOW PROFILELIST** для получения списка профилей.

```
mysql> SHOW PROFILELIST;
+--------------------------------------+---------------------+---------
-+----------+-----------+
| QueryId                              | StartTime           | Time
| State    | Statement |
+--------------------------------------+---------------------+---------
-+----------+-----------+
| 90481df8-afaf-c0fd-8e91-a7889c1746b6 | 2024-09-19 10:43:38 | 9s571ms
| Finished |           |
| 9c41a13f-4d7b-2c18-4eaf-cdeea3facba5 | 2024-09-19 10:43:37 | 10s664ms
| Finished |           |
| 5641cf37-0af4-f116-46c6-ca7cce149886 | 2024-09-19 10:43:20 | 13s88ms
| Finished |           |
| 4446c8b3-4dc5-9faa-dccb-e1a71ab3519e | 2024-09-19 10:43:20 | 13s64ms
| Finished |           |
| 48469b66-3866-1cd9-9f3b-17d786bb4fa7 | 2024-09-19 10:43:20 | 13s85ms
| Finished |           |
| bc441907-e779-bc5a-be8e-992757e4d992 | 2024-09-19 10:43:19 | 845ms
| Finished |           |
+--------------------------------------+---------------------+---------
-+----------+-----------+
```

3. Используйте **ANALYZE PROFILE** для просмотра профиля.

```
mysql> ANALYZE PROFILE FROM '90481df8-afaf-c0fd-8e91-a7889c1746b6';
+-----------------------------------------------------------+
| Explain String                                            |
+-----------------------------------------------------------+
| Load:                                                     |
|   Summary:                                                |
|      - Query ID: 90481df8-afaf-c0fd-8e91-a7889c1746b6     |
|      - Start Time: 2024-09-19 10:43:38                    |
|      - End Time: 2024-09-19 10:43:48                      |
|      - Query Type: Load                                   |
|      - Load Type: STREAM_LOAD                             |
|      - Query State: Finished                              |
|      - Selena Version: main-d49cb08                    |
|      - Sql Statement                                      |
|      - Default Db: ingestion_db                           |
|      - NumLoadBytesTotal: 799008                          |
|      - NumRowsAbnormal: 0                                 |
|      - NumRowsNormal: 280                                 |
|      - Total: 9s571ms                                     |
|      - numRowsUnselected: 0                               |
|   Execution:                                              |
|     Fragment 0:                                           |
|        - Address: 172.26.93.218:59498                     |
|        - InstanceId: 90481df8-afaf-c0fd-8e91-a7889c1746b7 |
|        - TxnID: 1367                                      |
|        - ReplicatedStorage: true                          |
|        - AutomaticPartition: false                        |
|        - InstanceAllocatedMemoryUsage: 12.478 MB          |
|        - InstanceDeallocatedMemoryUsage: 10.745 MB        |
|        - InstancePeakMemoryUsage: 9.422 MB                |
|        - MemoryLimit: -1.000 B                            |
|        - RowsProduced: 280                                |
|          - AllocAutoIncrementTime: 348ns                  |
|          - AutomaticBucketSize: 0                         |
|          - BytesRead: 0.000 B                             |
|          - CloseWaitTime: 9s504ms                         |
|          - IOTaskExecTime: 0ns                            |
|          - IOTaskWaitTime: 0ns                            |
|          - IndexNum: 1                                    |
|          - NumDiskAccess: 0                               |
|          - OpenTime: 15.639ms                             |
|          - PeakMemoryUsage: 0.000 B                       |
|          - PrepareDataTime: 583.480us                     |
|            - ConvertChunkTime: 44.670us                   |
|            - ValidateDataTime: 109.333us                  |
```

```
|          - RowsFiltered: 0                                |
|          - RowsRead: 0                                    |
|          - RowsReturned: 280                              |
|          - RowsReturnedRate: 12.049K (12049) /sec         |
|          - RpcClientSideTime: 28s396ms                    |
|          - RpcServerSideTime: 28s385ms                    |
|          - RpcServerWaitFlushTime: 0ns                    |
|          - ScanTime: 9.841ms                              |
|          - ScannerQueueCounter: 1                         |
|          - ScannerQueueTime: 3.272us                      |
|          - ScannerThreadsInvoluntaryContextSwitches: 0    |
|          - ScannerThreadsTotalWallClockTime: 0ns          |
|            - MaterializeTupleTime(*): 0ns                 |
|            - ScannerThreadsSysTime: 0ns                   |
|            - ScannerThreadsUserTime: 0ns                  |
|          - ScannerThreadsVoluntaryContextSwitches: 0      |
|          - SendDataTime: 2.452ms                          |
|            - PackChunkTime: 1.475ms                       |
|            - SendRpcTime: 1.617ms                         |
|              - CompressTime: 0ns                          |
|              - SerializeChunkTime: 880.424us              |
|            - WaitResponseTime: 0ns                        |
|          - TotalRawReadTime(*): 0ns                       |
|          - TotalReadThroughput: 0.000 B/sec               |
|       DataSource:                                         |
|          - DataSourceType: FileDataSource                 |
|          - FileScanner:                                   |
|            - CastChunkTime: 0ns                           |
|            - CreateChunkTime: 227.100us                   |
|            - FileReadCount: 3                             |
|            - FileReadTime: 253.765us                      |
|            - FillTime: 6.892ms                            |
|            - MaterializeTime: 133.637us                   |
|            - ReadTime: 0ns                                |
|          - ScannerTotalTime: 9.292ms                      |
+-----------------------------------------------------------+
76 rows in set (0.00 sec)
```

Приложение

Полезные SQL-запросы для эксплуатации

> **ПРИМЕЧАНИЕ**
>
> Этот раздел применяется только к cluster shared-nothing.

Запрос пропускной способности в минуту

```
-- общая
select date_trunc('minute', load_finish_time) as t,count(*) as
tpm,sum(SCAN_BYTES) as scan_bytes,sum(sink_rows) as sink_rows from
_statistics_.loads_history group by t order by t desc limit 10;

-- по таблице
select date_trunc('minute', load_finish_time) as t,count(*) as
tpm,sum(SCAN_BYTES) as scan_bytes,sum(sink_rows) as sink_rows from
_statistics_.loads_history where table_name = 't' group by t order by t
desc limit 10;
```

Запрос RowsetNum и SegmentNum таблицы

```
-- общий
select * from information_schema.be_tablets t,
information_schema.tables_config c where t.table_id = c.table_id order by
num_segment desc limit 5;
select * from information_schema.be_tablets t,
information_schema.tables_config c where t.table_id = c.table_id order by
num_rowset desc limit 5;

-- по таблице
select * from information_schema.be_tablets t,
information_schema.tables_config c where t.table_id = c.table_id and
table_name = 't' order by num_segment desc limit 5;
select * from information_schema.be_tablets t,
information_schema.tables_config c where t.table_id = c.table_id and
table_name = 't' order by num_rowset desc limit 5;
```

- Высокий RowsetNum (>100) указывает на слишком частые загрузки. Рассмотрите возможность уменьшения частоты или увеличения потоков Compaction.

- Высокий SegmentNum (>100) указывает на избыточное количество сегментов на загрузку. Рассмотрите возможность увеличения потоков Compaction или применения стратегии случайного распределения для таблицы.

Проверка перекоса данных

Перекос данных по узлам

```
-- общий
SELECT tbt.be_id, sum(tbt.DATA_SIZE) FROM information_schema.tables_config
tb JOIN information_schema.be_tablets tbt ON tb.TABLE_ID = tbt.TABLE_ID
group by be_id;

-- по таблице
SELECT tbt.be_id, sum(tbt.DATA_SIZE) FROM information_schema.tables_config
tb JOIN information_schema.be_tablets tbt ON tb.TABLE_ID = tbt.TABLE_ID
WHERE tb.table_name = 't' group by be_id;
```

Если обнаружен перекос на уровне узлов, рассмотрите возможность использования столбца с более высокой кардинальностью в качестве ключа распределения или применения стратегии случайного распределения для таблицы.

Перекос данных по tablet

```
select
tablet_id,t.data_size,num_row,visible_version,num_version,num_rowset,num_segm
from information_schema.partitions_meta m, information_schema.be_tablets t wh
= m.partition_id and m.partition_name = 'att' and m.table_name='att' order by
desc;
```

Общие метрики мониторинга для загрузки

BE Load

Эти метрики доступны в категории **BE Load** в Grafana. Если вы не можете найти эту категорию, убедитесь, что вы используете последний шаблон дашборда Grafana.

ThreadPool

Эти метрики помогают анализировать состояние пулов потоков — например, накапливаются ли задачи в очереди или сколько времени они проводят в ожидании. В настоящее время мониторятся четыре пула потоков:

- `async_delta_writer`
- `memtable_flush`
- `segment_replicate_sync`
- `segment_flush`

Каждый пул потоков включает следующие метрики:

Имя	Описание
rate	Скорость обработки задач.
pending	Время ожидания задач в очереди.
execute	Время выполнения задачи.
total	Максимальное количество потоков, доступных в пуле.
util	Утилизация пула за заданный период; из-за неточности выборки может превышать 100% при высокой нагрузке.
count	Мгновенное количество задач в очереди.

> **ПРИМЕЧАНИЕ**
>
> - Надёжным индикатором накопления является постоянное увеличение **pending duration**. **workers util** и **queue count** являются необходимыми, но не достаточными индикаторами.
> - Если происходит накопление, используйте **rate** и **execute duration**, чтобы определить, связано ли это с увеличением нагрузки или замедлением обработки.
> - **workers util** помогает оценить, насколько загружен пул, что может направить усилия по оптимизации.

LoadChannel::add_chunks

Эти метрики помогают анализировать поведение `LoadChannel::add_chunks` после получения запроса `BRPC tablet_writer_add_chunks`.

Имя	Описание
rate	Скорость обработки запросов `add_chunks`.
execute	Среднее время выполнения `add_chunks`.
wait_memtable	Среднее время ожидания сброса MemTable первичной replica.
wait_writer	Среднее время ожидания выполнения записи/фиксации async delta writer первичной replica.

Имя	Описание
wait_replica	Среднее время ожидания завершения сброса сегмента вторичными replica.

> **ПРИМЕЧАНИЕ**
>
> - Метрика **latency** равна сумме `wait_memtable`, `wait_writer` и `wait_replica`.
> - Высокая доля ожидания указывает на узкие места downstream, которые следует проанализировать дополнительно.

Async Delta Writer

Эти метрики помогают анализировать поведение **async delta writer**.

Имя	Описание
rate	Скорость обработки задач записи/фиксации.
pending	Время ожидания в очереди пула потоков.
execute	Среднее время обработки одной задачи.
wait_memtable	Среднее время ожидания сброса MemTable.
wait_replica	Среднее время ожидания синхронизации сегмента.

> **ПРИМЕЧАНИЕ**
>
> - Общее время на задачу (с точки зрения upstream) равно **pending** плюс **execute**.
> - **execute** дополнительно включает **wait_memtable** плюс **wait_replica**.
> - Высокое время **pending** может указывать на медленный **execute** или недостаточный размер пула потоков.
> - Если **wait** занимает большую часть **execute**, узкое место находится на downstream-этапах; в противном случае узкое место, вероятно, находится в логике самого writer.

MemTable Flush

Эти метрики анализируют производительность **MemTable flush**.

Имя	Описание
rate	Скорость сброса MemTable.
memory-size	Объём данных в памяти, сбрасываемых в секунду.
disk-size	Объём данных, записываемых на диск в секунду.
execute	Время выполнения задачи.

Имя	Описание
io	Время I/O задачи сброса.

> **ПРИМЕЧАНИЕ**
>
> - Сравнивая **rate** и **size**, вы можете определить, изменяется ли нагрузка или происходит массовый импорт — например, малый **rate**, но большой **size** указывает на массовый импорт.
> - Коэффициент сжатия можно оценить с помощью `memory-size / disk-size`.
> - Вы также можете оценить, является ли I/O узким местом, проверив долю времени **io** в **execute**.

Segment Replicate Sync

Имя	Описание
rate	Скорость синхронизации сегментов.
execute	Время синхронизации одной replica tablet.

Segment Flush

Эти метрики анализируют производительность **segment flush**.

Имя	Описание
rate	Скорость сброса сегментов.
size	Объём данных, сбрасываемых на диск в секунду.
execute	Время выполнения задачи.
io	Время I/O задачи сброса.

> **ПРИМЕЧАНИЕ**
>
> - Сравнивая **rate** и **size**, вы можете определить, изменяется ли нагрузка или происходит массовый импорт — например, малый **rate**, но большой **size** указывает на массовый импорт.
> - Вы также можете оценить, является ли I/O узким местом, проверив долю времени **io** в **execute**.

3.2 Загрузка данных из объектного хранилища

3.2.1 Загрузка данных из AWS S3

Selena предоставляет следующие варианты загрузки данных из AWS S3:

- Синхронная загрузка с использованием INSERT+`FILES()`
- Асинхронная загрузка с использованием Broker Load
- Непрерывная асинхронная загрузка с использованием Pipe

Каждый из этих вариантов имеет свои преимущества, которые подробно описаны в следующих разделах.

В большинстве случаев мы рекомендуем использовать метод INSERT+`FILES()`, который намного проще в использовании.

Однако метод INSERT+`FILES()` в настоящее время поддерживает только форматы файлов Parquet, ORC и CSV. Поэтому, если вам нужно загружать данные других форматов файлов, таких как JSON, или выполнять изменения данных, такие как DELETE, во время загрузки данных, вы можете использовать Broker Load.

Если вам нужно загрузить большое количество файлов данных со значительным общим объёмом данных (например, более 100 ГБ или даже 1 ТБ), мы рекомендуем использовать метод Pipe. Pipe может разбивать файлы по их количеству или размеру, разделяя задание загрузки на более мелкие последовательные задачи. Такой подход гарантирует, что ошибки в одном файле не повлияют на всё задание загрузки, и минимизирует необходимость повторных попыток из-за ошибок в данных.

Перед началом работы

Подготовка исходных данных

Убедитесь, что исходные данные, которые вы хотите загрузить в Selena, правильно хранятся в bucket S3. Вы также можете рассмотреть, где находятся данные и база данных, потому что затраты на передачу данных значительно ниже, когда ваш bucket и ваш cluster Selena находятся в одном регионе.

В этой теме мы предоставляем вам пример набора данных в bucket S3, `s3://selena-examples/user-behavior-10-million-rows.parquet`. Вы можете получить доступ к этому набору данных с любыми действительными учётными данными, так как объект доступен для чтения любому аутентифицированному пользователю AWS.

Проверка привилегий

Вы можете загружать данные в таблицы Selena только как пользователь, имеющий привилегию INSERT на эти таблицы Selena. Если у вас нет привилегии INSERT, следуйте инструкциям в разделе GRANT, чтобы предоставить привилегию INSERT пользователю, которого вы используете для подключения к вашему cluster Selena. Синтаксис: `GRANT INSERT ON TABLE <table_name> IN DATABASE <database_name> TO { ROLE <role_name> | USER <user_identity>}`.

Сбор данных аутентификации

Примеры в этой теме используют аутентификацию на основе IAM-пользователя. Чтобы убедиться, что у вас есть разрешение на чтение данных из AWS S3, мы рекомендуем прочитать Подготовка к аутентификации на основе IAM-пользователя и следовать инструкциям для создания IAM-пользователя с правильно настроенными политиками IAM.

Вкратце, если вы практикуете аутентификацию на основе IAM-пользователя, вам нужно собрать информацию о следующих ресурсах AWS:

- Bucket S3, в котором хранятся ваши данные.
- Ключ объекта S3 (имя объекта), если вы обращаетесь к определённому объекту в bucket. Обратите внимание, что ключ объекта может включать префикс, если ваши объекты S3 хранятся в подпапках.
- Регион AWS, к которому принадлежит bucket S3.
- Access key и secret key, используемые в качестве учётных данных доступа.

Для получения информации обо всех доступных методах аутентификации см. Аутентификация в ресурсах AWS.

Использование INSERT+FILES()

Этот метод доступен начиная с версии v1.5.2 и в настоящее время поддерживает только форматы файлов Parquet, ORC и CSV (начиная с версии v1.5.2).

Преимущества INSERT+FILES()

`FILES()` может читать файл, хранящийся в облачном хранилище, на основе указанных вами свойств, связанных с путём, выводить схему таблицы данных в файле, а затем возвращать данные из файла в виде строк данных.

С помощью `FILES()` вы можете:

- Запрашивать данные напрямую из S3 с помощью SELECT.
- Создавать и загружать таблицу с помощью CREATE TABLE AS SELECT (CTAS).
- Загружать данные в существующую таблицу с помощью INSERT.

Типичные примеры

Прямой запрос из S3 с помощью SELECT

Прямой запрос из S3 с помощью SELECT+`FILES()` позволяет получить хороший предварительный просмотр содержимого набора данных перед созданием таблицы. Например:

- Получить предварительный просмотр набора данных без сохранения данных.
- Запросить минимальные и максимальные значения и решить, какие типы данных использовать.
- Проверить наличие значений `NULL`.

Следующий пример запрашивает пример набора данных `s3://selena-examples/user-behavior-10-million-rows.parquet`:

```
SELECT * FROM FILES
(
    "path" = "s3://selena-examples/user-behavior-10-million-rows.parquet",
    "format" = "parquet",
    "aws.s3.region" = "us-east-1",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
)
LIMIT 3;
```

> **ПРИМЕЧАНИЕ**
>
> Замените `AAA` и `BBB` на свои учётные данные в приведённой выше команде. Можно использовать любой действительный `aws.s3.access_key` и `aws.s3.secret_key`, так как объект доступен для чтения любому аутентифицированному пользователю AWS.

Система возвращает следующий результат запроса:

```
+--------+---------+------------+--------------+---------------------+
| UserID | ItemID  | CategoryID | BehaviorType | Timestamp           |
+--------+---------+------------+--------------+---------------------+
|      1 | 2576651 |     149192 | pv           | 2017-11-25 01:21:25 |
|      1 | 3830808 |    4181361 | pv           | 2017-11-25 07:04:53 |
|      1 | 4365585 |    2520377 | pv           | 2017-11-25 07:49:06 |
+--------+---------+------------+--------------+---------------------+
```

> **ПРИМЕЧАНИЕ**
>
> Обратите внимание, что имена столбцов, возвращённые выше, предоставлены файлом Parquet.

Создание и загрузка таблицы с помощью CTAS

Это продолжение предыдущего примера. Предыдущий запрос обёрнут в CREATE TABLE AS SELECT (CTAS) для автоматизации создания таблицы с использованием вывода схемы. Это означает, что Selena выведет схему таблицы, создаст нужную вам таблицу, а затем загрузит данные в таблицу. Имена и типы столбцов не требуются для создания таблицы при использовании табличной функции `FILES()` с файлами Parquet, поскольку формат Parquet включает имена столбцов.

> **ПРИМЕЧАНИЕ**
>
> Синтаксис CREATE TABLE при использовании вывода схемы не позволяет устанавливать количество replica, поэтому установите его перед созданием таблицы. Пример ниже для системы с одной replica:
>
> ```
> ADMIN SET FRONTEND CONFIG ('default_replication_num' = "1");
> ```

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Используйте CTAS для создания таблицы и загрузки данных из примера набора данных `s3://selena-examples/user-behavior-10-million-rows.parquet` в таблицу:

```
CREATE TABLE user_behavior_inferred AS
SELECT * FROM FILES
(
    "path" = "s3://selena-examples/user-behavior-10-million-rows.parquet",
    "format" = "parquet",
    "aws.s3.region" = "us-east-1",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
);
```

> **ПРИМЕЧАНИЕ**
>
> Замените `AAA` и `BBB` на свои учётные данные в приведённой выше команде. Можно использовать любой действительный `aws.s3.access_key` и `aws.s3.secret_key`, так как объект доступен для чтения любому аутентифицированному пользователю AWS.

После создания таблицы вы можете просмотреть её схему с помощью DESCRIBE:

```
DESCRIBE user_behavior_inferred;
```

Система возвращает следующий результат запроса:

```
+--------------+------------------+------+-------+---------+-------+
| Field        | Type             | Null | Key   | Default | Extra |
+--------------+------------------+------+-------+---------+-------+
| UserID       | bigint           | YES  | true  | NULL    |       |
| ItemID       | bigint           | YES  | true  | NULL    |       |
| CategoryID   | bigint           | YES  | true  | NULL    |       |
| BehaviorType | varchar(1048576) | YES  | false | NULL    |       |
| Timestamp    | varchar(1048576) | YES  | false | NULL    |       |
+--------------+------------------+------+-------+---------+-------+
```

Запросите таблицу, чтобы убедиться, что данные были загружены в неё. Пример:

```
SELECT * from user_behavior_inferred LIMIT 3;
```

Возвращается следующий результат запроса, указывающий, что данные были успешно загружены:

```
+--------+---------+------------+--------------+---------------------+
| UserID | ItemID  | CategoryID | BehaviorType | Timestamp           |
+--------+---------+------------+--------------+---------------------+
| 225586 | 3694958 |    1040727 | pv           | 2017-12-01 00:58:40 |
| 225586 | 3726324 |     965809 | pv           | 2017-12-01 02:16:02 |
| 225586 | 3732495 |    1488813 | pv           | 2017-12-01 00:59:46 |
+--------+---------+------------+--------------+---------------------+
```

Загрузка в существующую таблицу с помощью INSERT

Вы можете захотеть настроить таблицу, в которую вставляете данные, например:

- тип данных столбца, настройку nullable или значения по умолчанию
- типы ключей и столбцы
- разбиение данных на partition и bucket

> **ПРИМЕЧАНИЕ**
>
> Создание наиболее эффективной структуры таблицы требует знания того, как данные будут использоваться и содержимого столбцов. Эта тема не охватывает проектирование таблиц. Для получения информации о проектировании таблиц см. Типы таблиц.

В этом примере мы создаём таблицу на основе знаний о том, как таблица будет запрашиваться, и данных в файле Parquet. Знание данных в файле Parquet можно получить, запросив файл напрямую в S3.

- Поскольку запрос набора данных в S3 указывает, что столбец `Timestamp` содержит данные, соответствующие типу данных VARCHAR, и Selena может преобразовать VARCHAR в DATETIME, тип данных изменён на DATETIME в следующем DDL.
- Запрашивая данные в S3, вы можете обнаружить, что в наборе данных нет значений `NULL`, поэтому DDL также может установить все столбцы как non-nullable.
- На основе знаний об ожидаемых типах запросов sort key и столбец для bucket установлены на столбец `UserID`. Ваш случай использования может отличаться для этих данных, поэтому вы можете решить использовать `ItemID` в дополнение к `UserID` или вместо него для sort key.

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Создайте таблицу вручную:

```
CREATE TABLE user_behavior_declared
(
    UserID int(11),
    ItemID int(11),
    CategoryID int(11),
    BehaviorType varchar(65533),
    Timestamp datetime
)
ENGINE = OLAP
DUPLICATE KEY(UserID)
DISTRIBUTED BY HASH(UserID);
```

Отобразите схему, чтобы вы могли сравнить её с выведенной схемой, созданной табличной функцией `FILES()`:

```
DESCRIBE user_behavior_declared;
```

```
+--------------+----------------+------+-------+---------+-------+
| Field        | Type           | Null | Key   | Default | Extra |
+--------------+----------------+------+-------+---------+-------+
| UserID       | int            | YES  | true  | NULL    |       |
| ItemID       | int            | YES  | false | NULL    |       |
| CategoryID   | int            | YES  | false | NULL    |       |
| BehaviorType | varchar(65533) | YES  | false | NULL    |       |
| Timestamp    | datetime       | YES  | false | NULL    |       |
+--------------+----------------+------+-------+---------+-------+
```

ПОДСКАЗКА

Сравните схему, которую вы только что создали, со схемой, выведенной ранее с помощью табличной функции `FILES()`. Посмотрите на:

- типы данных
- nullable
- ключевые поля

Для лучшего контроля схемы целевой таблицы и лучшей производительности запросов мы рекомендуем указывать схему таблицы вручную в производственных средах.

После создания таблицы вы можете загрузить её с помощью INSERT INTO SELECT FROM FILES():

```
INSERT INTO user_behavior_declared
SELECT * FROM FILES
(
    "path" = "s3://selena-examples/user-behavior-10-million-rows.parquet",
    "format" = "parquet",
    "aws.s3.region" = "us-east-1",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
);
```

ПРИМЕЧАНИЕ

Замените `AAA` и `BBB` на свои учётные данные в приведённой выше команде. Можно использовать любой действительный `aws.s3.access_key` и `aws.s3.secret_key`, так как объект доступен для чтения любому аутентифицированному пользователю AWS.

После завершения загрузки вы можете запросить таблицу, чтобы убедиться, что данные были загружены в неё. Пример:

```
SELECT * from user_behavior_declared LIMIT 3;
```

Возвращается следующий результат запроса, указывающий, что данные были успешно загружены:

```
+--------+---------+------------+--------------+---------------------+
| UserID | ItemID  | CategoryID | BehaviorType | Timestamp           |
+--------+---------+------------+--------------+---------------------+
| 393529 | 3715112 |     883960 | pv           | 2017-12-02 02:45:44 |
| 393529 | 2650583 |     883960 | pv           | 2017-12-02 02:45:59 |
| 393529 | 3715112 |     883960 | pv           | 2017-12-02 03:00:56 |
+--------+---------+------------+--------------+---------------------+
```

Проверка прогресса загрузки

Вы можете запросить прогресс заданий INSERT из представления `loads` в Information Schema Selena. Эта функция поддерживается начиная с версии v1.5.2. Пример:

```
SELECT * FROM information_schema.loads ORDER BY JOB_ID DESC;
```

Для получения информации о полях, предоставляемых в представлении `loads`, см. `loads`.

Если вы отправили несколько заданий загрузки, вы можете фильтровать по `LABEL`, связанному с заданием. Пример:

```
SELECT * FROM information_schema.loads WHERE LABEL = 'insert_e3b882f5-7eb3-11
*************************** 1. row ***************************
              JOB_ID: 10243
               LABEL: insert_e3b882f5-7eb3-11ee-ae77-00163e267b60
       DATABASE_NAME: mydatabase
               STATE: FINISHED
            PROGRESS: ETL:100%; LOAD:100%
                TYPE: INSERT
            PRIORITY: NORMAL
           SCAN_ROWS: 10000000
       FILTERED_ROWS: 0
     UNSELECTED_ROWS: 0
           SINK_ROWS: 10000000
            ETL_INFO:
           TASK_INFO: resource:N/A; timeout(s):300; max_filter_ratio:0.0
         CREATE_TIME: 2023-11-09 11:56:01
      ETL_START_TIME: 2023-11-09 11:56:01
     ETL_FINISH_TIME: 2023-11-09 11:56:01
     LOAD_START_TIME: 2023-11-09 11:56:01
    LOAD_FINISH_TIME: 2023-11-09 11:56:44
         JOB_DETAILS: {"All backends":{"e3b882f5-7eb3-11ee-ae77-00163e267b60"
[10142]},"FileNumber":0,"FileSize":0,"InternalTableLoadBytes":311710786,"Inte
backends":{"e3b882f5-7eb3-11ee-ae77-00163e267b60":[]}}
           ERROR_MSG: NULL
        TRACKING_URL: NULL
        TRACKING_SQL: NULL
REJECTED_RECORD_PATH: NULL
```

> **ПРИМЕЧАНИЕ**
>
> INSERT — это синхронная команда. Если задание INSERT всё ещё выполняется, вам нужно открыть другую сессию, чтобы проверить его статус выполнения.

Использование Broker Load

Асинхронный процесс Broker Load обрабатывает подключение к S3, извлечение данных и сохранение данных в Selena.

Этот метод поддерживает следующие форматы файлов:

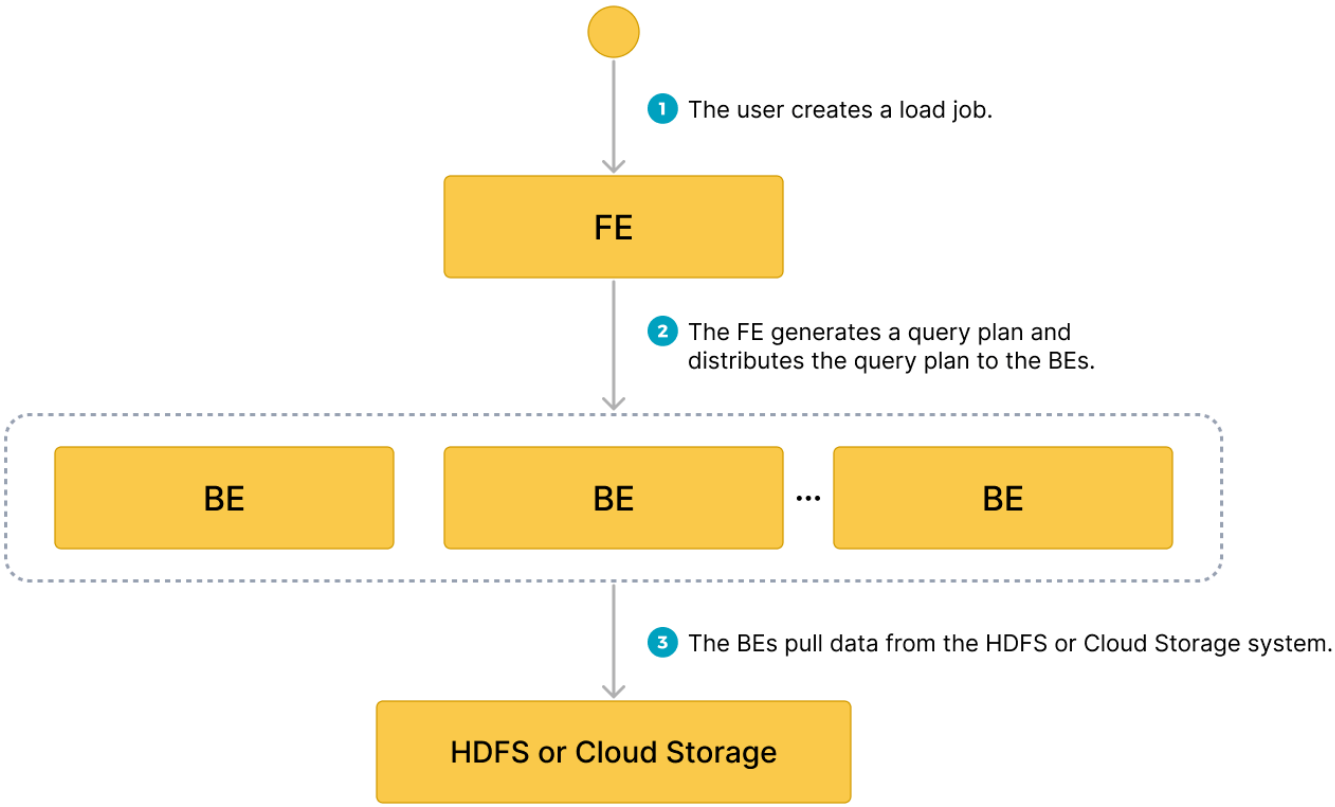
- Parquet
- ORC

- CSV
- JSON (поддерживается начиная с версии v1.5.2)

Преимущества Broker Load

- Broker Load выполняется в фоновом режиме, и клиентам не нужно оставаться подключёнными для продолжения задания.
- Broker Load предпочтителен для длительных заданий, тайм-аут по умолчанию составляет 4 часа.
- В дополнение к форматам файлов Parquet и ORC, Broker Load поддерживает формат файлов CSV и формат файлов JSON (формат файлов JSON поддерживается начиная с версии v1.5.2).

Поток данных



1. Пользователь создаёт задание загрузки.
2. Frontend (FE) создаёт план запроса и распределяет план на backend-узлы (BE) или compute-узлы (CN).
3. BE или CN извлекают данные из источника и загружают данные в Selena.

Типичный пример

Создайте таблицу, запустите процесс загрузки, который извлекает пример набора данных `s3://selena-examples/user-behavior-10-million-rows.parquet` из S3, и проверьте прогресс и успешность загрузки данных.

Создание базы данных и таблицы

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Создайте таблицу вручную (мы рекомендуем, чтобы таблица имела ту же схему, что и файл Parquet, который вы хотите загрузить из AWS S3):

```
CREATE TABLE user_behavior
(
    UserID int(11),
    ItemID int(11),
    CategoryID int(11),
    BehaviorType varchar(65533),
    Timestamp datetime
)
ENGINE = OLAP
DUPLICATE KEY(UserID)
DISTRIBUTED BY HASH(UserID);
```

Запуск Broker Load

Выполните следующую команду, чтобы запустить задание Broker Load, которое загружает данные из примера набора данных `s3://selena-examples/user-behavior-10-million-rows.parquet` в таблицу `user_behavior`:

```
LOAD LABEL user_behavior
(
    DATA INFILE("s3://selena-examples/user-behavior-10-million-
rows.parquet")
    INTO TABLE user_behavior
    FORMAT AS "parquet"
 )
 WITH BROKER
 (
    "aws.s3.enable_ssl" = "true",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.region" = "us-east-1",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
 )
PROPERTIES
(
    "timeout" = "72000"
);
```

> **ПРИМЕЧАНИЕ**
>
> Замените `AAA` и `BBB` на свои учётные данные в приведённой выше команде. Можно использовать любой действительный `aws.s3.access_key` и `aws.s3.secret_key`, так как объект доступен для чтения любому аутентифицированному пользователю AWS.

Это задание имеет четыре основных раздела:

- `LABEL`: Строка, используемая при запросе состояния задания загрузки.
- `LOAD` declaration: URI источника, формат исходных данных и имя целевой таблицы.
- `BROKER`: Данные подключения к источнику.
- `PROPERTIES`: Значение тайм-аута и любые другие свойства, применяемые к заданию загрузки.

Подробный синтаксис и описание параметров см. в BROKER LOAD.

Проверка прогресса загрузки

Вы можете запросить прогресс задания Broker Load из представления `loads` в Information Schema Selena. Эта функция поддерживается начиная с версии v1.5.2.

```
SELECT * FROM information_schema.loads WHERE LABEL = 'user_behavior';
```

Для получения информации о полях, предоставляемых в представлении `loads`, см. `loads`.

Эта запись показывает состояние `LOADING` и прогресс 39%. Если вы видите что-то подобное, выполните команду ещё раз, пока не увидите состояние `FINISHED`.

```
               JOB_ID: 10466
                LABEL: user_behavior
        DATABASE_NAME: mydatabase
                STATE: LOADING
             PROGRESS: ETL:100%; LOAD:39%
                 TYPE: BROKER
             PRIORITY: NORMAL
            SCAN_ROWS: 4620288
        FILTERED_ROWS: 0
      UNSELECTED_ROWS: 0
            SINK_ROWS: 4620288
             ETL_INFO:
            TASK_INFO: resource:N/A; timeout(s):72000; max_filter_ratio:0.0
          CREATE_TIME: 2024-02-28 22:11:36
       ETL_START_TIME: 2024-02-28 22:11:41
      ETL_FINISH_TIME: 2024-02-28 22:11:41
      LOAD_START_TIME: 2024-02-28 22:11:41
     LOAD_FINISH_TIME: NULL
          JOB_DETAILS: {"All backends":{"2fb97223-b14c-404b-9be1-83aa9b3a7715"
[10004]},"FileNumber":1,"FileSize":136901706,"InternalTableLoadBytes":1440327
backends":{"2fb97223-b14c-404b-9be1-83aa9b3a7715":[10004]}}
            ERROR_MSG: NULL
         TRACKING_URL: NULL
         TRACKING_SQL: NULL
REJECTED_RECORD_PATH: NULL
```

После того как вы убедитесь, что задание загрузки завершено, вы можете проверить подмножество целевой таблицы, чтобы увидеть, были ли данные успешно загружены. Пример:

```
SELECT * from user_behavior LIMIT 3;
```

Возвращается следующий результат запроса, указывающий, что данные были успешно загружены:

```
+--------+---------+------------+--------------+---------------------+
| UserID | ItemID  | CategoryID | BehaviorType | Timestamp           |
+--------+---------+------------+--------------+---------------------+
|     34 |  856384 |    1029459 | pv           | 2017-11-27 14:43:27 |
|     34 | 5079705 |    1029459 | pv           | 2017-11-27 14:44:13 |
|     34 | 4451615 |    1029459 | pv           | 2017-11-27 14:45:52 |
+--------+---------+------------+--------------+---------------------+
```

Использование Pipe

Начиная с версии v1.5.2, Selena предоставляет метод загрузки Pipe, который в настоящее время поддерживает только форматы файлов Parquet и ORC.

Преимущества Pipe

Pipe идеально подходит для непрерывной загрузки данных и крупномасштабной загрузки данных:

- **Крупномасштабная загрузка данных микро-пакетами помогает снизить затраты на повторные попытки, вызванные ошибками данных.**

  С помощью Pipe Selena обеспечивает эффективную загрузку большого количества файлов данных со значительным общим объёмом данных. Pipe автоматически разбивает файлы на основе их количества или размера, разделяя задание загрузки на более мелкие последовательные задачи. Такой подход гарантирует, что ошибки в одном файле не повлияют на всё задание загрузки. Статус загрузки каждого файла записывается Pipe, что позволяет легко идентифицировать и исправлять файлы, содержащие ошибки. Минимизируя необходимость повторных попыток из-за ошибок данных, этот подход помогает снизить затраты.

- **Непрерывная загрузка данных помогает сократить трудозатраты.**

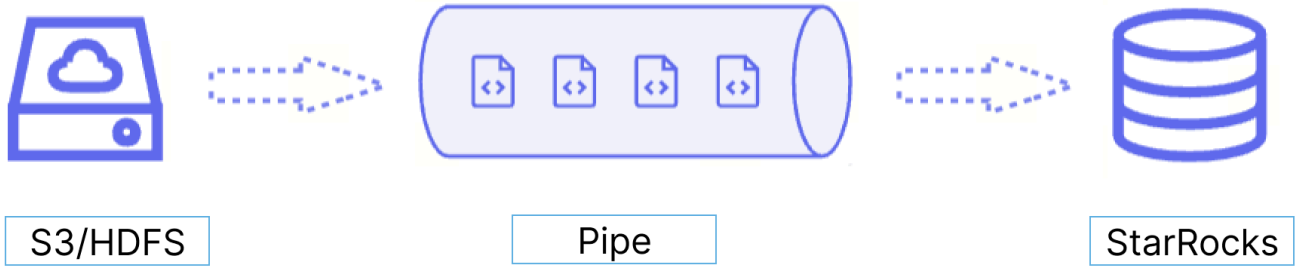
  Pipe помогает записывать новые или обновлённые файлы данных в определённое место и непрерывно загружать новые данные из этих файлов в Selena. После создания задания Pipe с указанием `"AUTO_INGEST" = "TRUE"` оно будет постоянно отслеживать изменения файлов данных, хранящихся по указанному пути, и

автоматически загружать новые или обновлённые данные из файлов данных в целевую таблицу Selena.

Кроме того, Pipe выполняет проверку уникальности файлов, чтобы предотвратить повторную загрузку данных. В процессе загрузки Pipe проверяет уникальность каждого файла данных на основе имени файла и дайджеста. Если файл с определённым именем и дайджестом уже был обработан заданием Pipe, это задание Pipe пропустит все последующие файлы с тем же именем и дайджестом. Обратите внимание, что объектное хранилище, такое как AWS S3, использует ETag используется как дайджест файла.

Статус загрузки каждого файла данных записывается и сохраняется в представлении `information_schema.pipe_files`. После удаления задания Pipe, связанного с этим представлением, записи о файлах, загруженных в этом задании, также будут удалены.

Поток данных



Pipe идеально подходит для непрерывной загрузки данных и крупномасштабной загрузки данных:

- **Крупномасштабная загрузка данных в микропакетах помогает снизить затраты на повторные попытки, вызванные ошибками данных.**

  С помощью Pipe Selena обеспечивает эффективную загрузку большого количества файлов данных со значительным общим объёмом данных. Pipe автоматически разделяет файлы по их количеству или размеру, разбивая задание загрузки на меньшие, последовательные задачи. Этот подход гарантирует, что ошибки в одном файле не влияют на всё задание загрузки. Статус загрузки каждого файла записывается Pipe, позволяя вам легко идентифицировать и исправлять файлы, содержащие ошибки. Минимизируя необходимость повторных попыток из-за ошибок данных, этот подход помогает снизить затраты.

- **Непрерывная загрузка данных помогает сократить трудозатраты.**

  Pipe помогает записывать новые или обновлённые файлы данных в определённое место и непрерывно загружать новые данные из этих файлов в Selena. После создания задания Pipe с указанным `"AUTO_INGEST" = "TRUE"` оно будет постоянно отслеживать изменения в файлах данных, хранящихся по указанному пути, и автоматически загружать новые или обновлённые данные из файлов данных в целевую таблицу Selena.

Кроме того, Pipe выполняет проверки уникальности файлов для предотвращения дублирования загрузки данных. Во время процесса загрузки Pipe проверяет уникальность каждого файла данных на основе имени файла и дайджеста. Если файл с определённым именем и дайджестом уже был обработан заданием Pipe, задание Pipe пропустит все последующие файлы с тем же именем и дайджестом. Обратите внимание, что объектное хранилище, такое как AWS S3, использует `ETag` в качестве дайджеста файла.

Статус загрузки каждого файла данных записывается и сохраняется в представлении `information_schema.pipe_files`. После удаления задания Pipe, связанного с представлением, записи о загруженных в этом задании файлах также будут удалены.

Различия между Pipe и INSERT+FILES()

Задание Pipe разделяется на одну или несколько транзакций в зависимости от размера и количества строк в каждом файле данных. Пользователи могут запрашивать промежуточные результаты во время процесса загрузки. В отличие от этого, задание INSERT+`FILES()` обрабатывается как одна транзакция, и пользователи не могут просматривать данные во время процесса загрузки.

Последовательность загрузки файлов

Для каждого задания Pipe Selena поддерживает очередь файлов, из которой извлекает и загружает файлы данных как микропакеты. Pipe не гарантирует, что файлы данных загружаются в том же порядке, в котором они были загружены. Поэтому более новые данные могут быть загружены раньше более старых данных.

Типичный пример

Создание базы данных и таблицы

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Создайте таблицу вручную (мы рекомендуем, чтобы таблица имела ту же схему, что и файл Parquet, который вы хотите загрузить из AWS S3):

```
CREATE TABLE user_behavior_from_pipe
(
    UserID int(11),
    ItemID int(11),
    CategoryID int(11),
    BehaviorType varchar(65533),
    Timestamp datetime
)
ENGINE = OLAP
DUPLICATE KEY(UserID)
DISTRIBUTED BY HASH(UserID);
```

Запуск задания Pipe

Выполните следующую команду, чтобы запустить задание Pipe, которое загружает данные из примера набора данных `s3://selena-examples/user-behavior-10-million-rows/` в таблицу `user_behavior_from_pipe`. Это задание pipe использует как микропакеты, так и непрерывную загрузку (описанную выше) — специфические функции pipe.

В других примерах в этом руководстве загружается один файл Parquet с 10 миллионами строк. Для примера с pipe тот же набор данных разбит на 57 отдельных файлов, и все они хранятся в одной папке S3. Обратите внимание, что в команде `CREATE PIPE` ниже `path` — это URI для папки S3, и вместо указания имени файла URI заканчивается на `/*`. Установив `AUTO_INGEST` и указав папку вместо отдельного файла, задание pipe будет опрашивать папку S3 на наличие новых файлов и загружать их по мере их добавления в папку.

```
CREATE PIPE user_behavior_pipe
PROPERTIES
(
    "AUTO_INGEST" = "TRUE"
)
AS
INSERT INTO user_behavior_from_pipe
SELECT * FROM FILES
(
    "path" = "s3://selena-examples/user-behavior-10-million-rows/*",
    "format" = "parquet",
    "aws.s3.region" = "us-east-1",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
);
```

> **ПРИМЕЧАНИЕ**
>
> Замените `AAA` и `BBB` на свои учётные данные в приведённой выше команде. Можно использовать любой действительный `aws.s3.access_key` и `aws.s3.secret_key`, так как объект доступен для чтения любому аутентифицированному пользователю AWS.

Это задание имеет четыре основных раздела:

- `pipe_name`: Имя pipe. Имя pipe должно быть уникальным в пределах базы данных, к которой принадлежит pipe.
- `INSERT_SQL`: Оператор INSERT INTO SELECT FROM FILES, который используется для загрузки данных из указанного исходного файла данных в целевую таблицу.
- `PROPERTIES`: Набор необязательных параметров, которые указывают, как выполнять pipe. К ним относятся `AUTO_INGEST`, `POLL_INTERVAL`, `BATCH_SIZE` и `BATCH_FILES`. Указывайте эти свойства в формате `"key" = "value"`.

Подробный синтаксис и описание параметров см. в CREATE PIPE.

Проверка прогресса загрузки

- Запросите прогресс задания Pipe с помощью SHOW PIPES в текущей базе данных, к которой принадлежит задание Pipe.

```
SHOW PIPES WHERE NAME = 'user_behavior_pipe' \G
```

Возвращается следующий результат:

> **ПОДСКАЗКА**
>
> В выводе, показанном ниже, pipe находится в состоянии `RUNNING`. Pipe будет оставаться в состоянии `RUNNING`, пока вы вручную не остановите его. Вывод также показывает количество загруженных файлов (57) и время последней загрузки файла.

```
*************************** 1. row ***************************
DATABASE_NAME: mydatabase
      PIPE_ID: 10476
    PIPE_NAME: user_behavior_pipe
        STATE: RUNNING
   TABLE_NAME: mydatabase.user_behavior_from_pipe
  LOAD_STATUS:
{"loadedFiles":57,"loadedBytes":295345637,"loadingFiles":0,"lastLoadedTim
02-28 22:14:19"}
   LAST_ERROR: NULL
 CREATED_TIME: 2024-02-28 22:13:41
1 row in set (0.02 sec)
```

- Запросите прогресс задания Pipe из представления `pipes` в Information Schema Selena.

```
SELECT * FROM information_schema.pipes WHERE pipe_name =
'user_behavior_replica' \G
```

Возвращается следующий результат:

> **ПОДСКАЗКА**
>
> Некоторые запросы в этом руководстве заканчиваются на `\G` вместо точки с запятой (`;`). Это заставляет клиент MySQL выводить результаты в вертикальном формате. Если вы используете DBeaver или другой клиент, вам может потребоваться использовать точку с запятой (`;`) вместо `\G`.

```
*************************** 1. row ***************************
DATABASE_NAME: mydatabase
      PIPE_ID: 10217
    PIPE_NAME: user_behavior_replica
        STATE: RUNNING
   TABLE_NAME: mydatabase.user_behavior_replica
  LOAD_STATUS:
{"loadedFiles":1,"loadedBytes":132251298,"loadingFiles":0,"lastLoadedTime
11-09 15:35:42"}
   LAST_ERROR:
 CREATED_TIME: 9891-01-15 07:51:45
1 row in set (0.01 sec)
```

Проверка статуса файлов

Вы можете запросить статус загрузки файлов из представления `pipe_files` в Information Schema Selena.

```
SELECT * FROM information_schema.pipe_files WHERE pipe_name =
'user_behavior_replica' \G
```

Возвращается следующий результат:

```
*************************** 1. row ***************************
   DATABASE_NAME: mydatabase
         PIPE_ID: 10217
       PIPE_NAME: user_behavior_replica
       FILE_NAME: s3://selena-examples/user-behavior-10-million-
rows.parquet
    FILE_VERSION: e29daa86b1120fea58ad0d047e671787-8
       FILE_SIZE: 132251298
   LAST_MODIFIED: 2023-11-06 13:25:17
      LOAD_STATE: FINISHED
     STAGED_TIME: 2023-11-09 15:35:02
 START_LOAD_TIME: 2023-11-09 15:35:03
FINISH_LOAD_TIME: 2023-11-09 15:35:42
       ERROR_MSG:
1 row in set (0.03 sec)
```

Управление заданиями Pipe

Вы можете изменять, приостанавливать или возобновлять, удалять или запрашивать созданные вами pipe, а также повторять загрузку определённых файлов данных. Для получения дополнительной информации см. ALTER PIPE, SUSPEND or RESUME PIPE, DROP PIPE, SHOW PIPES и RETRY FILE.

3.2.2 Загрузка данных из Microsoft Azure Storage

Selena предоставляет следующие варианты загрузки данных из Azure:

- Синхронная загрузка с использованием INSERT+`FILES()`
- Асинхронная загрузка с использованием Broker Load

Каждый из этих вариантов имеет свои преимущества, которые подробно описаны в следующих разделах.

В большинстве случаев мы рекомендуем использовать метод INSERT+`FILES()`, который гораздо проще в использовании.

Однако метод INSERT+`FILES()` в настоящее время поддерживает только форматы файлов Parquet, ORC и CSV. Поэтому, если вам нужно загрузить данные других форматов файлов, таких как JSON, или выполнить изменения данных, такие как DELETE, во время загрузки данных, вы можете использовать Broker Load.

Перед началом работы

Подготовка исходных данных

Убедитесь, что исходные данные, которые вы хотите загрузить в Selena, правильно хранятся в контейнере внутри вашей учётной записи хранения Azure.

В этой теме предположим, что вы хотите загрузить данные из примера набора данных в формате Parquet (`user_behavior_ten_million_rows.parquet`), хранящегося в корневом каталоге контейнера (`selena-container`) внутри учётной записи хранения Azure Data Lake Storage Gen2 (ADLS Gen2) (`selena`).

Проверка привилегий

Вы можете загружать данные в таблицы Selena только как пользователь, имеющий привилегию INSERT на эти таблицы Selena. Если у вас нет привилегии INSERT, следуйте

инструкциям в разделе GRANT, чтобы предоставить привилегию INSERT пользователю, которого вы используете для подключения к вашему cluster Selena. Синтаксис: `GRANT INSERT ON TABLE <table_name> IN DATABASE <database_name> TO { ROLE <role_name> | USER <user_identity>}`.

Сбор данных аутентификации

Примеры в этой теме используют метод аутентификации Shared Key. Чтобы убедиться, что у вас есть разрешение на чтение данных из ADLS Gen2, мы рекомендуем прочитать Azure Data Lake Storage Gen2 > Shared Key (ключ доступа учётной записи хранения), чтобы понять параметры аутентификации, которые вам нужно настроить.

Вкратце, если вы используете аутентификацию Shared Key, вам нужно собрать следующую информацию:

- Имя пользователя вашей учётной записи хранения ADLS Gen2
- Shared key вашей учётной записи хранения ADLS Gen2

Для получения информации обо всех доступных методах аутентификации см. Аутентификация в облачном хранилище Azure.

Использование INSERT+FILES()

Этот метод доступен начиная с версии v1.5.2 и в настоящее время поддерживает только форматы файлов Parquet, ORC и CSV (начиная с версии v1.5.2).

Преимущества INSERT+FILES()

`FILES()` может читать файл, хранящийся в облачном хранилище, на основе указанных вами свойств, связанных с путём, выводить схему таблицы данных в файле, а затем возвращать данные из файла в виде строк данных.

С помощью `FILES()` вы можете:

- Запрашивать данные напрямую из Azure с помощью SELECT.
- Создавать и загружать таблицу с помощью CREATE TABLE AS SELECT (CTAS).
- Загружать данные в существующую таблицу с помощью INSERT.

Типичные примеры

Прямой запрос из Azure с помощью SELECT

Прямой запрос из Azure с помощью SELECT+`FILES()` позволяет получить хороший предварительный просмотр содержимого набора данных перед созданием таблицы. Например:

- Получить предварительный просмотр набора данных без сохранения данных.
- Запросить минимальные и максимальные значения и решить, какие типы данных использовать.
- Проверить наличие значений `NULL`.

Следующий пример запрашивает пример набора данных `user_behavior_ten_million_rows.parquet`, хранящийся в контейнере `selena-container` внутри вашей учётной записи хранения `selena`:

```
SELECT * FROM FILES
(
    "path" = "abfss://selena-
container@selena.dfs.core.windows.net/user_behavior_ten_million_rows.parquet"
    "format" = "parquet",
    "azure.adls2.storage_account" = "selena",
    "azure.adls2.shared_key" = "xxxxxxxxxxxxxxxxxx"
)
LIMIT 3;
```

Система возвращает результат запроса, аналогичный следующему:

```
+--------+---------+------------+--------------+---------------------+
| UserID | ItemID  | CategoryID | BehaviorType | Timestamp           |
+--------+---------+------------+--------------+---------------------+
| 543711 |  829192 |    2355072 | pv           | 2017-11-27 08:22:37 |
| 543711 | 2056618 |    3645362 | pv           | 2017-11-27 10:16:46 |
| 543711 | 1165492 |    3645362 | pv           | 2017-11-27 10:17:00 |
+--------+---------+------------+--------------+---------------------+
```

> **ПРИМЕЧАНИЕ**

> Обратите внимание, что имена столбцов, возвращённые выше, предоставлены файлом Parquet.

Создание и загрузка таблицы с помощью CTAS

Это продолжение предыдущего примера. Предыдущий запрос обёрнут в CREATE TABLE AS SELECT (CTAS) для автоматизации создания таблицы с использованием вывода схемы. Это означает, что Selena выведет схему таблицы, создаст нужную вам таблицу, а затем загрузит данные в таблицу. Имена и типы столбцов не требуются для создания таблицы при использовании табличной функции `FILES()` с файлами Parquet, поскольку формат Parquet включает имена столбцов.

> **ПРИМЕЧАНИЕ**
>
> Синтаксис CREATE TABLE при использовании вывода схемы не позволяет устанавливать количество replica. Если вы используете cluster Selena с shared-nothing, установите количество replica перед созданием таблицы. Пример ниже для системы с тремя replica:
>
> ```
> ADMIN SET FRONTEND CONFIG ('default_replication_num' = "3");
> ```

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Используйте CTAS для создания таблицы и загрузки данных из примера набора данных `user_behavior_ten_million_rows.parquet`, который хранится в контейнере `selena-container` внутри вашей учётной записи хранения `selena`, в таблицу:

```
CREATE TABLE user_behavior_inferred AS
SELECT * FROM FILES
(
    "path" = "abfss://selena-
container@selena.dfs.core.windows.net/user_behavior_ten_million_rows.parquet"
    "format" = "parquet",
    "azure.adls2.storage_account" = "selena",
    "azure.adls2.shared_key" = "xxxxxxxxxxxxxxxxxx"
);
```

После создания таблицы вы можете просмотреть её схему с помощью DESCRIBE:

```
DESCRIBE user_behavior_inferred;
```

Система возвращает следующий результат запроса:

```
+--------------+-----------+------+-------+---------+-------+
| Field        | Type      | Null | Key   | Default | Extra |
+--------------+-----------+------+-------+---------+-------+
| UserID       | bigint    | YES  | true  | NULL    |       |
| ItemID       | bigint    | YES  | true  | NULL    |       |
| CategoryID   | bigint    | YES  | true  | NULL    |       |
| BehaviorType | varbinary | YES  | false | NULL    |       |
| Timestamp    | varbinary | YES  | false | NULL    |       |
+--------------+-----------+------+-------+---------+-------+
```

Запросите таблицу, чтобы убедиться, что данные были загружены в неё. Пример:

```
SELECT * from user_behavior_inferred LIMIT 3;
```

Возвращается следующий результат запроса, указывающий, что данные были успешно загружены:

```
+--------+--------+------------+--------------+---------------------+
| UserID | ItemID | CategoryID | BehaviorType | Timestamp           |
+--------+--------+------------+--------------+---------------------+
|     84 | 162325 |    2939262 | pv           | 2017-12-02 05:41:41 |
|     84 | 232622 |    4148053 | pv           | 2017-11-27 04:36:10 |
|     84 | 595303 |     903809 | pv           | 2017-11-26 08:03:59 |
+--------+--------+------------+--------------+---------------------+
```

Загрузка в существующую таблицу с помощью INSERT

Вы можете захотеть настроить таблицу, в которую вставляете данные, например:

- тип данных столбца, настройку nullable или значения по умолчанию
- типы ключей и столбцы
- разбиение данных на partition и bucket

> **ПРИМЕЧАНИЕ**
>
> Создание наиболее эффективной структуры таблицы требует знания того, как данные будут использоваться и содержимого столбцов. Эта тема не охватывает проектирование таблиц. Для получения информации о проектировании таблиц см. Типы таблиц.

В этом примере мы создаём таблицу на основе знаний о том, как таблица будет запрашиваться, и данных в файле Parquet. Знание данных в файле Parquet можно получить, запросив файл напрямую в Azure.

- Поскольку запрос набора данных в Azure указывает, что столбец `Timestamp` содержит данные, соответствующие типу данных VARBINARY, тип столбца указан в следующем DDL.
- Запрашивая данные в Azure, вы можете обнаружить, что в наборе данных нет значений `NULL`, поэтому DDL не устанавливает какие-либо столбцы как nullable.
- На основе знаний об ожидаемых типах запросов sort key и столбец для bucket установлены на столбец `UserID`. Ваш случай использования может отличаться для этих данных, поэтому вы можете решить использовать `ItemID` в дополнение к `UserID` или вместо него для sort key.

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Создайте таблицу вручную (мы рекомендуем, чтобы таблица имела ту же схему, что и файл Parquet, который вы хотите загрузить из Azure):

```
CREATE TABLE user_behavior_declared
(
    UserID int(11),
    ItemID int(11),
    CategoryID int(11),
    BehaviorType varchar(65533),
    Timestamp varbinary
)
ENGINE = OLAP
DUPLICATE KEY(UserID)
DISTRIBUTED BY HASH(UserID);
```

Отобразите схему, чтобы вы могли сравнить её с выведенной схемой, созданной табличной функцией `FILES()`:

```
DESCRIBE user_behavior_declared;
```

```
+--------------+----------------+------+-------+---------+-------+
| Field        | Type           | Null | Key   | Default | Extra |
+--------------+----------------+------+-------+---------+-------+
| UserID       | int            | NO   | true  | NULL    |       |
| ItemID       | int            | NO   | false | NULL    |       |
| CategoryID   | int            | NO   | false | NULL    |       |
| BehaviorType | varchar(65533) | NO   | false | NULL    |       |
| Timestamp    | varbinary      | NO   | false | NULL    |       |
+--------------+----------------+------+-------+---------+-------+
5 rows in set (0.00 sec)
```

ПОДСКАЗКА

Сравните схему, которую вы только что создали, со схемой, выведенной ранее с помощью табличной функции `FILES()`. Посмотрите на:

- типы данных
- nullable
- ключевые поля

Для лучшего контроля схемы целевой таблицы и лучшей производительности запросов мы рекомендуем указывать схему таблицы вручную в производственных средах.

После создания таблицы вы можете загрузить её с помощью INSERT INTO SELECT FROM FILES():

```
INSERT INTO user_behavior_declared
SELECT * FROM FILES
(
    "path" = "abfss://selena-
container@selena.dfs.core.windows.net/user_behavior_ten_million_rows.parquet"
    "format" = "parquet",
    "azure.adls2.storage_account" = "selena",
    "azure.adls2.shared_key" = "xxxxxxxxxxxxxxxxxx"
);
```

После завершения загрузки вы можете запросить таблицу, чтобы убедиться, что данные были загружены в неё. Пример:

```
SELECT * from user_behavior_declared LIMIT 3;
```

Система возвращает результат запроса, аналогичный следующему, указывающий, что данные были успешно загружены:

```
 +--------+---------+------------+--------------+---------------------+
| UserID | ItemID  | CategoryID | BehaviorType | Timestamp           |
+--------+---------+------------+--------------+---------------------+
|    142 | 2869980 |    2939262 | pv           | 2017-11-25 03:43:22 |
|    142 | 2522236 |    1669167 | pv           | 2017-11-25 15:14:12 |
|    142 | 3031639 |    3607361 | pv           | 2017-11-25 15:19:25 |
+--------+---------+------------+--------------+---------------------+
```

Проверка прогресса загрузки

Вы можете запросить прогресс заданий INSERT из представления `loads` в Information Schema Selena. Эта функция поддерживается начиная с версии v1.5.2. Пример:

```
SELECT * FROM information_schema.loads ORDER BY JOB_ID DESC;
```

Для получения информации о полях, предоставляемых в представлении `loads`, см. `loads`.

Если вы отправили несколько заданий загрузки, вы можете фильтровать по `LABEL`, связанному с заданием. Пример:

```
SELECT * FROM information_schema.loads WHERE LABEL = 'insert_f3fc2298-a553-11
*************************** 1. row ***************************
              JOB_ID: 10193
               LABEL: insert_f3fc2298-a553-11ee-92f4-00163e0842bd
       DATABASE_NAME: mydatabase
               STATE: FINISHED
            PROGRESS: ETL:100%; LOAD:100%
                TYPE: INSERT
            PRIORITY: NORMAL
           SCAN_ROWS: 10000000
       FILTERED_ROWS: 0
     UNSELECTED_ROWS: 0
           SINK_ROWS: 10000000
            ETL_INFO:
           TASK_INFO: resource:N/A; timeout(s):300; max_filter_ratio:0.0
         CREATE_TIME: 2023-12-28 15:37:38
      ETL_START_TIME: 2023-12-28 15:37:38
     ETL_FINISH_TIME: 2023-12-28 15:37:38
     LOAD_START_TIME: 2023-12-28 15:37:38
    LOAD_FINISH_TIME: 2023-12-28 15:39:35
         JOB_DETAILS: {"All backends":{"f3fc2298-a553-11ee-92f4-00163e0842bd"
[10120]},"FileNumber":0,"FileSize":0,"InternalTableLoadBytes":581730322,"Inte
backends":{"f3fc2298-a553-11ee-92f4-00163e0842bd":[]}}
           ERROR_MSG: NULL
        TRACKING_URL: NULL
        TRACKING_SQL: NULL
REJECTED_RECORD_PATH: NULL
```

> **ПРИМЕЧАНИЕ**
>
> INSERT — это синхронная команда. Если задание INSERT всё ещё выполняется, вам нужно открыть другую сессию, чтобы проверить его статус выполнения.

Использование Broker Load

Асинхронный процесс Broker Load обрабатывает подключение к Azure, извлечение данных и сохранение данных в Selena.

Этот метод поддерживает следующие форматы файлов:

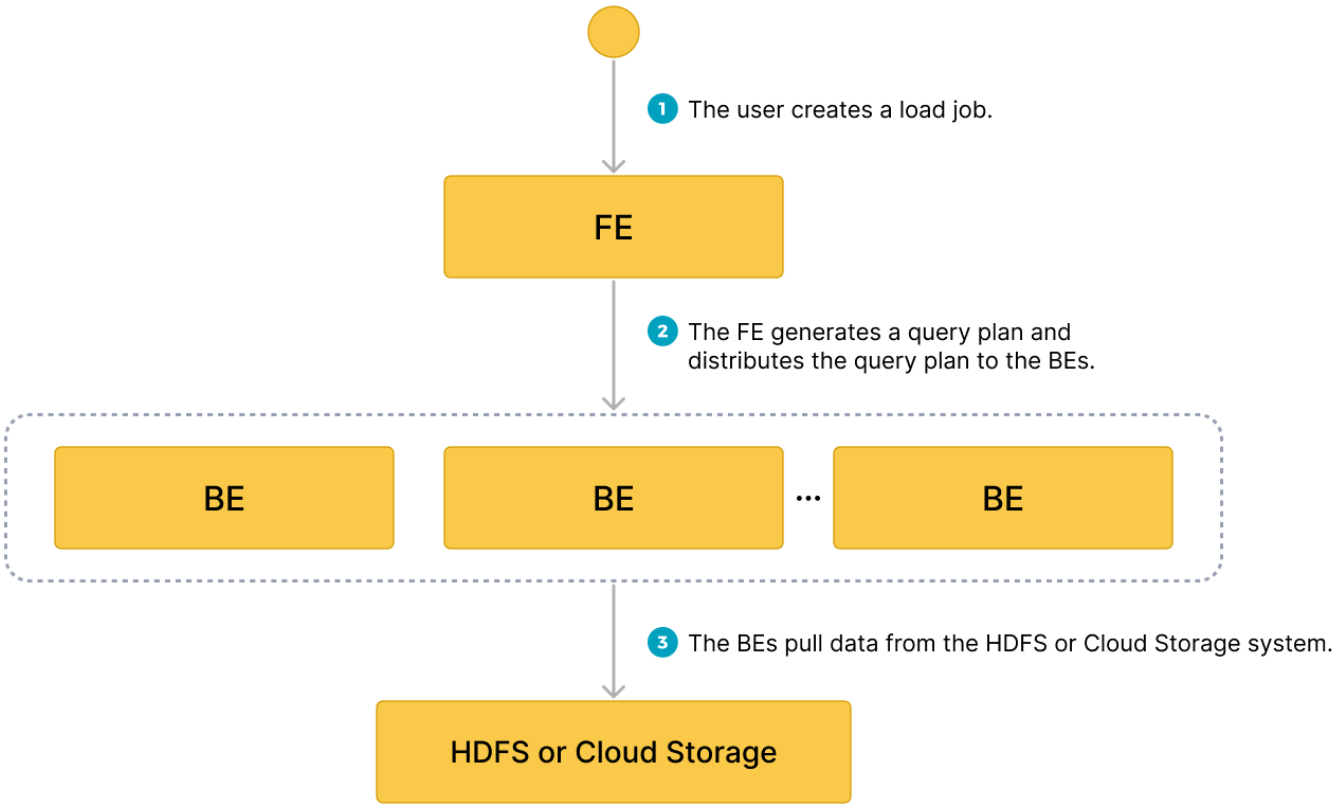
- Parquet
- ORC

- CSV
- JSON (поддерживается начиная с версии v1.5.2)

Преимущества Broker Load

- Broker Load выполняется в фоновом режиме, и клиентам не нужно оставаться подключёнными для продолжения задания.
- Broker Load предпочтителен для длительных заданий, тайм-аут по умолчанию составляет 4 часа.
- В дополнение к форматам файлов Parquet и ORC, Broker Load поддерживает формат файлов CSV и формат файлов JSON (формат файлов JSON поддерживается начиная с версии v1.5.2).

Поток данных



1. Пользователь создаёт задание загрузки.
2. Frontend (FE) создаёт план запроса и распределяет план на backend-узлы (BE) или compute-узлы (CN).
3. BE или CN извлекают данные из источника и загружают данные в Selena.

Типичный пример

Создайте таблицу, запустите процесс загрузки, который извлекает пример набора данных `user_behavior_ten_million_rows.parquet` из Azure, и проверьте прогресс и успешность загрузки данных.

Создание базы данных и таблицы

Подключитесь к вашему cluster Selena. Затем создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Создайте таблицу вручную (мы рекомендуем, чтобы таблица имела ту же схему, что и файл Parquet, который вы хотите загрузить из Azure):

```
CREATE TABLE user_behavior
(
    UserID int(11),
    ItemID int(11),
    CategoryID int(11),
    BehaviorType varchar(65533),
    Timestamp varbinary
)
ENGINE = OLAP
DUPLICATE KEY(UserID)
DISTRIBUTED BY HASH(UserID);
```

Запуск Broker Load

Выполните следующую команду, чтобы запустить задание Broker Load, которое загружает данные из примера набора данных `user_behavior_ten_million_rows.parquet` в таблицу `user_behavior`:

```
LOAD LABEL user_behavior
(
    DATA INFILE("abfss://selena-
container@selena.dfs.core.windows.net/user_behavior_ten_million_rows.parquet"
    INTO TABLE user_behavior
    FORMAT AS "parquet"
)
WITH BROKER
(
    "azure.adls2.storage_account" = "selena",
    "azure.adls2.shared_key" = "xxxxxxxxxxxxxxxxxx"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Это задание имеет четыре основных раздела:

- `LABEL`: Строка, используемая при запросе состояния задания загрузки.
- `LOAD` declaration: URI источника, формат исходных данных и имя целевой таблицы.
- `BROKER`: Данные подключения к источнику.
- `PROPERTIES`: Значение тайм-аута и любые другие свойства, применяемые к заданию загрузки.

Подробный синтаксис и описание параметров см. в BROKER LOAD.

Проверка прогресса загрузки

Вы можете запросить прогресс заданий Broker Load из представления `loads` в Information Schema Selena. Эта функция поддерживается начиная с версии v1.5.2.

```
SELECT * FROM information_schema.loads \G
```

Для получения информации о полях, предоставляемых в представлении `loads`, см. `loads`.

Если вы отправили несколько заданий загрузки, вы можете фильтровать по `LABEL`, связанному с заданием:

```
SELECT * FROM information_schema.loads WHERE LABEL = 'user_behavior' \G
*************************** 1. row ***************************
              JOB_ID: 10250
               LABEL: user_behavior
       DATABASE_NAME: mydatabase
               STATE: FINISHED
            PROGRESS: ETL:100%; LOAD:100%
                TYPE: BROKER
            PRIORITY: NORMAL
           SCAN_ROWS: 10000000
       FILTERED_ROWS: 0
     UNSELECTED_ROWS: 0
           SINK_ROWS: 10000000
            ETL_INFO:
           TASK_INFO: resource:N/A; timeout(s):3600; max_filter_ratio:0.0
         CREATE_TIME: 2023-12-28 16:15:19
      ETL_START_TIME: 2023-12-28 16:15:25
     ETL_FINISH_TIME: 2023-12-28 16:15:25
     LOAD_START_TIME: 2023-12-28 16:15:25
    LOAD_FINISH_TIME: 2023-12-28 16:16:31
         JOB_DETAILS: {"All backends":{"6a8ef4c0-1009-48c9-8d18-c4061d2255bf"
[10121]},"FileNumber":1,"FileSize":132251298,"InternalTableLoadBytes":3117107
backends":{"6a8ef4c0-1009-48c9-8d18-c4061d2255bf":[]}}
           ERROR_MSG: NULL
        TRACKING_URL: NULL
        TRACKING_SQL: NULL
REJECTED_RECORD_PATH: NULL
```

После того как вы убедитесь, что задание загрузки завершено, вы можете проверить подмножество целевой таблицы, чтобы увидеть, были ли данные успешно загружены. Пример:

```
SELECT * from user_behavior LIMIT 3;
```

Система возвращает результат запроса, аналогичный следующему, указывающий, что данные были успешно загружены:

```
+--------+---------+------------+--------------+---------------------+
| UserID | ItemID  | CategoryID | BehaviorType | Timestamp           |
+--------+---------+------------+--------------+---------------------+
|    142 | 2869980 |    2939262 | pv           | 2017-11-25 03:43:22 |
|    142 | 2522236 |    1669167 | pv           | 2017-11-25 15:14:12 |
|    142 | 3031639 |    3607361 | pv           | 2017-11-25 15:19:25 |
+--------+---------+------------+--------------+---------------------+
```

3.2.3 Загрузка данных из GCS

Selena предоставляет следующие варианты загрузки данных из GCS:

- Синхронная загрузка с использованием INSERT+`FILES()`
- Асинхронная загрузка с использованием Broker Load

Каждый из этих вариантов имеет свои преимущества, которые подробно описаны в следующих разделах.

В большинстве случаев мы рекомендуем использовать метод INSERT+`FILES()`, который гораздо проще в использовании.

Однако метод INSERT+`FILES()` в настоящее время поддерживает только форматы файлов Parquet, ORC и CSV. Поэтому, если вам нужно загрузить данные других форматов файлов, таких как JSON, или выполнить изменения данных, такие как DELETE, во время загрузки данных, вы можете использовать Broker Load.

Перед началом работы

Подготовка исходных данных

Убедитесь, что исходные данные, которые вы хотите загрузить в Selena, правильно хранятся в bucket GCS. Вы также можете рассмотреть, где находятся данные и база данных, потому что затраты на передачу данных значительно ниже, когда ваш bucket и ваш cluster Selena находятся в одном регионе.

В этой теме мы предоставляем вам пример набора данных в bucket GCS, `gs://selena-samples/user_behavior_ten_million_rows.parquet`. Вы можете получить доступ к этому набору данных с любыми действительными учётными данными, так как объект доступен для чтения любому пользователю GCP.

Проверка привилегий

Вы можете загружать данные в таблицы Selena только как пользователь, имеющий привилегию INSERT на эти таблицы Selena. Если у вас нет привилегии INSERT, следуйте инструкциям в разделе GRANT, чтобы предоставить привилегию INSERT пользователю, которого вы используете для подключения к вашему cluster Selena. Синтаксис: `GRANT INSERT ON TABLE <table_name> IN DATABASE <database_name> TO { ROLE <role_name> | USER <user_identity>}`.

Сбор данных аутентификации

Примеры в этой теме используют аутентификацию на основе сервисной учётной записи. Чтобы практиковать аутентификацию на основе IAM-пользователя, вам нужно собрать информацию о следующих ресурсах GCS:

- Bucket GCS, в котором хранятся ваши данные.
- Ключ объекта GCS (имя объекта), если вы обращаетесь к определённому объекту в bucket. Обратите внимание, что ключ объекта может включать префикс, если ваши объекты GCS хранятся в подпапках.
- Регион GCS, к которому принадлежит bucket GCS.
- `private_key_id`, `private_key` и `client_email` вашей сервисной учётной записи Google Cloud

Для получения информации обо всех доступных методах аутентификации см. Аутентификация в Google Cloud Storage.

Использование INSERT+FILES()

Этот метод доступен начиная с версии v1.5.2 и в настоящее время поддерживает только форматы файлов Parquet, ORC и CSV (начиная с версии v1.5.2).

Преимущества INSERT+FILES()

`FILES()` может читать файл, хранящийся в облачном хранилище, на основе указанных вами свойств, связанных с путём, выводить схему таблицы данных в файле, а затем возвращать данные из файла в виде строк данных.

С помощью `FILES()` вы можете:

- Запрашивать данные напрямую из GCS с помощью SELECT.

- Создавать и загружать таблицу с помощью CREATE TABLE AS SELECT (CTAS).
- Загружать данные в существующую таблицу с помощью INSERT.

Типичные примеры

Прямой запрос из GCS с помощью SELECT

Прямой запрос из GCS с помощью SELECT+`FILES()` позволяет получить хороший предварительный просмотр содержимого набора данных перед созданием таблицы. Например:

- Получить предварительный просмотр набора данных без сохранения данных.
- Запросить минимальные и максимальные значения и решить, какие типы данных использовать.
- Проверить наличие значений `NULL`.

Следующий пример запрашивает пример набора данных `gs://selena-samples/user_behavior_ten_million_rows.parquet`:

```
SELECT * FROM FILES
(
    "path" = "gs://selena-samples/user_behavior_ten_million_rows.parquet",
    "format" = "parquet",
    "gcp.gcs.service_account_email" = "sampledatareader@xxxxx-xxxxxx-
000000.iam.gserviceaccount.com",
    "gcp.gcs.service_account_private_key_id" =
"baaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
    "gcp.gcs.service_account_private_key" = "-----BEGIN PRIVATE KEY-----
\n...\n-----END PRIVATE KEY-----"
)
LIMIT 3;
```

> **ПРИМЕЧАНИЕ**
>
> Замените учётные данные в приведённой выше команде на свои собственные. Можно использовать любой действительный email сервисной учётной записи, ключ и секрет, так как объект доступен для чтения любому аутентифицированному пользователю GCP.

Система возвращает результат запроса, аналогичный следующему:

```
+--------+---------+------------+--------------+---------------------+
| UserID | ItemID  | CategoryID | BehaviorType | Timestamp           |
+--------+---------+------------+--------------+---------------------+
| 543711 |  829192 |    2355072 | pv           | 2017-11-27 08:22:37 |
| 543711 | 2056618 |    3645362 | pv           | 2017-11-27 10:16:46 |
| 543711 | 1165492 |    3645362 | pv           | 2017-11-27 10:17:00 |
+--------+---------+------------+--------------+---------------------+
```

> **ПРИМЕЧАНИЕ**
>
> Обратите внимание, что имена столбцов, возвращённые выше, предоставлены файлом Parquet.

Создание и загрузка таблицы с помощью CTAS

Это продолжение предыдущего примера. Предыдущий запрос обёрнут в CREATE TABLE AS SELECT (CTAS) для автоматизации создания таблицы с использованием вывода схемы. Это означает, что Selena выведет схему таблицы, создаст нужную вам таблицу, а затем загрузит данные в таблицу. Имена и типы столбцов не требуются для создания таблицы при использовании табличной функции `FILES()` с файлами Parquet, поскольку формат Parquet включает имена столбцов.

> **ПРИМЕЧАНИЕ**
>
> Синтаксис CREATE TABLE при использовании вывода схемы не позволяет устанавливать количество replica. Если вы используете cluster Selena с shared-nothing, установите количество replica перед созданием таблицы. Пример ниже для системы с тремя replica:
>
> ```
> ADMIN SET FRONTEND CONFIG ('default_replication_num' = "3");
> ```

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Используйте CTAS для создания таблицы и загрузки данных из примера набора данных `gs://selena-samples/user_behavior_ten_million_rows.parquet` в таблицу:

```
CREATE TABLE user_behavior_inferred AS
SELECT * FROM FILES
(
    "path" = "gs://selena-samples/user_behavior_ten_million_rows.parquet",
    "format" = "parquet",
    "gcp.gcs.service_account_email" = "sampledatareader@xxxxx-xxxxxx-
000000.iam.gserviceaccount.com",
    "gcp.gcs.service_account_private_key_id" =
"baaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
    "gcp.gcs.service_account_private_key" = "-----BEGIN PRIVATE KEY-----
\n...\n-----END PRIVATE KEY-----"
);
```

> **ПРИМЕЧАНИЕ**
>
> Замените учётные данные в приведённой выше команде на свои собственные. Можно использовать любой действительный email сервисной учётной записи, ключ и секрет, так как объект доступен для чтения любому аутентифицированному пользователю GCP.

После создания таблицы вы можете просмотреть её схему с помощью DESCRIBE:

```
DESCRIBE user_behavior_inferred;
```

Система возвращает результат запроса, аналогичный следующему:

```
+--------------+-----------+------+-------+---------+-------+
| Field        | Type      | Null | Key   | Default | Extra |
+--------------+-----------+------+-------+---------+-------+
| UserID       | bigint    | YES  | true  | NULL    |       |
| ItemID       | bigint    | YES  | true  | NULL    |       |
| CategoryID   | bigint    | YES  | true  | NULL    |       |
| BehaviorType | varbinary | YES  | false | NULL    |       |
| Timestamp    | varbinary | YES  | false | NULL    |       |
+--------------+-----------+------+-------+---------+-------+
```

Запросите таблицу, чтобы убедиться, что данные были загружены в неё. Пример:

```
SELECT * from user_behavior_inferred LIMIT 3;
```

Возвращается следующий результат запроса, указывающий, что данные были успешно загружены:

```
+--------+--------+------------+--------------+---------------------+
| UserID | ItemID | CategoryID | BehaviorType | Timestamp           |
+--------+--------+------------+--------------+---------------------+
|     84 | 162325 |    2939262 | pv           | 2017-12-02 05:41:41 |
|     84 | 232622 |    4148053 | pv           | 2017-11-27 04:36:10 |
|     84 | 595303 |     903809 | pv           | 2017-11-26 08:03:59 |
+--------+--------+------------+--------------+---------------------+
```

Загрузка в существующую таблицу с помощью INSERT

Вы можете захотеть настроить таблицу, в которую вставляете данные, например:

- тип данных столбца, настройку nullable или значения по умолчанию
- типы ключей и столбцы
- разбиение данных на partition и bucket

> **ПРИМЕЧАНИЕ**
>
> Создание наиболее эффективной структуры таблицы требует знания того, как данные будут использоваться и содержимого столбцов. Эта тема не охватывает проектирование таблиц. Для получения информации о проектировании таблиц см. Типы таблиц.

В этом примере мы создаём таблицу на основе знаний о том, как таблица будет запрашиваться, и данных в файле Parquet. Знание данных в файле Parquet можно получить, запросив файл напрямую в GCS.

- Поскольку запрос набора данных в GCS указывает, что столбец `Timestamp` содержит данные, соответствующие типу данных VARBINARY, тип столбца указан в следующем DDL.
- Запрашивая данные в GCS, вы можете обнаружить, что в наборе данных нет значений `NULL`, поэтому DDL не устанавливает какие-либо столбцы как nullable.
- На основе знаний об ожидаемых типах запросов sort key и столбец для bucket установлены на столбец `UserID`. Ваш случай использования может отличаться для

этих данных, поэтому вы можете решить использовать `ItemID` в дополнение к `UserID` или вместо него для sort key.

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Создайте таблицу вручную (мы рекомендуем, чтобы таблица имела ту же схему, что и файл Parquet, который вы хотите загрузить из GCS):

```
CREATE TABLE user_behavior_declared
(
    UserID int(11),
    ItemID int(11),
    CategoryID int(11),
    BehaviorType varchar(65533),
    Timestamp varbinary
)
ENGINE = OLAP
DUPLICATE KEY(UserID)
DISTRIBUTED BY HASH(UserID);
```

Отобразите схему, чтобы вы могли сравнить её с выведенной схемой, созданной табличной функцией `FILES()`:

```
DESCRIBE user_behavior_declared;
```

```
+--------------+----------------+------+-------+---------+-------+
| Field        | Type           | Null | Key   | Default | Extra |
+--------------+----------------+------+-------+---------+-------+
| UserID       | int            | NO   | true  | NULL    |       |
| ItemID       | int            | NO   | false | NULL    |       |
| CategoryID   | int            | NO   | false | NULL    |       |
| BehaviorType | varchar(65533) | NO   | false | NULL    |       |
| Timestamp    | varbinary      | NO   | false | NULL    |       |
+--------------+----------------+------+-------+---------+-------+
5 rows in set (0.00 sec)
```

ПОДСКАЗКА

Сравните схему, которую вы только что создали, со схемой, выведенной ранее с помощью табличной функции `FILES()`. Посмотрите на:

- типы данных
- nullable
- ключевые поля

Для лучшего контроля схемы целевой таблицы и лучшей производительности запросов мы рекомендуем указывать схему таблицы вручную в производственных средах.

После создания таблицы вы можете загрузить её с помощью INSERT INTO SELECT FROM FILES():

```
INSERT INTO user_behavior_declared
  SELECT * FROM FILES
  (
    "path" = "gs://selena-samples/user_behavior_ten_million_rows.parquet",
    "format" = "parquet",
    "gcp.gcs.service_account_email" = "sampledatareader@xxxxx-xxxxxx-
000000.iam.gserviceaccount.com",
    "gcp.gcs.service_account_private_key_id" =
"baaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
    "gcp.gcs.service_account_private_key" = "-----BEGIN PRIVATE KEY-----
\n...\n-----END PRIVATE KEY-----"
);
```

ПРИМЕЧАНИЕ

Замените учётные данные в приведённой выше команде на свои собственные. Можно использовать любой действительный email сервисной учётной записи, ключ и секрет, так как объект доступен для чтения любому аутентифицированному пользователю GCP.

После завершения загрузки вы можете запросить таблицу, чтобы убедиться, что данные были загружены в неё. Пример:

```
SELECT * from user_behavior_declared LIMIT 3;
```

Система возвращает результат запроса, аналогичный следующему, указывающий, что данные были успешно загружены:

```
+--------+---------+------------+--------------+---------------------+
| UserID | ItemID  | CategoryID | BehaviorType | Timestamp           |
+--------+---------+------------+--------------+---------------------+
|    142 | 2869980 |    2939262 | pv           | 2017-11-25 03:43:22 |
|    142 | 2522236 |    1669167 | pv           | 2017-11-25 15:14:12 |
|    142 | 3031639 |    3607361 | pv           | 2017-11-25 15:19:25 |
+--------+---------+------------+--------------+---------------------+
```

Проверка прогресса загрузки

Вы можете запросить прогресс заданий INSERT из представления `loads` в Information Schema Selena. Эта функция поддерживается начиная с версии v1.5.2. Пример:

```
SELECT * FROM information_schema.loads ORDER BY JOB_ID DESC;
```

Для получения информации о полях, предоставляемых в представлении `loads`, см. `loads`.

Если вы отправили несколько заданий загрузки, вы можете фильтровать по `LABEL`, связанному с заданием. Пример:

```
SELECT * FROM information_schema.loads WHERE LABEL = 'insert_f3fc2298-a553-11
*************************** 1. row ***************************
              JOB_ID: 10193
               LABEL: insert_f3fc2298-a553-11ee-92f4-00163e0842bd
       DATABASE_NAME: mydatabase
               STATE: FINISHED
            PROGRESS: ETL:100%; LOAD:100%
                TYPE: INSERT
            PRIORITY: NORMAL
           SCAN_ROWS: 10000000
       FILTERED_ROWS: 0
     UNSELECTED_ROWS: 0
           SINK_ROWS: 10000000
            ETL_INFO:
           TASK_INFO: resource:N/A; timeout(s):300; max_filter_ratio:0.0
         CREATE_TIME: 2023-12-28 15:37:38
      ETL_START_TIME: 2023-12-28 15:37:38
     ETL_FINISH_TIME: 2023-12-28 15:37:38
     LOAD_START_TIME: 2023-12-28 15:37:38
    LOAD_FINISH_TIME: 2023-12-28 15:39:35
         JOB_DETAILS: {"All backends":{"f3fc2298-a553-11ee-92f4-00163e0842bd"
[10120]},"FileNumber":0,"FileSize":0,"InternalTableLoadBytes":581730322,"Inte
backends":{"f3fc2298-a553-11ee-92f4-00163e0842bd":[]}}
           ERROR_MSG: NULL
        TRACKING_URL: NULL
        TRACKING_SQL: NULL
REJECTED_RECORD_PATH: NULL
```

> **ПРИМЕЧАНИЕ**
>
> INSERT — это синхронная команда. Если задание INSERT всё ещё выполняется, вам нужно открыть другую сессию, чтобы проверить его статус выполнения.

Использование Broker Load

Асинхронный процесс Broker Load обрабатывает подключение к GCS, извлечение данных и сохранение данных в Selena.

Этот метод поддерживает следующие форматы файлов:

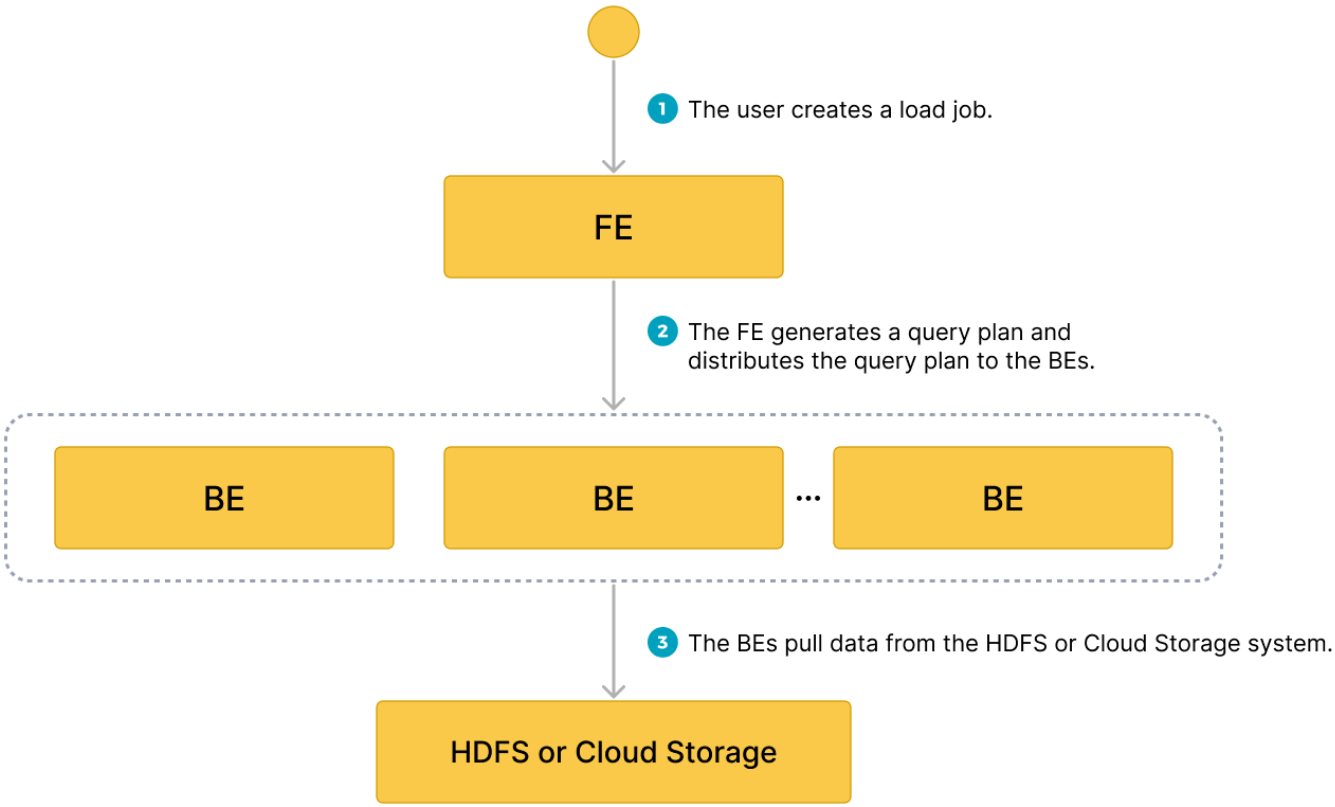
- Parquet
- ORC

- CSV
- JSON (поддерживается начиная с версии v1.5.2)

Преимущества Broker Load

- Broker Load выполняется в фоновом режиме, и клиентам не нужно оставаться подключёнными для продолжения задания.
- Broker Load предпочтителен для длительных заданий, тайм-аут по умолчанию составляет 4 часа.
- В дополнение к форматам файлов Parquet и ORC, Broker Load поддерживает формат файлов CSV и формат файлов JSON (формат файлов JSON поддерживается начиная с версии v1.5.2).

Поток данных



1. Пользователь создаёт задание загрузки.
2. Frontend (FE) создаёт план запроса и распределяет план на backend-узлы (BE) или compute-узлы (CN).
3. BE или CN извлекают данные из источника и загружают данные в Selena.

Типичный пример

Создайте таблицу, запустите процесс загрузки, который извлекает пример набора данных `gs://selena-samples/user_behavior_ten_million_rows.parquet` из GCS, и проверьте прогресс и успешность загрузки данных.

Создание базы данных и таблицы

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Создайте таблицу вручную (мы рекомендуем, чтобы таблица имела ту же схему, что и файл Parquet, который вы хотите загрузить из GCS):

```
CREATE TABLE user_behavior
(
    UserID int(11),
    ItemID int(11),
    CategoryID int(11),
    BehaviorType varchar(65533),
    Timestamp varbinary
)
ENGINE = OLAP
DUPLICATE KEY(UserID)
DISTRIBUTED BY HASH(UserID);
```

Запуск Broker Load

Выполните следующую команду, чтобы запустить задание Broker Load, которое загружает данные из примера набора данных `gs://selena-samples/user_behavior_ten_million_rows.parquet` в таблицу `user_behavior`:

```
LOAD LABEL user_behavior
(
    DATA INFILE("gs://selena-
samples/user_behavior_ten_million_rows.parquet")
    INTO TABLE user_behavior
    FORMAT AS "parquet"
 )
 WITH BROKER
 (

    "gcp.gcs.service_account_email" = "sampledatareader@xxxxx-xxxxxx-
000000.iam.gserviceaccount.com",
    "gcp.gcs.service_account_private_key_id" =
"baaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
    "gcp.gcs.service_account_private_key" = "-----BEGIN PRIVATE KEY-----
\n...\n-----END PRIVATE KEY-----"
 )
PROPERTIES
(
    "timeout" = "72000"
);
```

> **ПРИМЕЧАНИЕ**
>
> Замените учётные данные в приведённой выше команде на свои собственные. Можно использовать любой действительный email сервисной учётной записи, ключ и секрет, так как объект доступен для чтения любому аутентифицированному пользователю GCP.

Это задание имеет четыре основных раздела:

- `LABEL`: Строка, используемая при запросе состояния задания загрузки.
- `LOAD` declaration: URI источника, формат исходных данных и имя целевой таблицы.
- `BROKER`: Данные подключения к источнику.
- `PROPERTIES`: Значение тайм-аута и любые другие свойства, применяемые к заданию загрузки.

Подробный синтаксис и описание параметров см. в BROKER LOAD.

Проверка прогресса загрузки

Вы можете запросить прогресс заданий INSERT из представления `loads` в Information Schema Selena. Эта функция поддерживается начиная с версии v1.5.2.

```
SELECT * FROM information_schema.loads;
```

Для получения информации о полях, предоставляемых в представлении `loads`, см. `loads`.

Если вы отправили несколько заданий загрузки, вы можете фильтровать по `LABEL`, связанному с заданием. Пример:

```
SELECT * FROM information_schema.loads WHERE LABEL = 'user_behavior';
```

В выводе ниже есть две записи для задания загрузки `user_behavior`:

- Первая запись показывает состояние `CANCELLED`. Прокрутите до `ERROR_MSG`, и вы увидите, что задание завершилось неудачей из-за `listPath failed`.
- Вторая запись показывает состояние `FINISHED`, что означает, что задание выполнено успешно.

```
JOB_ID|LABEL                                          |DATABASE_NAME|STATE    |PR
|CREATE_TIME        |ETL_START_TIME     |ETL_FINISH_TIME    |LOAD_START_TIME
|ERROR_MSG                                |TRACKING_URL|TRACKING_SQL|REJECTED_RE
------+----------------------------------------------+-------------+---------+--
--------------------------------------+-------------------+------------------
-----------------------------------------------------------------------------
--------------------------+--------------------------------------+-----------
 10121|user_behavior                                 |mydatabase   |CANCELLED|ET
timeout(s):72000; max_filter_ratio:0.0|2023-08-10 14:59:30|
{},"FileNumber":0,"FileSize":0,"InternalTableLoadBytes":0,"InternalTableLoadR
|type:ETL_RUN_FAIL; msg:listPath failed|             |             |
 10106|user_behavior                                 |mydatabase   |FINISHED |ET
timeout(s):72000; max_filter_ratio:0.0|2023-08-10 14:50:15|2023-08-10 14:50:1
c7348f9a5f2f":
[10004]},"FileNumber":1,"FileSize":1225637388,"InternalTableLoadBytes":271060
backends":{"a5|                                          |             |
```

После того как вы убедитесь, что задание загрузки завершено, вы можете проверить подмножество целевой таблицы, чтобы увидеть, были ли данные успешно загружены. Пример:

```
SELECT * from user_behavior LIMIT 3;
```

Система возвращает результат запроса, аналогичный следующему, указывающий, что данные были успешно загружены:

```
+--------+---------+------------+--------------+---------------------+
| UserID | ItemID  | CategoryID | BehaviorType | Timestamp           |
+--------+---------+------------+--------------+---------------------+
|    142 | 2869980 |    2939262 | pv           | 2017-11-25 03:43:22 |
|    142 | 2522236 |    1669167 | pv           | 2017-11-25 15:14:12 |
|    142 | 3031639 |    3607361 | pv           | 2017-11-25 15:19:25 |
+--------+---------+------------+--------------+---------------------+
```

3.2.4 Загрузка данных из MinIO

Selena предоставляет следующие варианты загрузки данных из MinIO:

- Синхронная загрузка с использованием INSERT+`FILES()`
- Асинхронная загрузка с использованием Broker Load

Каждый из этих вариантов имеет свои преимущества, которые подробно описаны в следующих разделах.

В большинстве случаев мы рекомендуем использовать метод INSERT+`FILES()`, который гораздо проще в использовании.

Однако метод INSERT+`FILES()` в настоящее время поддерживает только форматы файлов Parquet, ORC и CSV. Поэтому, если вам нужно загрузить данные других форматов файлов, таких как JSON, или выполнить изменения данных, такие как DELETE, во время загрузки данных, вы можете использовать Broker Load.

Перед началом работы

Подготовка исходных данных

Убедитесь, что исходные данные, которые вы хотите загрузить в Selena, правильно хранятся в bucket MinIO. Вы также можете рассмотреть, где находятся данные и база данных, потому что затраты на передачу данных значительно ниже, когда ваш bucket и ваш cluster Selena находятся в одном регионе.

В этой теме мы предоставляем вам пример набора данных. Вы можете скачать его с помощью `curl`:

```
curl -O https://selena-examples.s3.amazonaws.com/user_behavior_ten_million_rows.parquet
```

Загрузите файл Parquet в вашу систему MinIO и запишите имя bucket. В примерах в этом руководстве используется имя bucket `/selena`.

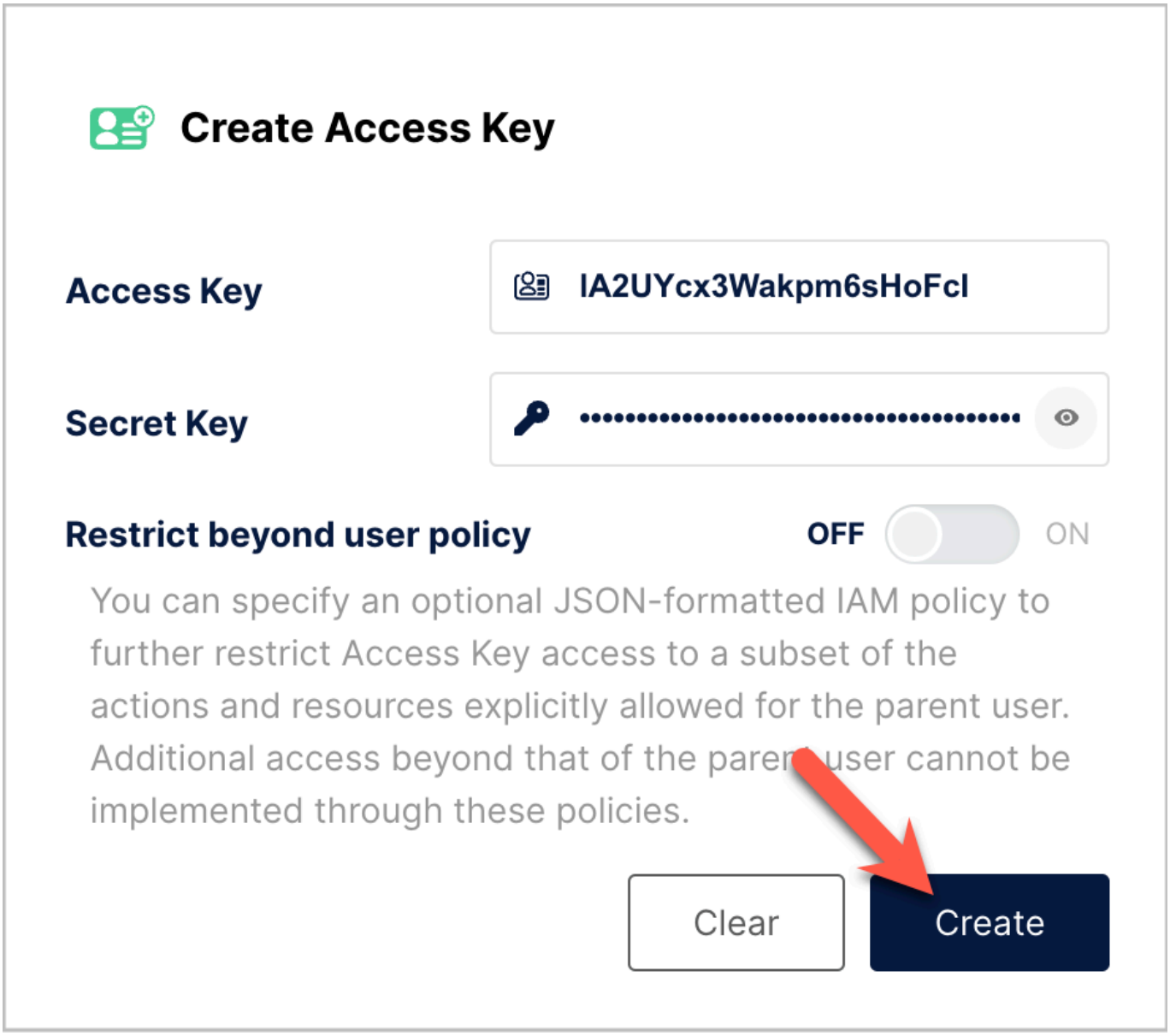
Проверка привилегий

Вы можете загружать данные в таблицы Selena только как пользователь, имеющий привилегию INSERT на эти таблицы Selena. Если у вас нет привилегии INSERT, следуйте инструкциям в разделе GRANT, чтобы предоставить привилегию INSERT пользователю, которого вы используете для подключения к вашему cluster Selena. Синтаксис: `GRANT INSERT ON TABLE <table_name> IN DATABASE <database_name> TO { ROLE <role_name> | USER <user_identity>}`.

Сбор данных подключения

Вкратце, для использования аутентификации MinIO Access Key вам нужно собрать следующую информацию:

- Bucket, в котором хранятся ваши данные
- Ключ объекта (имя объекта), если вы обращаетесь к определённому объекту в bucket
- Endpoint MinIO
- Access key и secret key, используемые в качестве учётных данных доступа.



Использование INSERT+FILES()

Этот метод доступен начиная с версии v1.5.2 и в настоящее время поддерживает только форматы файлов Parquet, ORC и CSV (начиная с версии v1.5.2).

Преимущества INSERT+FILES()

`FILES()` может читать файл, хранящийся в облачном хранилище, на основе указанных вами свойств, связанных с путём, выводить схему таблицы данных в файле, а затем возвращать данные из файла в виде строк данных.

С помощью `FILES()` вы можете:

- Запрашивать данные напрямую из MinIO с помощью SELECT.

- Создавать и загружать таблицу с помощью CREATE TABLE AS SELECT (CTAS).
- Загружать данные в существующую таблицу с помощью INSERT.

Типичные примеры

Прямой запрос из MinIO с помощью SELECT

Прямой запрос из MinIO с помощью SELECT+`FILES()` позволяет получить хороший предварительный просмотр содержимого набора данных перед созданием таблицы. Например:

- Получить предварительный просмотр набора данных без сохранения данных.
- Запросить минимальные и максимальные значения и решить, какие типы данных использовать.
- Проверить наличие значений `NULL`.

Следующий пример запрашивает пример набора данных, ранее добавленного в вашу систему MinIO.

ПОДСКАЗКА

Выделенный раздел команды включает настройки, которые вам может потребоваться изменить:

- Установите `endpoint` и `path` в соответствии с вашей системой MinIO.
- Если ваша система MinIO использует SSL, установите `enable_ssl` в `true`.
- Замените `AAA` и `BBB` на ваш access key и secret key MinIO.

```
SELECT * FROM FILES
(
    "aws.s3.endpoint" = "http://minio:9000",
    "path" = "s3://selena/user_behavior_ten_million_rows.parquet",
    "aws.s3.enable_ssl" = "false",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "format" = "parquet",
    "aws.s3.use_aws_sdk_default_behavior" = "false",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.enable_path_style_access" = "true"
)
LIMIT 3;
```

Система возвращает следующий результат запроса:

```
+--------+---------+------------+--------------+---------------------+
| UserID | ItemID  | CategoryID | BehaviorType | Timestamp           |
+--------+---------+------------+--------------+---------------------+
| 543711 |  829192 |    2355072 | pv           | 2017-11-27 08:22:37 |
| 543711 | 2056618 |    3645362 | pv           | 2017-11-27 10:16:46 |
| 543711 | 1165492 |    3645362 | pv           | 2017-11-27 10:17:00 |
+--------+---------+------------+--------------+---------------------+
3 rows in set (0.41 sec)
```

К СВЕДЕНИЮ

Обратите внимание, что имена столбцов, возвращённые выше, предоставлены файлом Parquet.

Создание и загрузка таблицы с помощью CTAS

Это продолжение предыдущего примера. Предыдущий запрос обёрнут в CREATE TABLE AS SELECT (CTAS) для автоматизации создания таблицы с использованием вывода схемы. Это означает, что Selena выведет схему таблицы, создаст нужную вам таблицу, а затем загрузит данные в таблицу. Имена и типы столбцов не требуются для создания таблицы при использовании табличной функции `FILES()` с файлами Parquet, поскольку формат Parquet включает имена столбцов.

ПРИМЕЧАНИЕ

Синтаксис CREATE TABLE при использовании вывода схемы не позволяет устанавливать количество replica, поэтому установите его перед созданием таблицы. Пример ниже для системы с одной replica:

```
ADMIN SET FRONTEND CONFIG ('default_replication_num' = '1');
```

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Используйте CTAS для создания таблицы и загрузки данных из примера набора данных, ранее добавленного в вашу систему MinIO.

ПОДСКАЗКА

Выделенный раздел команды включает настройки, которые вам может потребоваться изменить:

- Установите `endpoint` и `path` в соответствии с вашей системой MinIO.
- Если ваша система MinIO использует SSL, установите `enable_ssl` в `true`.
- Замените `AAA` и `BBB` на ваш access key и secret key MinIO.

```
CREATE TABLE user_behavior_inferred AS
SELECT * FROM FILES
(
    "aws.s3.endpoint" = "http://minio:9000",
    "path" = "s3://selena/user_behavior_ten_million_rows.parquet",
    "aws.s3.enable_ssl" = "false",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "format" = "parquet",
    "aws.s3.use_aws_sdk_default_behavior" = "false",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.enable_path_style_access" = "true"
);
```

```
Query OK, 10000000 rows affected (3.17 sec)
{'label':'insert_a5da3ff5-9ee4-11ee-90b0-02420a060004', 'status':'VISIBLE',
'txnId':'17'}
```

После создания таблицы вы можете просмотреть её схему с помощью DESCRIBE:

```
DESCRIBE user_behavior_inferred;
```

Система возвращает следующий результат запроса:

```
+--------------+------------------+------+-------+---------+-------+
| Field        | Type             | Null | Key   | Default | Extra |
+--------------+------------------+------+-------+---------+-------+
| UserID       | bigint           | YES  | true  | NULL    |       |
| ItemID       | bigint           | YES  | true  | NULL    |       |
| CategoryID   | bigint           | YES  | true  | NULL    |       |
| BehaviorType | varchar(1048576) | YES  | false | NULL    |       |
| Timestamp    | varchar(1048576) | YES  | false | NULL    |       |
+--------------+------------------+------+-------+---------+-------+
```

Запросите таблицу, чтобы убедиться, что данные были загружены в неё. Пример:

```
SELECT * from user_behavior_inferred LIMIT 3;
```

Возвращается следующий результат запроса, указывающий, что данные были успешно загружены:

```
+--------+--------+------------+--------------+---------------------+
| UserID | ItemID | CategoryID | BehaviorType | Timestamp           |
+--------+--------+------------+--------------+---------------------+
|     58 | 158350 |    2355072 | pv           | 2017-11-27 13:06:51 |
|     58 | 158590 |    3194735 | pv           | 2017-11-27 02:21:04 |
|     58 | 215073 |    3002561 | pv           | 2017-11-30 10:55:42 |
+--------+--------+------------+--------------+---------------------+
```

Загрузка в существующую таблицу с помощью INSERT

Вы можете захотеть настроить таблицу, в которую вставляете данные, например:

- тип данных столбца, настройку nullable или значения по умолчанию
- типы ключей и столбцы
- разбиение данных на partition и bucket

ПОДСКАЗКА

Создание наиболее эффективной структуры таблицы требует знания того, как данные будут использоваться и содержимого столбцов. Эта тема не охватывает проектирование таблиц. Для получения информации о проектировании таблиц см. Типы таблиц.

В этом примере мы создаём таблицу на основе знаний о том, как таблица будет запрашиваться, и данных в файле Parquet. Знание данных в файле Parquet можно получить, запросив файл напрямую в MinIO.

- Поскольку запрос набора данных в MinIO указывает, что столбец `Timestamp` содержит данные, соответствующие типу данных `datetime`, тип столбца указан в следующем DDL.
- Запрашивая данные в MinIO, вы можете обнаружить, что в наборе данных нет значений `NULL`, поэтому DDL не устанавливает какие-либо столбцы как nullable.
- На основе знаний об ожидаемых типах запросов sort key и столбец для bucket установлены на столбец `UserID`. Ваш случай использования может отличаться для этих данных, поэтому вы можете решить использовать `ItemID` в дополнение к `UserID` или вместо него для sort key.

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Создайте таблицу вручную (мы рекомендуем, чтобы таблица имела ту же схему, что и файл Parquet, который вы хотите загрузить из MinIO):

```
CREATE TABLE user_behavior_declared
(
    UserID int(11) NOT NULL,
    ItemID int(11) NOT NULL,
    CategoryID int(11) NOT NULL,
    BehaviorType varchar(65533) NOT NULL,
    Timestamp datetime NOT NULL
)
ENGINE = OLAP
DUPLICATE KEY(UserID)
DISTRIBUTED BY HASH(UserID)
PROPERTIES
(
    'replication_num' = '1'
);
```

Отобразите схему, чтобы вы могли сравнить её с выведенной схемой, созданной табличной функцией `FILES()`:

```
DESCRIBE user_behavior_declared;
```

```
+--------------+----------------+------+-------+---------+-------+
| Field        | Type           | Null | Key   | Default | Extra |
+--------------+----------------+------+-------+---------+-------+
| UserID       | int            | NO   | true  | NULL    |       |
| ItemID       | int            | NO   | false | NULL    |       |
| CategoryID   | int            | NO   | false | NULL    |       |
| BehaviorType | varchar(65533) | NO   | false | NULL    |       |
| Timestamp    | datetime       | NO   | false | NULL    |       |
+--------------+----------------+------+-------+---------+-------+
5 rows in set (0.00 sec)
```

ПОДСКАЗКА

Сравните схему, которую вы только что создали, со схемой, выведенной ранее с помощью табличной функции `FILES()`. Посмотрите на:

- типы данных
- nullable
- ключевые поля

Для лучшего контроля схемы целевой таблицы и лучшей производительности запросов мы рекомендуем указывать схему таблицы вручную в производственных средах. Использование типа данных `datetime` для поля timestamp более эффективно, чем использование `varchar`.

После создания таблицы вы можете загрузить её с помощью INSERT INTO SELECT FROM FILES():

ПОДСКАЗКА

Выделенный раздел команды включает настройки, которые вам может потребоваться изменить:

- Установите `endpoint` и `path` в соответствии с вашей системой MinIO.
- Если ваша система MinIO использует SSL, установите `enable_ssl` в `true`.
- Замените `AAA` и `BBB` на ваш access key и secret key MinIO.

```
INSERT INTO user_behavior_declared
SELECT * FROM FILES
(
    "aws.s3.endpoint" = "http://minio:9000",
    "path" = "s3://selena/user_behavior_ten_million_rows.parquet",
    "aws.s3.enable_ssl" = "false",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "format" = "parquet",
    "aws.s3.use_aws_sdk_default_behavior" = "false",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.enable_path_style_access" = "true"
);
```

После завершения загрузки вы можете запросить таблицу, чтобы убедиться, что данные были загружены в неё. Пример:

```
SELECT * from user_behavior_declared LIMIT 3;
```

Возвращается следующий результат запроса, указывающий, что данные были успешно загружены:

```
+--------+---------+------------+--------------+---------------------+
| UserID | ItemID  | CategoryID | BehaviorType | Timestamp           |
+--------+---------+------------+--------------+---------------------+
|     58 | 4309692 |    1165503 | pv           | 2017-11-25 14:06:52 |
|     58 |  181489 |    1165503 | pv           | 2017-11-25 14:07:22 |
|     58 | 3722956 |    1165503 | pv           | 2017-11-25 14:09:28 |
+--------+---------+------------+--------------+---------------------+
```

Проверка прогресса загрузки

Вы можете запросить прогресс заданий INSERT из представления `loads` в Information Schema Selena. Эта функция поддерживается начиная с версии v1.5.2. Пример:

```
SELECT * FROM information_schema.loads ORDER BY JOB_ID DESC;
```

Для получения информации о полях, предоставляемых в представлении `loads`, см. `loads`.

Если вы отправили несколько заданий загрузки, вы можете фильтровать по `LABEL`, связанному с заданием. Пример:

```
SELECT * FROM information_schema.loads WHERE LABEL = 'insert_e3b882f5-7eb3-11
*************************** 1. row ***************************
              JOB_ID: 10243
               LABEL: insert_e3b882f5-7eb3-11ee-ae77-00163e267b60
       DATABASE_NAME: mydatabase
               STATE: FINISHED
            PROGRESS: ETL:100%; LOAD:100%
                TYPE: INSERT
            PRIORITY: NORMAL
           SCAN_ROWS: 10000000
       FILTERED_ROWS: 0
     UNSELECTED_ROWS: 0
           SINK_ROWS: 10000000
            ETL_INFO:
           TASK_INFO: resource:N/A; timeout(s):300; max_filter_ratio:0.0
         CREATE_TIME: 2023-11-09 11:56:01
      ETL_START_TIME: 2023-11-09 11:56:01
     ETL_FINISH_TIME: 2023-11-09 11:56:01
     LOAD_START_TIME: 2023-11-09 11:56:01
    LOAD_FINISH_TIME: 2023-11-09 11:56:44
         JOB_DETAILS: {"All backends":{"e3b882f5-7eb3-11ee-ae77-00163e267b60"
[10142]},"FileNumber":0,"FileSize":0,"InternalTableLoadBytes":311710786,"Inte
backends":{"e3b882f5-7eb3-11ee-ae77-00163e267b60":[]}}
           ERROR_MSG: NULL
        TRACKING_URL: NULL
        TRACKING_SQL: NULL
REJECTED_RECORD_PATH: NULL
```

ПОДСКАЗКА

INSERT — это синхронная команда. Если задание INSERT всё ещё выполняется, вам нужно открыть другую сессию, чтобы проверить его статус выполнения.

Сравнение размеров таблиц на диске

Этот запрос сравнивает таблицу с выведенной схемой и таблицу с объявленной схемой. Поскольку выведенная схема имеет nullable столбцы и varchar для timestamp, длина данных больше:

```
SELECT TABLE_NAME,
       TABLE_ROWS,
       AVG_ROW_LENGTH,
       DATA_LENGTH
FROM information_schema.tables
WHERE TABLE_NAME like 'user_behavior%'\G
```

```
*************************** 1. row ***************************
    TABLE_NAME: user_behavior_declared
    TABLE_ROWS: 10000000
AVG_ROW_LENGTH: 10
   DATA_LENGTH: 102562516
*************************** 2. row ***************************
    TABLE_NAME: user_behavior_inferred
    TABLE_ROWS: 10000000
AVG_ROW_LENGTH: 17
   DATA_LENGTH: 176803880
2 rows in set (0.04 sec)
```

Использование Broker Load

Асинхронный процесс Broker Load обрабатывает подключение к MinIO, извлечение данных и сохранение данных в Selena.

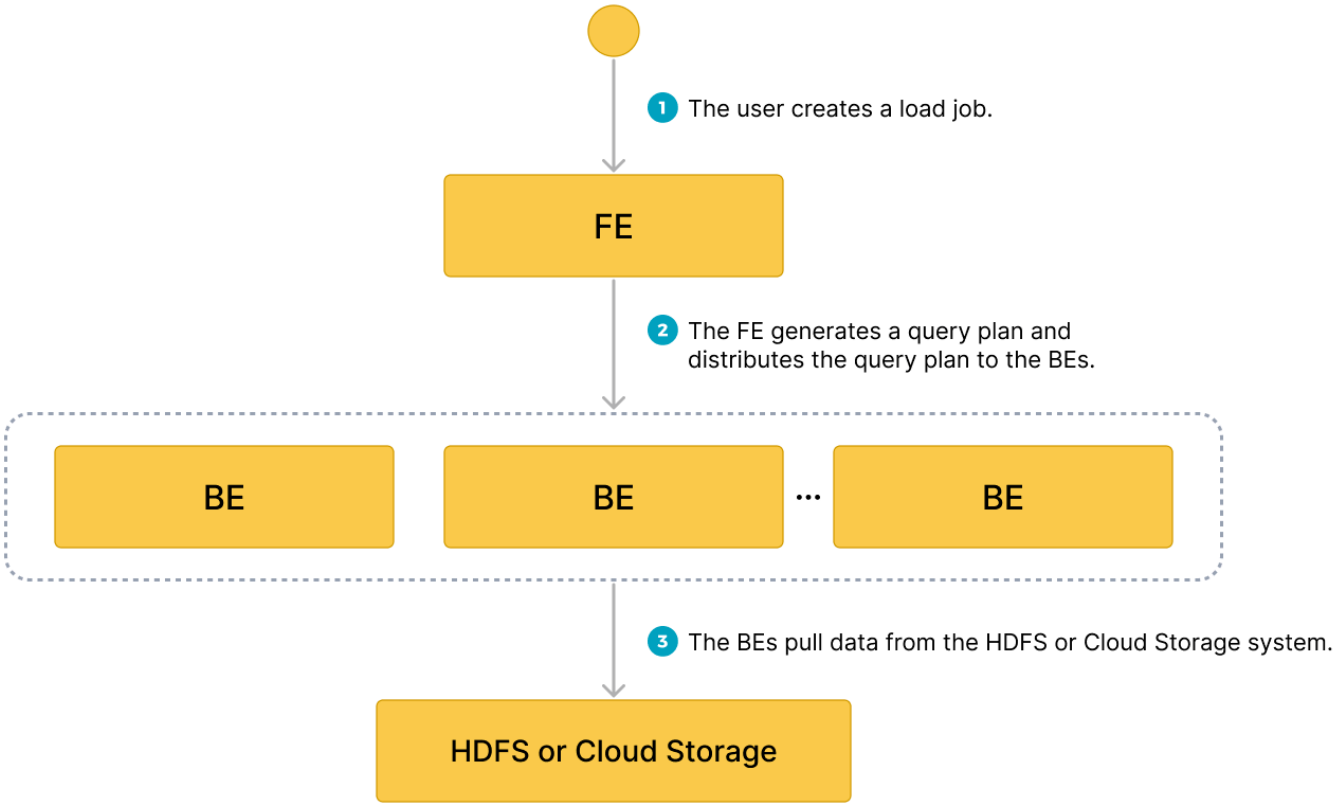
Этот метод поддерживает следующие форматы файлов:

- Parquet
- ORC
- CSV
- JSON (поддерживается начиная с версии v1.5.2)

Преимущества Broker Load

- Broker Load выполняется в фоновом режиме, и клиентам не нужно оставаться подключёнными для продолжения задания.
- Broker Load предпочтителен для длительных заданий, тайм-аут по умолчанию составляет 4 часа.
- В дополнение к форматам файлов Parquet и ORC, Broker Load поддерживает формат файлов CSV и формат файлов JSON (формат файлов JSON поддерживается начиная с версии v1.5.2).

Поток данных



1. Пользователь создаёт задание загрузки.
2. Frontend (FE) создаёт план запроса и распределяет план на backend-узлы (BE) или compute-узлы (CN).
3. BE или CN извлекают данные из источника и загружают данные в Selena.

Типичный пример

Создайте таблицу, запустите процесс загрузки, который извлекает пример набора данных, ранее загруженного в вашу систему MinIO.

Создание базы данных и таблицы

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Создайте таблицу вручную (мы рекомендуем, чтобы таблица имела ту же схему, что и файл Parquet, который вы хотите загрузить из MinIO):

```
CREATE TABLE user_behavior
(
    UserID int(11) NOT NULL,
    ItemID int(11) NOT NULL,
    CategoryID int(11) NOT NULL,
    BehaviorType varchar(65533) NOT NULL,
    Timestamp datetime NOT NULL
)
ENGINE = OLAP
DUPLICATE KEY(UserID)
DISTRIBUTED BY HASH(UserID)
PROPERTIES
(
    'replication_num' = '1'
);
```

Запуск Broker Load

Выполните следующую команду, чтобы запустить задание Broker Load, которое загружает данные из примера набора данных `user_behavior_ten_million_rows.parquet` в таблицу `user_behavior`:

ПОДСКАЗКА

Выделенный раздел команды включает настройки, которые вам может потребоваться изменить:

- Установите `endpoint` и `DATA INFILE` в соответствии с вашей системой MinIO.
- Если ваша система MinIO использует SSL, установите `enable_ssl` в `true`.
- Замените `AAA` и `BBB` на ваш access key и secret key MinIO.

```
LOAD LABEL UserBehavior
(
    DATA INFILE("s3://selena/user_behavior_ten_million_rows.parquet")
    INTO TABLE user_behavior
 )
 WITH BROKER
 (
    "aws.s3.endpoint" = "http://minio:9000",
    "aws.s3.enable_ssl" = "false",
    "aws.s3.access_key" = "AAAAAAAAAAAAAAAAAAAA",
    "aws.s3.secret_key" = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB",
    "aws.s3.use_aws_sdk_default_behavior" = "false",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.enable_path_style_access" = "true"
 )
PROPERTIES
(
    "timeout" = "72000"
);
```

Это задание имеет четыре основных раздела:

- `LABEL`: Строка, используемая при запросе состояния задания загрузки.
- `LOAD` declaration: URI источника, формат исходных данных и имя целевой таблицы.
- `BROKER`: Данные подключения к источнику.
- `PROPERTIES`: Значение тайм-аута и любые другие свойства, применяемые к заданию загрузки.

Подробный синтаксис и описание параметров см. в BROKER LOAD.

Проверка прогресса загрузки

Вы можете запросить прогресс заданий Broker Load из представления `loads` в Information Schema Selena. Эта функция поддерживается начиная с версии v1.5.2.

```
SELECT * FROM information_schema.loads;
```

Для получения информации о полях, предоставляемых в представлении `loads`, см. `loads`.

Если вы отправили несколько заданий загрузки, вы можете фильтровать по `LABEL`, связанному с заданием. Пример:

```
SELECT * FROM information_schema.loads
WHERE LABEL = 'UserBehavior'\G
```

```
*************************** 1. row ***************************
              JOB_ID: 10176
               LABEL: userbehavior
       DATABASE_NAME: mydatabase
               STATE: FINISHED
            PROGRESS: ETL:100%; LOAD:100%
                TYPE: BROKER
            PRIORITY: NORMAL
           SCAN_ROWS: 10000000
       FILTERED_ROWS: 0
     UNSELECTED_ROWS: 0
           SINK_ROWS: 10000000
            ETL_INFO:
           TASK_INFO: resource:N/A; timeout(s):72000; max_filter_ratio:0.0
         CREATE_TIME: 2023-12-19 23:02:41
      ETL_START_TIME: 2023-12-19 23:02:44
     ETL_FINISH_TIME: 2023-12-19 23:02:44
     LOAD_START_TIME: 2023-12-19 23:02:44
    LOAD_FINISH_TIME: 2023-12-19 23:02:46
         JOB_DETAILS: {"All backends":{"4aeec563-a91e-4c1e-b169-977b660950d1"
[10004]},"FileNumber":1,"FileSize":132251298,"InternalTableLoadBytes":3117107
backends":{"4aeec563-a91e-4c1e-b169-977b660950d1":[]}}
           ERROR_MSG: NULL
        TRACKING_URL: NULL
        TRACKING_SQL: NULL
REJECTED_RECORD_PATH: NULL
1 row in set (0.02 sec)
```

После того как вы убедитесь, что задание загрузки завершено, вы можете проверить подмножество целевой таблицы, чтобы увидеть, были ли данные успешно загружены. Пример:

```
SELECT * from user_behavior LIMIT 3;
```

Возвращается следующий результат запроса, указывающий, что данные были успешно загружены:

```
+--------+---------+------------+--------------+---------------------+
| UserID | ItemID  | CategoryID | BehaviorType | Timestamp           |
+--------+---------+------------+--------------+---------------------+
|    142 | 2869980 |    2939262 | pv           | 2017-11-25 03:43:22 |
|    142 | 2522236 |    1669167 | pv           | 2017-11-25 15:14:12 |
|    142 | 3031639 |    3607361 | pv           | 2017-11-25 15:19:25 |
+--------+---------+------------+--------------+---------------------+
```

3.3 Загрузка данных из локальной файловой системы

Selena предоставляет два метода загрузки данных из локальной файловой системы:

- Синхронная загрузка с использованием Stream Load
- Асинхронная загрузка с использованием Broker Load

Каждый из этих вариантов имеет свои преимущества:

- Stream Load поддерживает форматы CSV и JSON. Этот метод рекомендуется, если вы хотите загрузить данные из небольшого количества файлов, размер каждого из которых не превышает 10 ГБ.
- Broker Load поддерживает форматы Parquet, ORC, CSV и JSON (формат JSON поддерживается начиная с версии v1.5.2). Этот метод рекомендуется, если вы хотите загрузить данные из большого количества файлов, размер каждого из которых превышает 10 ГБ, или если файлы хранятся на сетевом хранилище (NAS). **Использование Broker Load для загрузки данных из локальной файловой системы поддерживается начиная с версии v1.5.2.**

Для данных в формате CSV обратите внимание на следующее:

- Вы можете использовать строку UTF-8, такую как запятая (,), табуляция или вертикальная черта (|), длиной не более 50 байт в качестве текстового разделителя.
- Значения NULL обозначаются с помощью `\N`. Например, если файл данных состоит из трёх столбцов, и запись из этого файла содержит данные в первом и третьем столбцах, но не содержит данных во втором столбце, необходимо использовать `\N` во втором столбце для обозначения значения NULL. Это означает, что запись должна

быть записана как `a,\N,b`, а не `a,,b`. Запись `a,,b` означает, что второй столбец содержит пустую строку.

Stream Load и Broker Load поддерживают трансформацию данных при загрузке, а также изменения данных с помощью операций UPSERT и DELETE во время загрузки. Для получения дополнительной информации см. Трансформация данных при загрузке и Изменение данных через загрузку.

3.3.1 Перед началом работы

Проверка привилегий

Вы можете загружать данные в таблицы Selena только как пользователь, имеющий привилегию INSERT на эти таблицы Selena. Если у вас нет привилегии INSERT, следуйте инструкциям в разделе GRANT, чтобы предоставить привилегию INSERT пользователю, которого вы используете для подключения к вашему cluster Selena. Синтаксис: `GRANT INSERT ON TABLE <table_name> IN DATABASE <database_name> TO { ROLE <role_name> | USER <user_identity>}`.

Проверка сетевой конфигурации

Убедитесь, что машина, на которой находятся загружаемые данные, имеет доступ к узлам FE и BE cluster Selena через порты `http_port` (по умолчанию: `8030`) и `be_http_port` (по умолчанию: `8040`) соответственно.

3.3.2 Загрузка из локальной файловой системы через Stream Load

Stream Load — это синхронный метод загрузки на основе HTTP PUT. После отправки задания загрузки Selena синхронно выполняет задание и возвращает результат после завершения. Вы можете определить успешность задания по его результату.

> **ПРИМЕЧАНИЕ**
>
> После загрузки данных в таблицу Selena с помощью Stream Load данные материализованных представлений, созданных на этой таблице, также обновляются.

Принцип работы

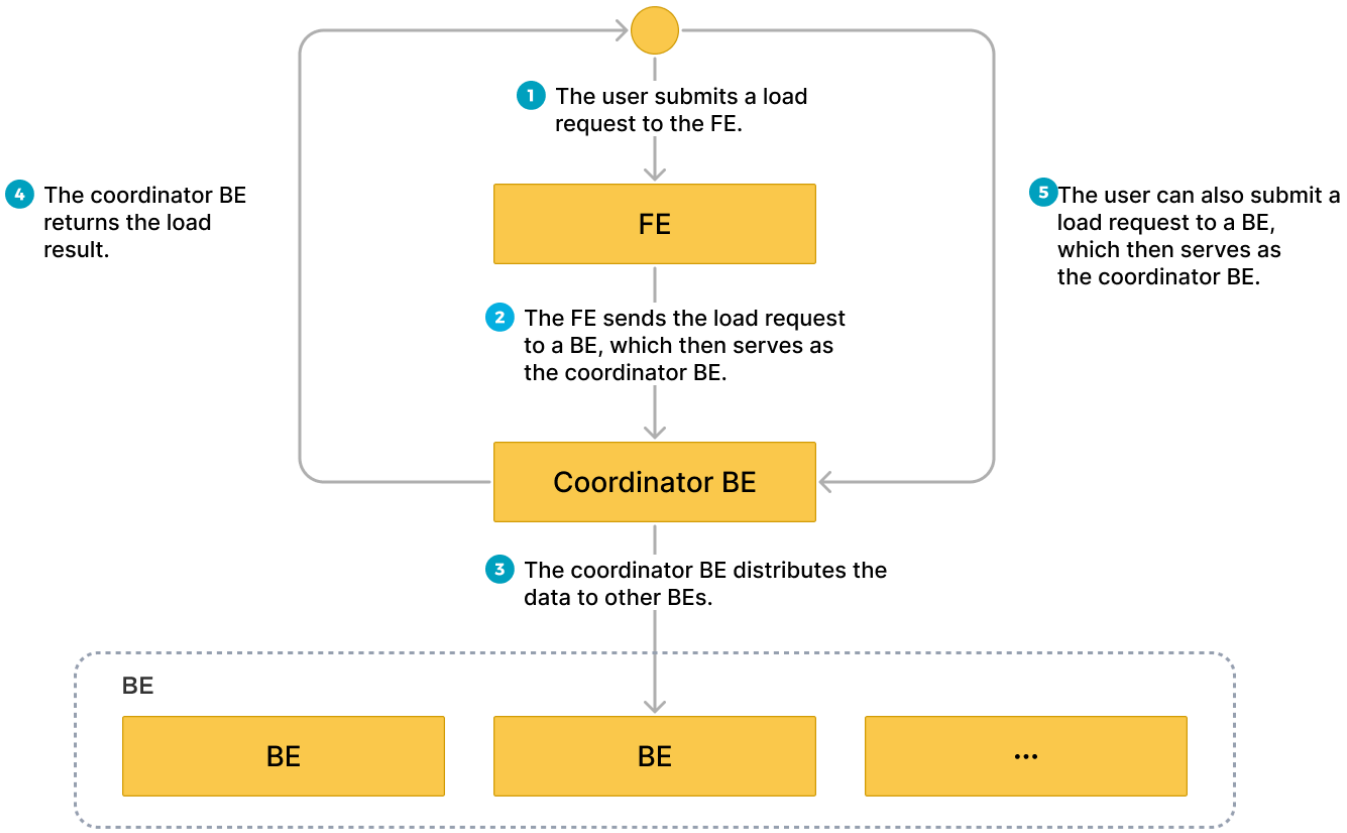
Вы можете отправить запрос загрузки с клиента на FE по протоколу HTTP, и FE затем перенаправит запрос загрузки на конкретный BE или CN с помощью HTTP redirect. Вы

также можете напрямую отправить запрос загрузки с клиента на выбранный BE или CN.

> **ПРИМЕЧАНИЕ**
>
> Если вы отправляете запросы загрузки на FE, FE использует механизм опроса для определения того, какой BE или CN будет служить координатором для получения и обработки запросов загрузки. Механизм опроса помогает достичь балансировки нагрузки в вашем cluster Selena. Поэтому мы рекомендуем отправлять запросы загрузки на FE.

BE или CN, получивший запрос загрузки, работает как Coordinator BE или CN для разделения данных на основе используемой схемы на части и распределения каждой части данных между другими задействованными BE или CN. После завершения загрузки Coordinator BE или CN возвращает результат задания загрузки вашему клиенту. Обратите внимание, что если вы остановите Coordinator BE или CN во время загрузки, задание загрузки завершится неудачей.

На следующем рисунке показан рабочий процесс задания Stream Load.



Ограничения

Stream Load не поддерживает загрузку данных из CSV-файла, содержащего столбец в формате JSON.

Типичный пример

В этом разделе в качестве примера используется curl для описания загрузки данных из файла CSV или JSON из локальной файловой системы в Selena. Подробный синтаксис и описание параметров см. в STREAM LOAD.

Обратите внимание, что в Selena некоторые литералы используются как зарезервированные ключевые слова языка SQL. Не используйте эти ключевые слова напрямую в SQL-операторах. Если вы хотите использовать такое ключевое слово в SQL-операторе, заключите его в обратные кавычки (`). См. Ключевые слова.

Загрузка данных CSV

Подготовка наборов данных

В локальной файловой системе создайте CSV-файл с именем `example1.csv`. Файл состоит из трёх столбцов, которые представляют идентификатор пользователя, имя пользователя и оценку пользователя соответственно.

```
1,Lily,23
2,Rose,23
3,Alice,24
4,Julia,25
```

Создание базы данных и таблицы

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Создайте таблицу с первичным ключом с именем `table1`. Таблица состоит из трёх столбцов: `id`, `name` и `score`, где `id` является первичным ключом.

```
CREATE TABLE `table1`
(
    `id` int(11) NOT NULL COMMENT "user ID",
    `name` varchar(65533) NULL COMMENT "user name",
    `score` int(11) NOT NULL COMMENT "user score"
)
ENGINE=OLAP
PRIMARY KEY(`id`)
DISTRIBUTED BY HASH(`id`);
```

> **ПРИМЕЧАНИЕ**
>
> Начиная с версии v1.5.2, Selena может автоматически устанавливать количество бакетов (BUCKETS) при создании таблицы или добавлении партиции. Вам больше не нужно вручную устанавливать количество бакетов. Подробную информацию см. в установка количества бакетов.

Запуск Stream Load

Выполните следующую команду для загрузки данных из `example1.csv` в `table1`:

```
curl --location-trusted -u <username>:<password> -H "label:123" \
    -H "Expect:100-continue" \
    -H "column_separator:," \
    -H "columns: id, name, score" \
    -T example1.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/mydatabase/table1/_stream_load
```

> **ПРИМЕЧАНИЕ**
>
> - Если вы используете учётную запись без установленного пароля, вам нужно ввести только `<username>:`.
> - Вы можете использовать SHOW FRONTENDS для просмотра IP-адреса и HTTP-порта узла FE.

`example1.csv` состоит из трёх столбцов, разделённых запятыми (,), которые могут быть последовательно сопоставлены со столбцами `id`, `name` и `score` таблицы `table1`. Поэтому необходимо использовать параметр `column_separator` для указания запятой (,) в качестве разделителя столбцов. Также необходимо использовать параметр `columns` для

временного именования трёх столбцов `example1.csv` как `id`, `name` и `score`, которые последовательно сопоставляются с тремя столбцами `table1`.

После завершения загрузки вы можете запросить `table1`, чтобы убедиться в успешности загрузки:

```
SELECT * FROM table1;
+------+-------+-------+
| id   | name  | score |
+------+-------+-------+
|    1 | Lily  |    23 |
|    2 | Rose  |    23 |
|    3 | Alice |    24 |
|    4 | Julia |    25 |
+------+-------+-------+
4 rows in set (0.00 sec)
```

Загрузка данных JSON

Начиная с версии v1.5.2, Stream Load поддерживает сжатие данных JSON при передаче, уменьшая накладные расходы на пропускную способность сети. Пользователи могут указать различные алгоритмы сжатия с помощью параметров `compression` и `Content-Encoding`. Поддерживаемые алгоритмы сжатия включают GZIP, BZIP2, LZ4_FRAME и ZSTD. Синтаксис см. в STREAM LOAD.

Подготовка наборов данных

В локальной файловой системе создайте JSON-файл с именем `example2.json`. Файл состоит из двух столбцов, которые представляют идентификатор города и название города соответственно.

```
{"name": "Beijing", "code": 2}
```

Создание базы данных и таблицы

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Создайте таблицу с первичным ключом с именем `table2`. Таблица состоит из двух столбцов: `id` и `city`, где `id` является первичным ключом.

```
CREATE TABLE `table2`
(
    `id` int(11) NOT NULL COMMENT "city ID",
    `city` varchar(65533) NULL COMMENT "city name"
)
ENGINE=OLAP
PRIMARY KEY(`id`)
DISTRIBUTED BY HASH(`id`);
```

> **ПРИМЕЧАНИЕ**
>
> Начиная с версии v1.5.2, Selena может автоматически устанавливать количество бакетов (BUCKETS) при создании таблицы или добавлении партиции. Вам больше не нужно вручную устанавливать количество бакетов. Подробную информацию см. в установка количества бакетов.

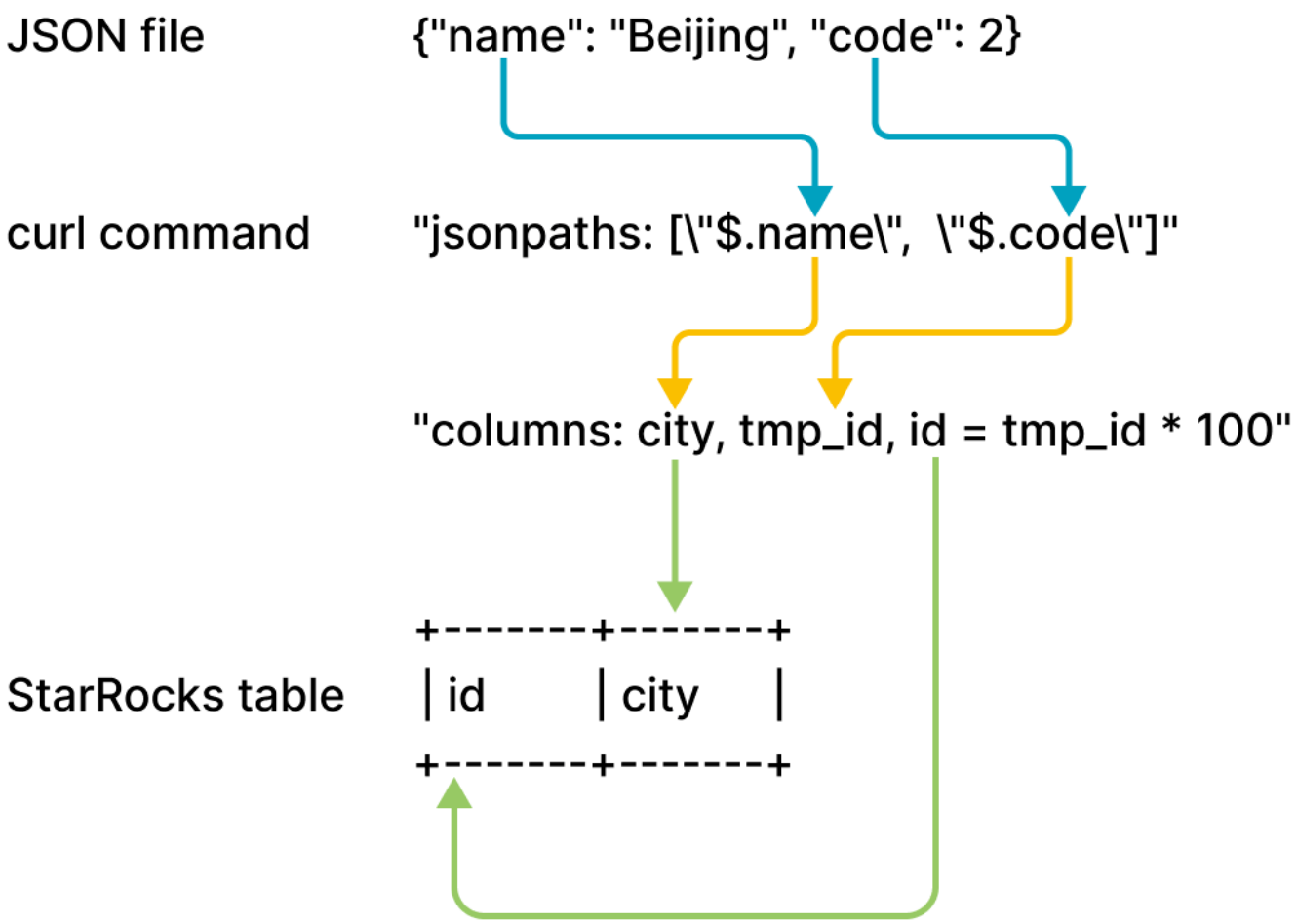
Запуск Stream Load

Выполните следующую команду для загрузки данных из `example2.json` в `table2`:

```
curl -v --location-trusted -u <username>:<password> -H "strict_mode: true"
\
    -H "Expect:100-continue" \
    -H "format: json" -H "jsonpaths: [\"$.name\", \"$.code\"]" \
    -H "columns: city,tmp_id, id = tmp_id * 100" \
    -T example2.json -XPUT \
    http://<fe_host>:<fe_http_port>/api/mydatabase/table2/_stream_load
```

> **ПРИМЕЧАНИЕ**
>
> - Если вы используете учётную запись без установленного пароля, вам нужно ввести только `<username>:`.
> - Вы можете использовать SHOW FRONTENDS для просмотра IP-адреса и HTTP-порта узла FE.

`example2.json` состоит из двух ключей, `name` и `code`, которые сопоставляются со столбцами `id` и `city` таблицы `table2`, как показано на следующем рисунке.



Сопоставления, показанные на предыдущем рисунке, описаны следующим образом:

- Selena извлекает ключи `name` и `code` из `example2.json` и сопоставляет их с полями `name` и `code`, объявленными в параметре `jsonpaths`.
- Selena извлекает поля `name` и `code`, объявленные в параметре `jsonpaths`, и **последовательно сопоставляет** их с полями `city` и `tmp_id`, объявленными в параметре `columns`.
- Selena извлекает поля `city` и `tmp_id`, объявленные в параметре `columns`, и **сопоставляет их по имени** со столбцами `city` и `id` таблицы `table2`.

> **ПРИМЕЧАНИЕ**
>
> В предыдущем примере значение `code` в `example2.json` умножается на 100 перед загрузкой в столбец `id` таблицы `table2`.

Подробное описание сопоставлений между `jsonpaths`, `columns` и столбцами таблицы Selena см. в разделе "Сопоставление столбцов" в STREAM LOAD.

После завершения загрузки вы можете запросить `table2`, чтобы убедиться в успешности загрузки:

```
SELECT * FROM table2;
+------+--------+
| id   | city   |
+------+--------+
| 200  | Beijing|
+------+--------+
4 rows in set (0.01 sec)
```

Объединение запросов Stream Load

> *Beta* **BETA FEATURE**
>
> Рекомендации по использованию бета-функций

Начиная с версии v1.5.2, система поддерживает объединение нескольких запросов Stream Load.

> **WARNING**
>
> Обратите внимание, что оптимизация Merge Commit подходит для сценариев с **параллельными** заданиями Stream Load на одной таблице. Она не рекомендуется, если параллелизм равен единице. Также хорошо подумайте перед установкой `merge_commit_async` в `false` и `merge_commit_interval_ms` в большое значение, так как это может вызвать снижение производительности загрузки.

Merge Commit — это оптимизация для Stream Load, разработанная для сценариев загрузки в реальном времени с высокой параллельностью и небольшими пакетами (от КБ

до десятков МБ). В более ранних версиях каждый запрос Stream Load создавал транзакцию и версию данных, что приводило к следующим проблемам в сценариях загрузки с высокой параллельностью:

- Избыточные версии данных влияют на производительность запросов, а ограничение количества версий может вызвать ошибки `too many versions`.
- Слияние версий данных через Compaction увеличивает потребление ресурсов.
- Создаются мелкие файлы, увеличивая IOPS и задержку I/O. В cluster с разделяемыми данными это также увеличивает затраты на облачное объектное хранилище.
- Узел Leader FE как менеджер транзакций может стать единой точкой узкого места.

Merge Commit смягчает эти проблемы путём объединения нескольких параллельных запросов Stream Load в течение временного окна в одну транзакцию. Это уменьшает количество транзакций и версий, генерируемых запросами с высокой параллельностью, тем самым улучшая производительность загрузки.

Merge Commit поддерживает как синхронный, так и асинхронный режимы. Каждый режим имеет свои преимущества и недостатки. Вы можете выбрать в зависимости от ваших сценариев использования.

- **Синхронный режим**

  Сервер возвращает ответ только после фиксации объединённой транзакции, гарантируя успешность и видимость загрузки.

- **Асинхронный режим**

  Сервер возвращает ответ сразу после получения данных. Этот режим не гарантирует успешность загрузки.

Режим	Преимущества	Недостатки
Синхронный	• Гарантирует сохранение и видимость данных при возврате запроса. • Гарантирует, что несколько последовательных запросов загрузки от одного клиента выполняются по порядку.	Каждый запрос загрузки от клиента блокируется до тех пор, пока сервер не закроет окно слияния. Это может снизить способность обработки данных одного клиента, если окно слишком большое.
Асинхро	Позволяет одному клиенту отправлять последующие запросы загрузки без ожидания	• Не гарантирует сохранение или видимость данных при возврате. Клиент должен позже

Режим	Преимущества	Недостатки
нный	закрытия окна слияния сервером, улучшая пропускную способность загрузки.	проверить статус транзакции. • Не гарантирует, что несколько последовательных запросов загрузки от одного клиента выполняются по порядку.

Запуск Stream Load

- Выполните следующую команду для запуска задания Stream Load с включённым Merge Commit в синхронном режиме, установив окно слияния на `5000` миллисекунд и степень параллелизма на `2`:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "column_separator:," \
    -H "columns: id, name, score" \
    -H "enable_merge_commit:true" \
    -H "merge_commit_interval_ms:5000" \
    -H "merge_commit_parallel:2" \
    -T example1.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/mydatabase/table1/_stream_load
```

- Выполните следующую команду для запуска задания Stream Load с включённым Merge Commit в асинхронном режиме, установив окно слияния на `60000` миллисекунд и степень параллелизма на `2`:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "column_separator:," \
    -H "columns: id, name, score" \
    -H "enable_merge_commit:true" \
    -H "merge_commit_async:true" \
    -H "merge_commit_interval_ms:60000" \
    -H "merge_commit_parallel:2" \
    -T example1.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/mydatabase/table1/_stream_load
```

ПРИМЕЧАНИЕ

- Merge Commit поддерживает только объединение **однородных** запросов загрузки в одну базу данных и таблицу. "Однородные" означает, что параметры Stream Load идентичны, включая: общие параметры, параметры формата JSON, параметры формата CSV, `opt_properties` и параметры Merge Commit.
- При загрузке данных в формате CSV вы должны убедиться, что каждая строка заканчивается разделителем строк. `skip_header` не поддерживается.
- Сервер автоматически генерирует метки для транзакций. Указанные метки будут игнорироваться.
- Merge Commit объединяет несколько запросов загрузки в одну транзакцию. Если один запрос содержит проблемы с качеством данных, все запросы в транзакции завершатся неудачей.

Проверка прогресса Stream Load

После завершения задания загрузки Selena возвращает результат задания в формате JSON. Для получения дополнительной информации см. раздел "Возвращаемое значение" в STREAM LOAD.

Stream Load не позволяет запрашивать результат задания загрузки с помощью оператора SHOW LOAD.

Отмена задания Stream Load

Stream Load не позволяет отменить задание загрузки. Если задание загрузки истекает по таймауту или возникают ошибки, Selena автоматически отменяет задание.

Конфигурация параметров

В этом разделе описаны некоторые системные параметры, которые необходимо настроить при выборе метода загрузки Stream Load. Эти конфигурации параметров влияют на все задания Stream Load.

- `streaming_load_max_mb`: максимальный размер каждого файла данных, который вы хотите загрузить. Максимальный размер по умолчанию — 10 ГБ. Для получения дополнительной информации см. Настройка динамических параметров BE или CN.

Мы рекомендуем не загружать более 10 ГБ данных за раз. Если размер файла данных превышает 10 ГБ, мы рекомендуем разделить файл данных на небольшие файлы размером менее 10 ГБ каждый, а затем загружать эти файлы по одному. Если вы не можете разделить файл данных размером более 10 ГБ, вы можете увеличить значение этого параметра в зависимости от размера файла.

После увеличения значения этого параметра новое значение вступит в силу только после перезапуска BE или CN вашего cluster Selena. Кроме того, производительность системы может ухудшиться, а затраты на повторные попытки в случае сбоя загрузки также увеличатся.

> **ПРИМЕЧАНИЕ**
>
> При загрузке данных из JSON-файла обратите внимание на следующее:
>
> - Размер каждого объекта JSON в файле не может превышать 4 ГБ. Если какой-либо объект JSON в файле превышает 4 ГБ, Selena выдаёт ошибку "This parser can't support a document that big."
> - По умолчанию тело JSON в HTTP-запросе не может превышать 100 МБ. Если тело JSON превышает 100 МБ, Selena выдаёт ошибку "The size of this batch exceed the max size [104857600] of json type data data [8617627793]. Set ignore_json_size to skip check, although it may lead huge memory consuming." Чтобы предотвратить эту ошибку, вы можете добавить `"ignore_json_size:true"` в заголовок HTTP-запроса, чтобы игнорировать проверку размера тела JSON.

- `stream_load_default_timeout_second`: тайм-аут для каждого задания загрузки. Тайм-аут по умолчанию — 600 секунд. Для получения дополнительной информации см. Настройка динамических параметров FE.

  Если многие из созданных вами заданий загрузки истекают по таймауту, вы можете увеличить значение этого параметра на основе результата расчёта, который вы получите по следующей формуле:

  Тайм-аут каждого задания загрузки > Объём загружаемых данных / Средняя скорость загрузки

Например, если размер файла данных, который вы хотите загрузить, составляет 10 ГБ, а средняя скорость загрузки вашего cluster Selena составляет 100 МБ/с, установите тайм-аут более 100 секунд.

> **ПРИМЕЧАНИЕ**
>
> **Средняя скорость загрузки** в предыдущей формуле — это средняя скорость загрузки вашего cluster Selena. Она зависит от I/O диска и количества BE или CN в вашем cluster Selena.

Stream Load также предоставляет параметр `timeout`, который позволяет указать тайм-аут для отдельного задания загрузки. Для получения дополнительной информации см. STREAM LOAD.

Примечания по использованию

Если поле отсутствует для записи в файле данных, который вы хотите загрузить, и столбец, на который сопоставляется поле в вашей таблице Selena, определён как `NOT NULL`, Selena автоматически заполняет значение `NULL` в сопоставляемом столбце вашей таблицы Selena во время загрузки записи. Вы также можете использовать функцию `ifnull()` для указания значения по умолчанию, которое вы хотите заполнить.

Например, если поле, представляющее идентификатор города, в предыдущем файле `example2.json` отсутствует, и вы хотите заполнить значение `x` в сопоставляемом столбце `table2`, вы можете указать `"columns: city, tmp_id, id = ifnull(tmp_id, 'x')"`.

3.3.3 Загрузка из локальной файловой системы через Broker Load

В дополнение к Stream Load вы также можете использовать Broker Load для загрузки данных из локальной файловой системы. Эта функция поддерживается начиная с версии v1.5.2.

Broker Load — это асинхронный метод загрузки. После отправки задания загрузки Selena асинхронно выполняет задание и не возвращает сразу результат задания. Вам нужно запросить результат задания вручную. См. Проверка прогресса Broker Load.

Ограничения

- В настоящее время Broker Load поддерживает загрузку из локальной файловой системы только через один брокер версии v1.5.2 или выше.
- Запросы с высокой параллельностью к одному брокеру могут вызвать такие проблемы, как тайм-аут и OOM. Для смягчения влияния вы можете использовать переменную `pipeline_dop` (см. Системная переменная) для установки параллелизма запросов для Broker Load. Для запросов к одному брокеру мы рекомендуем установить `pipeline_dop` на значение менее `16`.

Типичный пример

Broker Load поддерживает загрузку из одного файла данных в одну таблицу, загрузку из нескольких файлов данных в одну таблицу и загрузку из нескольких файлов данных в несколько таблиц. В этом разделе в качестве примера используется загрузка из нескольких файлов данных в одну таблицу.

Обратите внимание, что в Selena некоторые литералы используются как зарезервированные ключевые слова языка SQL. Не используйте эти ключевые слова напрямую в SQL-операторах. Если вы хотите использовать такое ключевое слово в SQL-операторе, заключите его в обратные кавычки (`). См. Ключевые слова.

Подготовка наборов данных

В качестве примера используется формат CSV. Войдите в локальную файловую систему и создайте два CSV-файла, `file1.csv` и `file2.csv`, в определённом месте хранения (например, `/home/disk1/business/`). Оба файла состоят из трёх столбцов, которые представляют идентификатор пользователя, имя пользователя и оценку пользователя соответственно.

- `file1.csv`

```
1,Lily,21
2,Rose,22
3,Alice,23
4,Julia,24
```

- `file2.csv`

```
5,Tony,25
6,Adam,26
7,Allen,27
8,Jacky,28
```

Создание базы данных и таблицы

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Создайте таблицу с первичным ключом с именем `mytable`. Таблица состоит из трёх столбцов: `id`, `name` и `score`, где `id` является первичным ключом.

```
CREATE TABLE `mytable`
(
    `id` int(11) NOT NULL COMMENT "User ID",
    `name` varchar(65533) NULL DEFAULT "" COMMENT "User name",
    `score` int(11) NOT NULL DEFAULT "0" COMMENT "User score"
)
ENGINE=OLAP
PRIMARY KEY(`id`)
DISTRIBUTED BY HASH(`id`)
PROPERTIES("replication_num"="1");
```

Запуск Broker Load

Выполните следующую команду для запуска задания Broker Load, которое загружает данные из всех файлов данных (`file1.csv` и `file2.csv`), хранящихся по пути `/home/disk1/business/` вашей локальной файловой системы, в таблицу Selena `mytable`:

```
LOAD LABEL mydatabase.label_local
(
    DATA INFILE("file:///home/disk1/business/csv/*")
    INTO TABLE mytable
    COLUMNS TERMINATED BY ","
    (id, name, score)
)
WITH BROKER "sole_broker"
PROPERTIES
(
    "timeout" = "3600"
);
```

Это задание содержит четыре основных раздела:

- `LABEL`: Строка, используемая при запросе состояния задания загрузки.
- `LOAD` объявление: Исходный URI, формат исходных данных и имя целевой таблицы.
- `PROPERTIES`: Значение тайм-аута и любые другие свойства, применяемые к заданию загрузки.

Подробный синтаксис и описание параметров см. в BROKER LOAD.

Проверка прогресса Broker Load

В версии v1.5.2 и более ранних используйте оператор SHOW LOAD или команду curl для просмотра прогресса заданий Broker Load.

В версии v1.5.2 и более поздних вы можете просмотреть прогресс заданий Broker Load из представления `information_schema.loads`:

```
SELECT * FROM information_schema.loads;
```

Если вы отправили несколько заданий загрузки, вы можете отфильтровать по `LABEL`, связанному с заданием. Пример:

```
SELECT * FROM information_schema.loads WHERE LABEL = 'label_local';
```

После подтверждения завершения задания загрузки вы можете запросить таблицу, чтобы проверить, были ли данные успешно загружены. Пример:

```
SELECT * FROM mytable;
+------+-------+-------+
| id   | name  | score |
+------+-------+-------+
|    3 | Alice |    23 |
|    5 | Tony  |    25 |
|    6 | Adam  |    26 |
|    1 | Lily  |    21 |
|    2 | Rose  |    22 |
|    4 | Julia |    24 |
|    7 | Allen |    27 |
|    8 | Jacky |    28 |
+------+-------+-------+
8 rows in set (0.07 sec)
```

Отмена задания Broker Load

Когда задание загрузки не находится на стадии **CANCELLED** или **FINISHED**, вы можете использовать оператор CANCEL LOAD для отмены задания.

Например, вы можете выполнить следующий оператор для отмены задания загрузки с меткой `label_local` в базе данных `mydatabase`:

```
CANCEL LOAD
FROM mydatabase
WHERE LABEL = "label_local";
```

3.3.4 Загрузка из NAS через Broker Load

Существует два способа загрузки данных из NAS с использованием Broker Load:

- Рассматривать NAS как локальную файловую систему и запускать задание загрузки с брокером. См. предыдущий раздел "Загрузка из локальной системы через Broker Load".
- (Рекомендуется) Рассматривать NAS как облачную систему хранения и запускать задание загрузки без брокера.

В этом разделе описан второй способ. Подробные операции следующие:

1. Смонтируйте устройство NAS по одному и тому же пути на всех узлах BE или CN и узлах FE вашего cluster Selena. Таким образом, все BE или CN могут получить доступ к устройству NAS так же, как они получают доступ к своим локально сохранённым файлам.

2. Используйте Broker Load для загрузки данных с устройства NAS в целевую таблицу Selena. Пример:

```
LOAD LABEL test_db.label_nas
(
    DATA INFILE("file:///home/disk1/sr/*")
    INTO TABLE mytable
    COLUMNS TERMINATED BY ","
)
WITH BROKER
PROPERTIES
(
    "timeout" = "3600"
);
```

   Это задание содержит четыре основных раздела:

   - `LABEL`: Строка, используемая при запросе состояния задания загрузки.
   - `LOAD` объявление: Исходный URI, формат исходных данных и имя целевой таблицы. Обратите внимание, что `DATA INFILE` в объявлении используется для указания пути к точке монтирования устройства NAS, как показано в приведённом выше примере, где `file:///` является префиксом, а `/home/disk1/sr` — путём к папке точки монтирования.
   - `BROKER`: Вам не нужно указывать имя брокера.
   - `PROPERTIES`: Значение тайм-аута и любые другие свойства, применяемые к заданию загрузки.

   Подробный синтаксис и описание параметров см. в BROKER LOAD.

После отправки задания вы можете просмотреть прогресс загрузки или отменить задание по мере необходимости. Подробные операции см. в разделах "Проверка прогресса Broker Load" и "Отмена задания Broker Load" в этой теме.

3.4 Загрузка данных из HDFS

Selena предоставляет следующие варианты загрузки данных из HDFS:

- Синхронная загрузка с использованием INSERT+`FILES()`
- Асинхронная загрузка с использованием Broker Load
- Непрерывная асинхронная загрузка с использованием Pipe

Каждый из этих вариантов имеет свои преимущества, которые подробно описаны в следующих разделах.

В большинстве случаев мы рекомендуем использовать метод INSERT+`FILES()`, который намного проще в использовании.

Однако метод INSERT+`FILES()` в настоящее время поддерживает только форматы файлов Parquet, ORC и CSV. Поэтому, если вам нужно загружать данные других форматов файлов, таких как JSON, или выполнять изменения данных, такие как DELETE, во время загрузки данных, вы можете использовать Broker Load.

Если вам нужно загрузить большое количество файлов данных со значительным общим объёмом данных (например, более 100 ГБ или даже 1 ТБ), мы рекомендуем использовать метод Pipe. Pipe может разбивать файлы по их количеству или размеру, разделяя задание загрузки на более мелкие последовательные задачи. Такой подход гарантирует, что ошибки в одном файле не повлияют на всё задание загрузки, и минимизирует необходимость повторных попыток из-за ошибок в данных.

3.4.1 Перед началом работы

Подготовка исходных данных

Убедитесь, что исходные данные, которые вы хотите загрузить в Selena, правильно хранятся в вашем cluster HDFS. В этой теме предполагается, что вы хотите загрузить `/user/amber/user_behavior_ten_million_rows.parquet` из HDFS в Selena.

Проверка привилегий

Вы можете загружать данные в таблицы Selena только как пользователь, имеющий привилегию INSERT на эти таблицы Selena. Если у вас нет привилегии INSERT, следуйте инструкциям в разделе GRANT, чтобы предоставить привилегию INSERT пользователю, которого вы используете для подключения к вашему cluster Selena. Синтаксис: `GRANT`

`INSERT ON TABLE <table_name> IN DATABASE <database_name> TO { ROLE <role_name> | USER <user_identity>}`.

Сбор данных аутентификации

Вы можете использовать простой метод аутентификации для установления соединений с вашим cluster HDFS. Чтобы использовать простую аутентификацию, вам нужно собрать имя пользователя и пароль учётной записи, которую вы можете использовать для доступа к NameNode cluster HDFS.

3.4.2 Использование INSERT+FILES()

Этот метод доступен начиная с версии v1.5.2 и в настоящее время поддерживает только форматы файлов Parquet, ORC и CSV (начиная с версии v1.5.2).

Преимущества INSERT+FILES()

`FILES()` может читать файл, хранящийся в облачном хранилище, на основе указанных вами свойств, связанных с путём, выводить схему таблицы данных в файле, а затем возвращать данные из файла в виде строк данных.

С помощью `FILES()` вы можете:

- Запрашивать данные напрямую из HDFS с помощью SELECT.
- Создавать и загружать таблицу с помощью CREATE TABLE AS SELECT (CTAS).
- Загружать данные в существующую таблицу с помощью INSERT.

Типичные примеры

Прямой запрос из HDFS с помощью SELECT

Прямой запрос из HDFS с помощью SELECT+`FILES()` позволяет получить хороший предварительный просмотр содержимого набора данных перед созданием таблицы. Например:

- Получить предварительный просмотр набора данных без сохранения данных.
- Запросить минимальные и максимальные значения и решить, какие типы данных использовать.
- Проверить наличие значений `NULL`.

Следующий пример запрашивает файл данных `/user/amber/user_behavior_ten_million_rows.parquet`, хранящийся в cluster HDFS:

```
SELECT * FROM FILES
(
    "path" = "hdfs://<hdfs_ip>:
<hdfs_port>/user/amber/user_behavior_ten_million_rows.parquet",
    "format" = "parquet",
    "hadoop.security.authentication" = "simple",
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
)
LIMIT 3;
```

Система возвращает следующий результат запроса:

```
+--------+---------+------------+--------------+---------------------+
| UserID | ItemID  | CategoryID | BehaviorType | Timestamp           |
+--------+---------+------------+--------------+---------------------+
| 543711 |  829192 |    2355072 | pv           | 2017-11-27 08:22:37 |
| 543711 | 2056618 |    3645362 | pv           | 2017-11-27 10:16:46 |
| 543711 | 1165492 |    3645362 | pv           | 2017-11-27 10:17:00 |
+--------+---------+------------+--------------+---------------------+
```

> **ПРИМЕЧАНИЕ**
>
> Обратите внимание, что имена столбцов, возвращённые выше, предоставлены файлом Parquet.

Создание и загрузка таблицы с помощью CTAS

Это продолжение предыдущего примера. Предыдущий запрос обёрнут в CREATE TABLE AS SELECT (CTAS) для автоматизации создания таблицы с использованием вывода схемы. Это означает, что Selena выведет схему таблицы, создаст нужную вам таблицу, а затем загрузит данные в таблицу. Имена и типы столбцов не требуются для создания таблицы при использовании табличной функции `FILES()` с файлами Parquet, поскольку формат Parquet включает имена столбцов.

> **ПРИМЕЧАНИЕ**

> Синтаксис CREATE TABLE при использовании вывода схемы не позволяет устанавливать количество replica, поэтому установите его перед созданием таблицы. Пример ниже для системы с тремя replica:
>
> ```
> ADMIN SET FRONTEND CONFIG ('default_replication_num' = "3");
> ```

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Используйте CTAS для создания таблицы и загрузки данных из файла данных `/user/amber/user_behavior_ten_million_rows.parquet` в таблицу:

```
CREATE TABLE user_behavior_inferred AS
SELECT * FROM FILES
(
    "path" = "hdfs://<hdfs_ip>:
<hdfs_port>/user/amber/user_behavior_ten_million_rows.parquet",
    "format" = "parquet",
    "hadoop.security.authentication" = "simple",
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

После создания таблицы вы можете просмотреть её схему с помощью DESCRIBE:

```
DESCRIBE user_behavior_inferred;
```

Система возвращает следующий результат запроса:

```
+--------------+-----------+------+-------+---------+-------+
| Field        | Type      | Null | Key   | Default | Extra |
+--------------+-----------+------+-------+---------+-------+
| UserID       | bigint    | YES  | true  | NULL    |       |
| ItemID       | bigint    | YES  | true  | NULL    |       |
| CategoryID   | bigint    | YES  | true  | NULL    |       |
| BehaviorType | varbinary | YES  | false | NULL    |       |
| Timestamp    | varbinary | YES  | false | NULL    |       |
+--------------+-----------+------+-------+---------+-------+
```

Запросите таблицу, чтобы убедиться, что данные были загружены в неё. Пример:

```
SELECT * from user_behavior_inferred LIMIT 3;
```

Возвращается следующий результат запроса, указывающий, что данные были успешно загружены:

```
+--------+--------+------------+--------------+---------------------+
| UserID | ItemID | CategoryID | BehaviorType | Timestamp           |
+--------+--------+------------+--------------+---------------------+
|     84 |  56257 |    1879194 | pv           | 2017-11-26 05:56:23 |
|     84 | 108021 |    2982027 | pv           | 2017-12-02 05:43:00 |
|     84 | 390657 |    1879194 | pv           | 2017-11-28 11:20:30 |
+--------+--------+------------+--------------+---------------------+
```

Загрузка в существующую таблицу с помощью INSERT

Вы можете захотеть настроить таблицу, в которую вставляете данные, например:

- тип данных столбца, настройку nullable или значения по умолчанию
- типы ключей и столбцы
- разбиение данных на partition и bucket

> **ПРИМЕЧАНИЕ**
>
> Создание наиболее эффективной структуры таблицы требует знания того, как данные будут использоваться и содержимого столбцов. Эта тема не охватывает проектирование таблиц. Для получения информации о проектировании таблиц см. Типы таблиц.

В этом примере мы создаём таблицу на основе знаний о том, как таблица будет запрашиваться, и данных в файле Parquet. Знание данных в файле Parquet можно получить, запросив файл напрямую в HDFS.

- Поскольку запрос набора данных в HDFS указывает, что столбец `Timestamp` содержит данные, соответствующие типу данных VARBINARY, тип столбца указан в следующем DDL.
- Запрашивая данные в HDFS, вы можете обнаружить, что в наборе данных нет значений `NULL`, поэтому DDL не устанавливает какие-либо столбцы как nullable.
- На основе знаний об ожидаемых типах запросов sort key и столбец для bucket установлены на столбец `UserID`. Ваш случай использования может отличаться для этих данных, поэтому вы можете решить использовать `ItemID` в дополнение к `UserID` или вместо него для sort key.

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Создайте таблицу вручную (мы рекомендуем, чтобы таблица имела ту же схему, что и файл Parquet, который вы хотите загрузить из HDFS):

```
CREATE TABLE user_behavior_declared
(
    UserID int(11),
    ItemID int(11),
    CategoryID int(11),
    BehaviorType varchar(65533),
    Timestamp varbinary
)
ENGINE = OLAP
DUPLICATE KEY(UserID)
DISTRIBUTED BY HASH(UserID);
```

Отобразите схему, чтобы вы могли сравнить её с выведенной схемой, созданной табличной функцией `FILES()`:

```
DESCRIBE user_behavior_declared;
```

```
+--------------+----------------+------+-------+---------+-------+
| Field        | Type           | Null | Key   | Default | Extra |
+--------------+----------------+------+-------+---------+-------+
| UserID       | int            | NO   | true  | NULL    |       |
| ItemID       | int            | NO   | false | NULL    |       |
| CategoryID   | int            | NO   | false | NULL    |       |
| BehaviorType | varchar(65533) | NO   | false | NULL    |       |
| Timestamp    | varbinary      | NO   | false | NULL    |       |
+--------------+----------------+------+-------+---------+-------+
5 rows in set (0.00 sec)
```

ПОДСКАЗКА

Сравните схему, которую вы только что создали, со схемой, выведенной ранее с помощью табличной функции `FILES()`. Посмотрите на:

- типы данных
- nullable
- ключевые поля

Для лучшего контроля схемы целевой таблицы и лучшей производительности запросов мы рекомендуем указывать схему таблицы вручную в производственных средах.

После создания таблицы вы можете загрузить её с помощью INSERT INTO SELECT FROM FILES():

```
INSERT INTO user_behavior_declared
SELECT * FROM FILES
(
    "path" = "hdfs://<hdfs_ip>:
<hdfs_port>/user/amber/user_behavior_ten_million_rows.parquet",
    "format" = "parquet",
    "hadoop.security.authentication" = "simple",
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

После завершения загрузки вы можете запросить таблицу, чтобы убедиться, что данные были загружены в неё. Пример:

```
SELECT * from user_behavior_declared LIMIT 3;
```

Возвращается следующий результат запроса, указывающий, что данные были успешно загружены:

```
+--------+---------+------------+--------------+---------------------+
| UserID | ItemID  | CategoryID | BehaviorType | Timestamp           |
+--------+---------+------------+--------------+---------------------+
|    107 | 1568743 |    4476428 | pv           | 2017-11-25 14:29:53 |
|    107 |  470767 |    1020087 | pv           | 2017-11-25 14:32:31 |
|    107 |  358238 |    1817004 | pv           | 2017-11-25 14:43:23 |
+--------+---------+------------+--------------+---------------------+
```

Проверка прогресса загрузки

Вы можете запросить прогресс заданий INSERT из представления `loads` в Information Schema Selena. Эта функция поддерживается начиная с версии v1.5.2. Пример:

```
SELECT * FROM information_schema.loads ORDER BY JOB_ID DESC;
```

Для получения информации о полях, предоставляемых в представлении `loads`, см. `loads`.

Если вы отправили несколько заданий загрузки, вы можете фильтровать по `LABEL`, связанному с заданием. Пример:

```
SELECT * FROM information_schema.loads WHERE LABEL = 'insert_0d86c3f9-851f-11
*************************** 1. row ***************************
              JOB_ID: 10214
               LABEL: insert_0d86c3f9-851f-11ee-9c3e-00163e044958
       DATABASE_NAME: mydatabase
               STATE: FINISHED
            PROGRESS: ETL:100%; LOAD:100%
                TYPE: INSERT
            PRIORITY: NORMAL
           SCAN_ROWS: 10000000
       FILTERED_ROWS: 0
     UNSELECTED_ROWS: 0
           SINK_ROWS: 10000000
            ETL_INFO:
           TASK_INFO: resource:N/A; timeout(s):300; max_filter_ratio:0.0
         CREATE_TIME: 2023-11-17 15:58:14
      ETL_START_TIME: 2023-11-17 15:58:14
     ETL_FINISH_TIME: 2023-11-17 15:58:14
     LOAD_START_TIME: 2023-11-17 15:58:14
    LOAD_FINISH_TIME: 2023-11-17 15:58:18
         JOB_DETAILS: {"All backends":{"0d86c3f9-851f-11ee-9c3e-00163e044958"
[10120]},"FileNumber":0,"FileSize":0,"InternalTableLoadBytes":311710786,"Inte
backends":{"0d86c3f9-851f-11ee-9c3e-00163e044958":[]}}
           ERROR_MSG: NULL
        TRACKING_URL: NULL
        TRACKING_SQL: NULL
REJECTED_RECORD_PATH: NULL
```

> **ПРИМЕЧАНИЕ**
>
> INSERT — это синхронная команда. Если задание INSERT всё ещё выполняется, вам нужно открыть другую сессию, чтобы проверить его статус выполнения.

3.4.3 Использование Broker Load

Асинхронный процесс Broker Load обрабатывает подключение к HDFS, извлечение данных и сохранение данных в Selena.

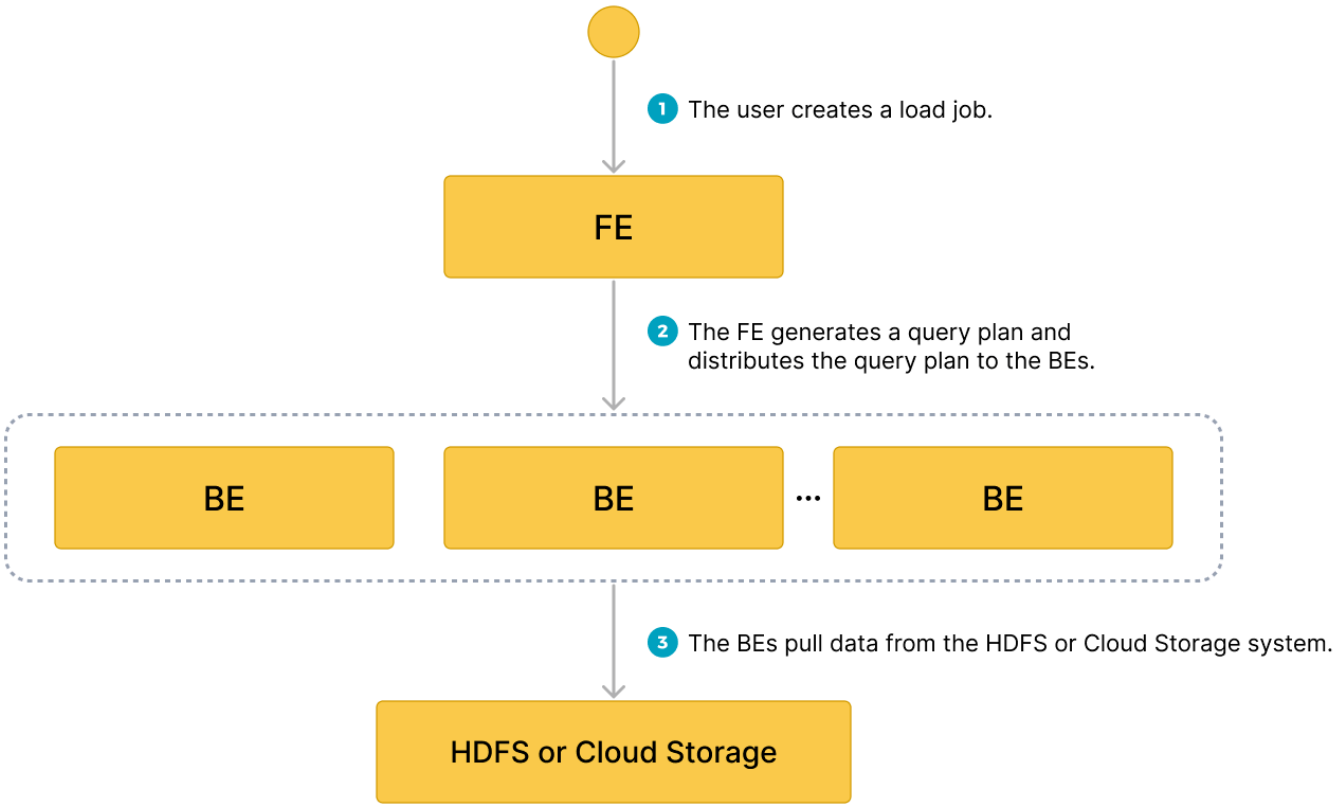
Этот метод поддерживает следующие форматы файлов:

- Parquet
- ORC
- CSV
- JSON (поддерживается начиная с версии v1.5.2)

Преимущества Broker Load

- Broker Load выполняется в фоновом режиме, и клиентам не нужно оставаться подключёнными для продолжения задания.
- Broker Load предпочтителен для длительных заданий, тайм-аут по умолчанию составляет 4 часа.
- В дополнение к форматам файлов Parquet и ORC, Broker Load поддерживает формат файлов CSV и формат файлов JSON (формат файлов JSON поддерживается начиная с версии v1.5.2).

Поток данных



1. Пользователь создаёт задание загрузки.
2. Frontend (FE) создаёт план запроса и распределяет план на backend-узлы (BE) или compute-узлы (CN).
3. BE или CN извлекают данные из источника и загружают данные в Selena.

Типичный пример

Создайте таблицу, запустите процесс загрузки, который извлекает файл данных `/user/amber/user_behavior_ten_million_rows.parquet` из HDFS, и проверьте прогресс и успешность загрузки данных.

Создание базы данных и таблицы

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Создайте таблицу вручную (мы рекомендуем, чтобы таблица имела ту же схему, что и файл Parquet, который вы хотите загрузить из HDFS):

```
CREATE TABLE user_behavior
(
    UserID int(11),
    ItemID int(11),
    CategoryID int(11),
    BehaviorType varchar(65533),
    Timestamp varbinary
)
ENGINE = OLAP
DUPLICATE KEY(UserID)
DISTRIBUTED BY HASH(UserID);
```

Запуск Broker Load

Выполните следующую команду, чтобы запустить задание Broker Load, которое загружает данные из файла данных `/user/amber/user_behavior_ten_million_rows.parquet` в таблицу `user_behavior`:

```
LOAD LABEL user_behavior
(
    DATA INFILE("hdfs://<hdfs_ip>:
<hdfs_port>/user/amber/user_behavior_ten_million_rows.parquet")
    INTO TABLE user_behavior
    FORMAT AS "parquet"
 )
 WITH BROKER
(
    "hadoop.security.authentication" = "simple",
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
)
PROPERTIES
(
    "timeout" = "72000"
);
```

Это задание имеет четыре основных раздела:

- `LABEL`: Строка, используемая при запросе состояния задания загрузки.
- `LOAD` declaration: URI источника, формат исходных данных и имя целевой таблицы.
- `BROKER`: Данные подключения к источнику.
- `PROPERTIES`: Значение тайм-аута и любые другие свойства, применяемые к заданию загрузки.

Подробный синтаксис и описание параметров см. в BROKER LOAD.

Проверка прогресса загрузки

Вы можете запросить прогресс заданий Broker Load из представления `information_schema.loads`. Эта функция поддерживается начиная с версии v1.5.2.

```
SELECT * FROM information_schema.loads;
```

Для получения информации о полях, предоставляемых в представлении `loads`, см. Information Schema).

Если вы отправили несколько заданий загрузки, вы можете фильтровать по `LABEL`, связанному с заданием. Пример:

```
SELECT * FROM information_schema.loads WHERE LABEL = 'user_behavior';
```

В выводе ниже есть две записи для задания загрузки `user_behavior`:

- Первая запись показывает состояние `CANCELLED`. Прокрутите до `ERROR_MSG`, и вы увидите, что задание завершилось неудачей из-за `listPath failed`.
- Вторая запись показывает состояние `FINISHED`, что означает, что задание выполнено успешно.

```
JOB_ID|LABEL                                       |DATABASE_NAME|STATE    |PR
|CREATE_TIME        |ETL_START_TIME     |ETL_FINISH_TIME    |LOAD_START_TIME
|ERROR_MSG                              |TRACKING_URL|TRACKING_SQL|REJECTED_RE
------+----------------------------------------------+-------------+---------+--
--------------------------------------+-------------------+------------------
-------------------------------------------------------------------------------
--------------------------+---------------------------------------+-----------
 10121|user_behavior                                 |mydatabase   |CANCELLED|ET
timeout(s):72000; max_filter_ratio:0.0|2023-08-10 14:59:30|
{},"FileNumber":0,"FileSize":0,"InternalTableLoadBytes":0,"InternalTableLoadR
|type:ETL_RUN_FAIL; msg:listPath failed|             |             |
 10106|user_behavior                                 |mydatabase   |FINISHED |ET
timeout(s):72000; max_filter_ratio:0.0|2023-08-10 14:50:15|2023-08-10 14:50:1
c7348f9a5f2f":
[10004]},"FileNumber":1,"FileSize":1225637388,"InternalTableLoadBytes":271060
backends":{"a5|                                       |             |
```

После того как вы убедитесь, что задание загрузки завершено, вы можете проверить подмножество целевой таблицы, чтобы увидеть, были ли данные успешно загружены. Пример:

```
SELECT * from user_behavior LIMIT 3;
```

Возвращается следующий результат запроса, указывающий, что данные были успешно загружены:

```
+--------+---------+------------+--------------+---------------------+
| UserID | ItemID  | CategoryID | BehaviorType | Timestamp           |
+--------+---------+------------+--------------+---------------------+
|    142 | 2869980 |    2939262 | pv           | 2017-11-25 03:43:22 |
|    142 | 2522236 |    1669167 | pv           | 2017-11-25 15:14:12 |
|    142 | 3031639 |    3607361 | pv           | 2017-11-25 15:19:25 |
+--------+---------+------------+--------------+---------------------+
```

3.4.4 Использование Pipe

Начиная с версии v1.5.2, Selena предоставляет метод загрузки Pipe, который в настоящее время поддерживает только форматы файлов Parquet и ORC.

Преимущества Pipe

Pipe идеально подходит для непрерывной загрузки данных и крупномасштабной загрузки данных:

- **Крупномасштабная загрузка данных микро-пакетами помогает снизить затраты на повторные попытки, вызванные ошибками данных.**

  С помощью Pipe Selena обеспечивает эффективную загрузку большого количества файлов данных со значительным общим объёмом данных. Pipe автоматически разбивает файлы на основе их количества или размера, разделяя задание загрузки на более мелкие последовательные задачи. Такой подход гарантирует, что ошибки в одном файле не повлияют на всё задание загрузки. Статус загрузки каждого файла записывается Pipe, что позволяет легко идентифицировать и исправлять файлы, содержащие ошибки. Минимизируя необходимость повторных попыток из-за ошибок данных, этот подход помогает снизить затраты.

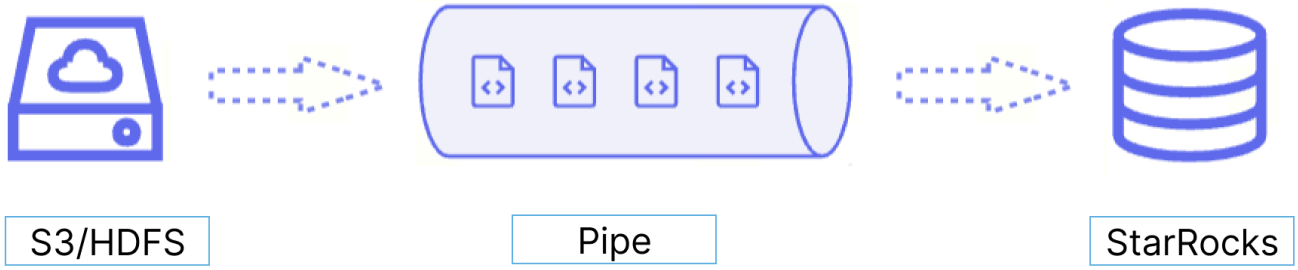
- **Непрерывная загрузка данных помогает сократить трудозатраты.**

  Pipe помогает записывать новые или обновлённые файлы данных в определённое место и непрерывно загружать новые данные из этих файлов в Selena. После создания задания Pipe с указанием `"AUTO_INGEST" = "TRUE"` оно будет постоянно отслеживать изменения файлов данных, хранящихся по указанному пути, и автоматически загружать новые или обновлённые данные из файлов данных в целевую таблицу Selena.

Кроме того, Pipe выполняет проверку уникальности файлов, чтобы предотвратить повторную загрузку данных. В процессе загрузки Pipe проверяет уникальность каждого файла данных на основе имени файла и дайджеста. Если файл с определённым именем и дайджестом уже был обработан заданием Pipe, это задание Pipe пропустит все последующие файлы с тем же именем и дайджестом. Обратите внимание, что HDFS uses LastModifiedTime используется как дайджест файла.

Статус загрузки каждого файла данных записывается и сохраняется в представлении `information_schema.pipe_files`. После удаления задания Pipe, связанного с этим представлением, записи о файлах, загруженных в этом задании, также будут удалены.

Поток данных



Pipe идеально подходит для непрерывной загрузки данных и крупномасштабной загрузки данных:

- **Крупномасштабная загрузка данных микропакетами помогает снизить затраты на повторные попытки, вызванные ошибками данных.**

  С помощью Pipe Selena обеспечивает эффективную загрузку большого количества файлов данных со значительным общим объёмом данных. Pipe автоматически разбивает файлы на основе их количества или размера, разделяя задание загрузки на меньшие последовательные задачи. Этот подход гарантирует, что ошибки в одном файле не повлияют на всё задание загрузки. Статус загрузки каждого файла записывается Pipe, что позволяет легко идентифицировать и исправлять файлы, содержащие ошибки. Минимизируя необходимость повторных попыток из-за ошибок данных, этот подход помогает снизить затраты.

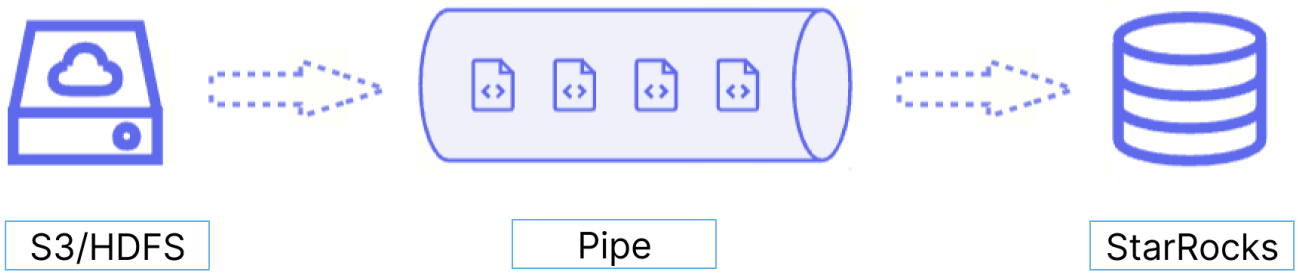
- **Непрерывная загрузка данных помогает сократить трудозатраты.**

Pipe помогает записывать новые или обновлённые файлы данных в определённое место и непрерывно загружать новые данные из этих файлов в Selena. После создания задания Pipe с указанием `"AUTO_INGEST" = "TRUE"` оно будет постоянно отслеживать изменения в файлах данных, хранящихся по указанному пути, и автоматически загружать новые или обновлённые данные из файлов данных в целевую таблицу Selena.

Кроме того, Pipe выполняет проверки уникальности файлов, чтобы предотвратить дублирование загрузки данных. В процессе загрузки Pipe проверяет уникальность каждого файла данных на основе имени файла и дайджеста. Если файл с определённым именем и дайджестом уже был обработан заданием Pipe, задание Pipe пропустит все последующие файлы с тем же именем и дайджестом. Обратите внимание, что HDFS использует `LastModifiedTime` в качестве дайджеста файла.

Статус загрузки каждого файла данных записывается и сохраняется в представлении `information_schema.pipe_files`. После удаления задания Pipe, связанного с представлением, записи о файлах, загруженных в этом задании, также будут удалены.

Поток данных



Различия между Pipe и INSERT+FILES()

Задание Pipe разделяется на одну или несколько транзакций в зависимости от размера и количества строк в каждом файле данных. Пользователи могут запрашивать промежуточные результаты во время процесса загрузки. Напротив, задание INSERT+`FILES()` обрабатывается как одна транзакция, и пользователи не могут просматривать данные во время процесса загрузки.

Последовательность загрузки файлов

Для каждого задания Pipe Selena поддерживает очередь файлов, из которой она извлекает и загружает файлы данных в виде микропакетов. Pipe не гарантирует, что файлы данных загружаются в том же порядке, в котором они были загружены. Поэтому более новые данные могут быть загружены раньше более старых данных.

Типичный пример

Создание базы данных и таблицы

Создайте базу данных и переключитесь на неё:

```
CREATE DATABASE IF NOT EXISTS mydatabase;
USE mydatabase;
```

Создайте таблицу вручную (мы рекомендуем, чтобы таблица имела ту же схему, что и файл Parquet, который вы хотите загрузить из HDFS):

```
CREATE TABLE user_behavior_replica
(
    UserID int(11),
    ItemID int(11),
    CategoryID int(11),
    BehaviorType varchar(65533),
    Timestamp varbinary
)
ENGINE = OLAP
DUPLICATE KEY(UserID)
DISTRIBUTED BY HASH(UserID);
```

Запуск задания Pipe

Выполните следующую команду, чтобы запустить задание Pipe, которое загружает данные из файла данных `/user/amber/user_behavior_ten_million_rows.parquet` в таблицу `user_behavior_replica`:

```
CREATE PIPE user_behavior_replica
PROPERTIES
(
    "AUTO_INGEST" = "TRUE"
)
AS
INSERT INTO user_behavior_replica
SELECT * FROM FILES
(
    "path" = "hdfs://<hdfs_ip>:
<hdfs_port>/user/amber/user_behavior_ten_million_rows.parquet",
    "format" = "parquet",
    "hadoop.security.authentication" = "simple",
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
);
```

Это задание имеет четыре основных раздела:

- `pipe_name`: Имя pipe. Имя pipe должно быть уникальным в базе данных, к которой принадлежит pipe.
- `INSERT_SQL`: Оператор INSERT INTO SELECT FROM FILES, который используется для загрузки данных из указанного исходного файла данных в целевую таблицу.
- `PROPERTIES`: Набор необязательных параметров, которые определяют, как выполнять pipe. Они включают `AUTO_INGEST`, `POLL_INTERVAL`, `BATCH_SIZE` и `BATCH_FILES`. Укажите эти свойства в формате `"key" = "value"`.

Подробный синтаксис и описание параметров см. в CREATE PIPE.

Проверка прогресса загрузки

- Запросите прогресс заданий Pipe с помощью SHOW PIPES.

  ```
  SHOW PIPES;
  ```

  Если вы отправили несколько заданий загрузки, вы можете фильтровать по `NAME`, связанному с заданием. Пример:

```
SHOW PIPES WHERE NAME = 'user_behavior_replica' \G
*************************** 1. row ***************************
DATABASE_NAME: mydatabase
      PIPE_ID: 10252
    PIPE_NAME: user_behavior_replica
        STATE: RUNNING
   TABLE_NAME: mydatabase.user_behavior_replica
  LOAD_STATUS:
{"loadedFiles":1,"loadedBytes":132251298,"loadingFiles":0,"lastLoadedTime
11-17 16:13:22"}
   LAST_ERROR: NULL
 CREATED_TIME: 2023-11-17 16:13:15
1 row in set (0.00 sec)
```

- Запросите прогресс заданий Pipe из представления `pipes` в Information Schema Selena.

```
SELECT * FROM information_schema.pipes;
```

Если вы отправили несколько заданий загрузки, вы можете фильтровать по `PIPE_NAME`, связанному с заданием. Пример:

```
SELECT * FROM information_schema.pipes WHERE pipe_name = 'user_behavior_r
\G
*************************** 1. row ***************************
DATABASE_NAME: mydatabase
      PIPE_ID: 10252
    PIPE_NAME: user_behavior_replica
        STATE: RUNNING
   TABLE_NAME: mydatabase.user_behavior_replica
  LOAD_STATUS:
{"loadedFiles":1,"loadedBytes":132251298,"loadingFiles":0,"lastLoadedTime
11-17 16:13:22"}
   LAST_ERROR:
 CREATED_TIME: 2023-11-17 16:13:15
1 row in set (0.00 sec)
```

Проверка статуса файлов

Вы можете запросить статус загрузки файлов, загруженных из представления `pipe_files` в Information Schema Selena.

```
SELECT * FROM information_schema.pipe_files;
```

Если вы отправили несколько заданий загрузки, вы можете фильтровать по `PIPE_NAME`, связанному с заданием. Пример:

```
SELECT * FROM information_schema.pipe_files WHERE pipe_name =
'user_behavior_replica' \G
*************************** 1. row ***************************
   DATABASE_NAME: mydatabase
         PIPE_ID: 10252
       PIPE_NAME: user_behavior_replica
       FILE_NAME:
hdfs://172.26.195.67:9000/user/amber/user_behavior_ten_million_rows.parquet
    FILE_VERSION: 1700035418838
       FILE_SIZE: 132251298
   LAST_MODIFIED: 2023-11-15 08:03:38
      LOAD_STATE: FINISHED
     STAGED_TIME: 2023-11-17 16:13:16
 START_LOAD_TIME: 2023-11-17 16:13:17
FINISH_LOAD_TIME: 2023-11-17 16:13:22
       ERROR_MSG:
1 row in set (0.02 sec)
```

Управление Pipe

Вы можете изменять, приостанавливать или возобновлять, удалять или запрашивать созданные вами pipe, а также повторять загрузку определённых файлов данных. Для получения дополнительной информации см. ALTER PIPE, SUSPEND or RESUME PIPE, DROP PIPE, SHOW PIPES и RETRY FILE.

3.5 Загрузка из Apache Kafka

3.5.1 Загрузка данных с помощью Kafka connector

Selena предоставляет собственный connector под названием Apache Kafka® connector (Selena Connector for Apache Kafka®, сокращённо Kafka connector), который является sink

connector'ом и непрерывно потребляет сообщения из Kafka, загружая их в Selena. Kafka connector гарантирует семантику at-least-once.

Kafka connector может легко интегрироваться с Kafka Connect, что позволяет Selena лучше интегрироваться с экосистемой Kafka. Это мудрый выбор, если вы хотите загружать данные в реальном времени в Selena. По сравнению с Routine Load, рекомендуется использовать Kafka connector в следующих сценариях:

- В отличие от Routine Load, которая поддерживает загрузку данных только в форматах CSV, JSON и Avro, Kafka connector может загружать данные в большем количестве форматов, таких как Protobuf. Если данные можно преобразовать в форматы JSON и CSV с помощью конвертеров Kafka Connect, то данные можно загрузить в Selena через Kafka connector.
- Настройка пользовательского преобразования данных, например, CDC-данных в формате Debezium.
- Загрузка данных из нескольких топиков Kafka.
- Загрузка данных из Confluent Cloud.
- Более точный контроль над размерами пакетов загрузки, параллелизмом и другими параметрами для достижения баланса между скоростью загрузки и использованием ресурсов.

Подготовка

Требования к версиям

Connector	Kafka	Selena	Java
1.0.4	3.4	2.5 и выше	8
1.0.3	3.4	2.5 и выше	8

Настройка окружения Kafka

Поддерживаются как самоуправляемые кластеры Apache Kafka, так и Confluent Cloud.

- Для самоуправляемого кластера Apache Kafka вы можете обратиться к Apache Kafka quickstart для быстрого развёртывания кластера Kafka. Kafka Connect уже интегрирован в Kafka.
- Для Confluent Cloud убедитесь, что у вас есть учётная запись Confluent и вы создали кластер.

Загрузка Kafka connector

Добавьте Kafka connector в Kafka Connect:

- Самоуправляемый кластер Kafka:

  Скачайте и распакуйте selena-kafka-connector-xxx.tar.gz.

- Confluent Cloud:

  В настоящее время Kafka connector не загружен в Confluent Hub. Вам нужно скачать и распаковать selena-kafka-connector-xxx.tar.gz, упаковать его в ZIP-файл и загрузить ZIP-файл в Confluent Cloud.

Конфигурация сети

Убедитесь, что машина, на которой расположен Kafka, может получить доступ к узлам FE кластера Selena через `http_port` (по умолчанию: `8030`) и `query_port` (по умолчанию: `9030`), а также к узлам BE через `be_http_port` (по умолчанию: `8040`).

Использование

В этом разделе в качестве примера используется самоуправляемый кластер Kafka для объяснения того, как настроить Kafka connector и Kafka Connect, а затем запустить Kafka Connect для загрузки данных в Selena.

Подготовка набора данных

Предположим, что в топике `test` кластера Kafka существуют данные в формате JSON.

```
{"id":1,"city":"New York"}
{"id":2,"city":"Los Angeles"}
{"id":3,"city":"Chicago"}
```

Создание таблицы

Создайте таблицу `test_tbl` в базе данных `example_db` в кластере Selena в соответствии с ключами данных в формате JSON.

```
CREATE DATABASE example_db;
USE example_db;
CREATE TABLE test_tbl (id INT, city STRING);
```

Настройка Kafka connector и Kafka Connect, запуск Kafka Connect для загрузки данных

Запуск Kafka Connect в автономном режиме

1. Настройте Kafka connector. В директории **config** в каталоге установки Kafka создайте файл конфигурации **connect-Selena-sink.properties** для Kafka connector и настройте следующие параметры. Больше параметров и их описаний см. в разделе Параметры.

> **К СВЕДЕНИЮ**
>
> - В этом примере Kafka connector, предоставляемый Selena, является sink connector'ом, который может непрерывно потреблять данные из Kafka и загружать данные в Selena.
> - Если исходные данные являются CDC-данными, например, данными в формате Debezium, и таблица Selena является таблицей Primary Key, вам также необходимо настроить `transform` в файле конфигурации **connect-Selena-sink.properties** для Kafka connector, предоставляемого Selena, чтобы синхронизировать изменения исходных данных с таблицей Primary Key.

```
name=selena-kafka-connector
connector.class=com.selena.connector.kafka.SelenaSinkConnector
topics=test
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter.schemas.enable=false
# HTTP URL узла FE в вашем кластере Selena. Порт по умолчанию - 8030.
selena.http.url=192.168.xxx.xxx:8030
# Если имя топика Kafka отличается от имени таблицы Selena, необходимо
настроить соответствие между ними.
selena.topic2table.map=test:test_tbl
# Введите имя пользователя Selena.
selena.username=user1
# Введите пароль Selena.
selena.password=123456
selena.database.name=example_db
sink.properties.strip_outer_array=true
```

2. Настройте и запустите Kafka Connect.

   i. Настройте Kafka Connect. В файле конфигурации **config/connect-standalone.properties** в директории **config** настройте следующие параметры. Больше параметров и их описаний см. в разделе Running Kafka Connect. Обратите внимание, что в следующих примерах используется версия selena-kafka-connector `1.0.3`. Если вы используете более новую версию, вам необходимо внести соответствующие изменения.

```
# Адреса брокеров Kafka. Несколько адресов брокеров Kafka должны
быть разделены запятыми (,).
# Обратите внимание, что в этом примере используется PLAINTEXT в
качестве протокола безопасности для доступа к кластеру Kafka. Если
вы используете другой протокол безопасности для доступа к кластеру
Kafka, вам необходимо настроить соответствующую информацию в этом
файле.
bootstrap.servers=<kafka_broker_ip>:9092
offset.storage.file.filename=/tmp/connect.offsets
offset.flush.interval.ms=10000
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter.schemas.enable=false
# Абсолютный путь к Kafka connector после распаковки. Например:
plugin.path=/home/kafka-connect/selena-kafka-connector-1.0.3
```

ii. Запустите Kafka Connect.

```
CLASSPATH=/home/kafka-connect/selena-kafka-connector-1.0.3/*
bin/connect-standalone.sh config/connect-standalone.properties
config/connect-selena-sink.properties
```

Запуск Kafka Connect в распределённом режиме

1. Настройте и запустите Kafka Connect.

   i. Настройте Kafka Connect. В файле конфигурации `config/connect-distributed.properties` в директории **config** настройте следующие параметры. Больше параметров и их описаний см. в разделе Running Kafka Connect.

```
# Адреса брокеров Kafka. Несколько адресов брокеров Kafka должны быть разделены запятыми (,).
# Обратите внимание, что в этом примере используется PLAINTEXT в качестве протокола безопасности для доступа к кластеру Kafka. Если вы используете другой протокол безопасности для доступа к кластеру Kafka, вам необходимо настроить соответствующую информацию в этом файле.
bootstrap.servers=<kafka_broker_ip>:9092
offset.storage.file.filename=/tmp/connect.offsets
offset.flush.interval.ms=10000
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter.schemas.enable=false
# Абсолютный путь к Kafka connector после распаковки. Например:
plugin.path=/home/kafka-connect/selena-kafka-connector-1.0.3
```

ii. Запустите Kafka Connect.

```
CLASSPATH=/home/kafka-connect/selena-kafka-connector-1.0.3/*
bin/connect-distributed.sh config/connect-distributed.properties
```

2. Настройте и создайте Kafka connector. Обратите внимание, что в распределённом режиме вам необходимо настроить и создать Kafka connector через REST API. Параметры и их описания см. в разделе Параметры.

> **К СВЕДЕНИЮ**
>
> - В этом примере Kafka connector, предоставляемый Selena, является sink connector'ом, который может непрерывно потреблять данные из Kafka и загружать данные в Selena.
> - Если исходные данные являются CDC-данными, например, данными в формате Debezium, и таблица Selena является таблицей Primary Key, вам также необходимо настроить `transform` в файле конфигурации **connect-Selena-sink.properties** для Kafka connector, предоставляемого Selena, чтобы синхронизировать изменения исходных данных с таблицей Primary Key.

```
curl -i http://127.0.0.1:8083/connectors -H "Content-Type:
application/json" -X POST -d '{
  "name":"selena-kafka-connector",
  "config":{
    "connector.class":"com.selena.connector.kafka.SelenaSinkConnector",
    "topics":"test",
    "key.converter":"org.apache.kafka.connect.json.JsonConverter",
    "value.converter":"org.apache.kafka.connect.json.JsonConverter",
    "key.converter.schemas.enable":"true",
    "value.converter.schemas.enable":"false",
    "selena.http.url":"192.168.xxx.xxx:8030",
    "selena.topic2table.map":"test:test_tbl",
    "selena.username":"user1",
    "selena.password":"123456",
    "selena.database.name":"example_db",
    "sink.properties.strip_outer_array":"true"
  }
}'
```

Запрос таблицы Selena

Запросите целевую таблицу Selena `test_tbl`.

```
MySQL [example_db]> select * from test_tbl;

+------+-------------+
| id   | city        |
+------+-------------+
|    1 | New York    |
|    2 | Los Angeles |
|    3 | Chicago     |
+------+-------------+
3 rows in set (0.01 sec)
```

Данные успешно загружены, когда возвращается вышеуказанный результат.

Параметры

name

Обязательный: ДА

Значение по умолчанию:

Описание: Имя для этого Kafka connector. Оно должно быть глобально уникальным среди всех Kafka connector'ов в этом кластере Kafka Connect. Например, selena-kafka-connector.

connector.class

Обязательный: ДА
Значение по умолчанию:
Описание: Класс, используемый sink'ом этого Kafka connector. Установите значение `com.selena.connector.kafka.SelenaSinkConnector`.

topics

Обязательный:
Значение по умолчанию:
Описание: Один или несколько топиков для подписки, где каждый топик соответствует таблице Selena. По умолчанию Selena предполагает, что имя топика совпадает с именем таблицы Selena. Таким образом, Selena определяет целевую таблицу Selena, используя имя топика. Пожалуйста, выберите заполнение либо `topics`, либо `topics.regex` (ниже), но не оба одновременно. Однако, если имя таблицы Selena не совпадает с именем топика, используйте необязательный параметр `selena.topic2table.map` (ниже) для указания соответствия между именем топика и именем таблицы.

topics.regex

Обязательный:
Значение по умолчанию: **Описание**: Регулярное выражение для сопоставления одного или нескольких топиков для подписки. Более подробное описание см. в `topics`. Пожалуйста, выберите заполнение либо `topics.regex`, либо `topics` (выше), но не оба одновременно.

selena.topic2table.map

Обязательный: НЕТ
Значение по умолчанию:
Описание: Соответствие между именем таблицы Selena и именем топика, когда имя топика отличается от имени таблицы Selena. Формат: `<topic-1>:<table-1>,<topic-2>:<table-2>,...`.

selena.http.url

Обязательный: ДА
Значение по умолчанию:
Описание: HTTP URL узла FE в вашем кластере Selena. Формат: `<fe_host1>:<fe_http_port1>,<fe_host2>:<fe_http_port2>,...`. Несколько адресов разделяются запятыми (,). Например, `192.168.xxx.xxx:8030,192.168.xxx.xxx:8030`.

selena.database.name

Обязательный: ДА
Значение по умолчанию:
Описание: Имя базы данных Selena.

selena.username

Обязательный: ДА
Значение по умолчанию:
Описание: Имя пользователя вашей учётной записи кластера Selena. Пользователь должен иметь привилегию INSERT на таблицу Selena.

selena.password

Обязательный: ДА
Значение по умолчанию:
Описание: Пароль вашей учётной записи кластера Selena.

key.converter

Обязательный: НЕТ
Значение по умолчанию: Key converter, используемый кластером Kafka Connect
Описание: Этот параметр указывает key converter для sink connector (Kafka-connector-selena), который используется для десериализации ключей данных Kafka. Key converter по умолчанию - тот, который используется кластером Kafka Connect.

value.converter

Обязательный: НЕТ
Значение по умолчанию: Value converter, используемый кластером Kafka Connect
Описание: Этот параметр указывает value converter для sink connector (Kafka-connector-

selena), который используется для десериализации значений данных Kafka. Value converter по умолчанию - тот, который используется кластером Kafka Connect.

key.converter.schema.registry.url

Обязательный: НЕТ
Значение по умолчанию:
Описание: URL Schema registry для key converter.

value.converter.schema.registry.url

Обязательный: НЕТ
Значение по умолчанию:
Описание: URL Schema registry для value converter.

tasks.max

Обязательный: НЕТ
Значение по умолчанию: 1
Описание: Верхний лимит для количества потоков задач, которые может создать Kafka connector, который обычно совпадает с количеством ядер CPU на рабочих узлах в кластере Kafka Connect. Вы можете настроить этот параметр для управления производительностью загрузки.

bufferflush.maxbytes

Обязательный: НЕТ
Значение по умолчанию: 94371840(90M)
Описание: Максимальный размер данных, которые могут быть накоплены в памяти перед отправкой в Selena за один раз. Максимальное значение варьируется от 64 MB до 10 GB. Имейте в виду, что буфер Stream Load SDK может создавать несколько заданий Stream Load для буферизации данных. Поэтому упомянутый здесь порог относится к общему размеру данных.

bufferflush.intervalms

Обязательный: НЕТ
Значение по умолчанию: 1000
Описание: Интервал для отправки пакета данных, который контролирует задержку загрузки. Диапазон: [1000, 3600000].

connect.timeoutms

Обязательный: НЕТ
Значение по умолчанию: 1000
Описание: Тайм-аут для подключения к HTTP URL. Диапазон: [100, 60000].

sink.properties.*

Обязательный:
Значение по умолчанию:
Описание: Параметры Stream Load для управления поведением загрузки. Например, параметр `sink.properties.format` указывает формат, используемый для Stream Load, такой как CSV или JSON. Список поддерживаемых параметров и их описания см. в разделе STREAM LOAD.

sink.properties.format

Обязательный: НЕТ
Значение по умолчанию: json
Описание: Формат, используемый для Stream Load. Kafka connector будет преобразовывать каждый пакет данных в указанный формат перед отправкой их в Selena. Допустимые значения: `csv` и `json`. Для получения дополнительной информации см. Параметры CSV и Параметры JSON.

sink.properties.partial_update

Обязательный: НЕТ
Значение по умолчанию: `FALSE`
Описание: Использовать ли частичные обновления. Допустимые значения: `TRUE` и `FALSE`. Значение по умолчанию: `FALSE`, что означает отключение этой функции.

sink.properties.partial_update_mode

Обязательный: НЕТ
Значение по умолчанию: `row`
Описание: Указывает режим для частичных обновлений. Допустимые значения: `row` и `column`.

- Значение `row` (по умолчанию) означает частичные обновления в режиме строк, что более подходит для обновлений в реальном времени с большим количеством

столбцов и небольшими пакетами.
- Значение `column` означает частичные обновления в режиме столбцов, что более подходит для пакетных обновлений с небольшим количеством столбцов и большим количеством строк. В таких сценариях включение режима столбцов обеспечивает более высокую скорость обновления. Например, в таблице с 100 столбцами, если обновляется только 10 столбцов (10% от общего количества) для всех строк, скорость обновления в режиме столбцов в 10 раз выше.

Примечания по использованию

Политика сброса

Kafka connector будет буферизовать данные в памяти и сбрасывать их пакетами в Selena через Stream Load. Сброс будет запущен при выполнении любого из следующих условий:

- Размер буферизованных строк в байтах достигает лимита `bufferflush.maxbytes`.
- Время, прошедшее с момента последнего сброса, достигает лимита `bufferflush.intervalms`.
- Достигается интервал, с которым connector пытается фиксировать смещения для задач. Интервал контролируется конфигурацией Kafka Connect `offset.flush.interval.ms`, и значение по умолчанию составляет `60000`.

Для снижения задержки данных настройте эти конфигурации в настройках Kafka connector. Однако более частые сбросы увеличат использование CPU и I/O.

Ограничения

- Не поддерживается преобразование одного сообщения из топика Kafka в несколько строк данных и загрузка в Selena.
- Sink Kafka connector, предоставляемый Selena, гарантирует семантику at-least-once.

Лучшие практики

Загрузка CDC-данных в формате Debezium

Debezium - это популярный инструмент Change Data Capture (CDC), который поддерживает мониторинг изменений данных в различных системах баз данных и потоковую передачу этих изменений в Kafka. Следующий пример демонстрирует, как настроить и использовать Kafka connector для записи изменений PostgreSQL в **таблицу Primary Key** в Selena.

Шаг 1: Установка и запуск Kafka

> **ПРИМЕЧАНИЕ**
>
> Вы можете пропустить этот шаг, если у вас есть собственное окружение Kafka.

1. Скачайте последний релиз Kafka с официального сайта и распакуйте пакет.

```
tar -xzf kafka_2.13-3.7.0.tgz
cd kafka_2.13-3.7.0
```

2. Запустите окружение Kafka.

   Сгенерируйте UUID кластера Kafka.

```
KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"
```

   Отформатируйте директории логов.

```
bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c
config/kraft/server.properties
```

   Запустите сервер Kafka.

```
bin/kafka-server-start.sh config/kraft/server.properties
```

Шаг 2: Настройка PostgreSQL

1. Убедитесь, что пользователю PostgreSQL предоставлены привилегии `REPLICATION`.

2. Настройте конфигурацию PostgreSQL.

   Установите `wal_level` в `logical` в **postgresql.conf**.

```
wal_level = logical
```

   Перезапустите сервер PostgreSQL для применения изменений.

```
pg_ctl restart
```

3. Подготовьте набор данных.

   Создайте таблицу и вставьте тестовые данные.

```
CREATE TABLE customers (
  id int primary key ,
  first_name varchar(65533) NULL,
  last_name varchar(65533) NULL ,
  email varchar(65533) NULL
);

INSERT INTO customers VALUES (1,'a','a','a@a.com');
```

4. Проверьте сообщения CDC-логов в Kafka.

```
{
    "schema": {
        "type": "struct",
        "fields": [
            {
                "type": "struct",
                "fields": [
                    {
                        "type": "int32",
                        "optional": false,
                        "field": "id"
                    },
                    {
                        "type": "string",
                        "optional": true,
                        "field": "first_name"
                    },
                    {
                        "type": "string",
                        "optional": true,
                        "field": "last_name"
                    },
                    {
                        "type": "string",
                        "optional": true,
                        "field": "email"
                    }
                ],
                "optional": true,
                "name": "test.public.customers.Value",
                "field": "before"
            },
            {
                "type": "struct",
                "fields": [
                    {
                        "type": "int32",
                        "optional": false,
                        "field": "id"
                    },
                    {
                        "type": "string",
                        "optional": true,
                        "field": "first_name"
                    },
```

```
                    {
                        "type": "string",
                        "optional": true,
                        "field": "last_name"
                    },
                    {
                        "type": "string",
                        "optional": true,
                        "field": "email"
                    }
                ],
                "optional": true,
                "name": "test.public.customers.Value",
                "field": "after"
            },
            {
                "type": "struct",
                "fields": [
                    {
                        "type": "string",
                        "optional": false,
                        "field": "version"
                    },
                    {
                        "type": "string",
                        "optional": false,
                        "field": "connector"
                    },
                    {
                        "type": "string",
                        "optional": false,
                        "field": "name"
                    },
                    {
                        "type": "int64",
                        "optional": false,
                        "field": "ts_ms"
                    },
                    {
                        "type": "string",
                        "optional": true,
                        "name": "io.debezium.data.Enum",
                        "version": 1,
                        "parameters": {
                            "allowed": "true,last,false,incremental"
                        },
```

```
                        "default": "false",
                        "field": "snapshot"
                    },
                    {
                        "type": "string",
                        "optional": false,
                        "field": "db"
                    },
                    {
                        "type": "string",
                        "optional": true,
                        "field": "sequence"
                    },
                    {
                        "type": "string",
                        "optional": false,
                        "field": "schema"
                    },
                    {
                        "type": "string",
                        "optional": false,
                        "field": "table"
                    },
                    {
                        "type": "int64",
                        "optional": true,
                        "field": "txId"
                    },
                    {
                        "type": "int64",
                        "optional": true,
                        "field": "lsn"
                    },
                    {
                        "type": "int64",
                        "optional": true,
                        "field": "xmin"
                    }
                ],
                "optional": false,
                "name": "io.debezium.connector.postgresql.Source",
                "field": "source"
            },
            {
                "type": "string",
                "optional": false,
```

```
                "field": "op"
            },
            {
                "type": "int64",
                "optional": true,
                "field": "ts_ms"
            },
            {
                "type": "struct",
                "fields": [
                    {
                        "type": "string",
                        "optional": false,
                        "field": "id"
                    },
                    {
                        "type": "int64",
                        "optional": false,
                        "field": "total_order"
                    },
                    {
                        "type": "int64",
                        "optional": false,
                        "field": "data_collection_order"
                    }
                ],
                "optional": true,
                "name": "event.block",
                "version": 1,
                "field": "transaction"
            }
        ],
        "optional": false,
        "name": "test.public.customers.Envelope",
        "version": 1
    },
    "payload": {
        "before": null,
        "after": {
            "id": 1,
            "first_name": "a",
            "last_name": "a",
            "email": "a@a.com"
        },
        "source": {
            "version": "2.5.3.Final",
```

```
            "connector": "postgresql",
            "name": "test",
            "ts_ms": 1714283798721,
            "snapshot": "false",
            "db": "postgres",
            "sequence": "[\"22910216\",\"22910504\"]",
            "schema": "public",
            "table": "customers",
            "txId": 756,
            "lsn": 22910504,
            "xmin": null
        },
        "op": "c",
        "ts_ms": 1714283798790,
        "transaction": null
    }
}
```

Шаг 3: Настройка Selena

Создайте таблицу Primary Key в Selena с той же схемой, что и исходная таблица в PostgreSQL.

```
CREATE TABLE `customers` (
  `id` int(11) COMMENT "",
  `first_name` varchar(65533) NULL COMMENT "",
  `last_name` varchar(65533) NULL COMMENT "",
  `email` varchar(65533) NULL COMMENT ""
) ENGINE=OLAP
PRIMARY KEY(`id`)
DISTRIBUTED BY hash(id) buckets 1
PROPERTIES (
"bucket_size" = "4294967296",
"in_memory" = "false",
"enable_persistent_index" = "true",
"replicated_storage" = "true",
"fast_schema_evolution" = "true"
);
```

Шаг 4: Установка connector

1. Скачайте connector'ы и распакуйте пакеты в директорию **plugins**.

```
mkdir plugins
tar -zxvf debezium-debezium-connector-postgresql-2.5.3.zip -C plugins
tar -zxvf selena-kafka-connector-1.0.3.tar.gz -C plugins
```

Эта директория является значением элемента конфигурации `plugin.path` в **config/connect-standalone.properties**.

```
plugin.path=/path/to/kafka_2.13-3.7.0/plugins
```

2. Настройте source connector для PostgreSQL в **pg-source.properties**.

```
{
  "name": "inventory-connector",
  "config": {
    "connector.class":
"io.debezium.connector.postgresql.PostgresConnector",
    "plugin.name": "pgoutput",
    "database.hostname": "localhost",
    "database.port": "5432",
    "database.user": "postgres",
    "database.password": "",
    "database.dbname" : "postgres",
    "topic.prefix": "test"
  }
}
```

3. Настройте sink connector для Selena в **sr-sink.properties**.

```
{
    "name": "selena-kafka-connector",
    "config": {
        "connector.class":
"com.selena.connector.kafka.SelenaSinkConnector",
        "tasks.max": "1",
        "topics": "test.public.customers",
        "selena.http.url": "172.26.195.69:28030",
        "selena.database.name": "test",
        "selena.username": "root",
        "selena.password": "Selena@123",
        "sink.properties.strip_outer_array": "true",
        "connect.timeoutms": "3000",
        "selena.topic2table.map": "test.public.customers:customers",
        "transforms": "addfield,unwrap",
        "transforms.addfield.type":
"com.selena.connector.kafka.transforms.AddOpFieldForDebeziumRecord",
        "transforms.unwrap.type":
"io.debezium.transforms.ExtractNewRecordState",
        "transforms.unwrap.drop.tombstones": "true",
        "transforms.unwrap.delete.handling.mode": "rewrite"
    }
}
```

> **ПРИМЕЧАНИЕ**
>
> - Если таблица Selena не является таблицей Primary Key, вам не нужно указывать трансформацию `addfield`.
> - Трансформация unwrap предоставляется Debezium и используется для распаковки сложной структуры данных Debezium на основе типа операции. Для получения дополнительной информации см. New Record State Extraction.

4. Настройте Kafka Connect.

   Настройте следующие элементы конфигурации в файле конфигурации Kafka Connect **config/connect-standalone.properties**.

```
# Адреса брокеров Kafka. Несколько адресов брокеров Kafka должны быть
разделены запятыми (,).
# Обратите внимание, что в этом примере используется PLAINTEXT в
качестве протокола безопасности для доступа к кластеру Kafka.
# Если вы используете другой протокол безопасности для доступа к
кластеру Kafka, настройте соответствующую информацию в этой части.

bootstrap.servers=<kafka_broker_ip>:9092
offset.storage.file.filename=/tmp/connect.offsets
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter.schemas.enable=false

# Абсолютный путь к selena-kafka-connector после распаковки. Например:
plugin.path=/home/kafka-connect/selena-kafka-connector-1.0.3

# Параметры, которые управляют политикой сброса. Для получения
дополнительной информации см. раздел Примечания по использованию.
offset.flush.interval.ms=10000
bufferflush.maxbytes = xxx
bufferflush.intervalms = xxx
```

Для описания дополнительных параметров см. Running Kafka Connect.

Шаг 5: Запуск Kafka Connect в автономном режиме

Запустите Kafka Connect в автономном режиме для инициализации connector'ов.

```
bin/connect-standalone.sh config/connect-standalone.properties config/pg-
source.properties config/sr-sink.properties
```

Шаг 6: Проверка загрузки данных

Протестируйте следующие операции и убедитесь, что данные правильно загружены в Selena.

INSERT

- В PostgreSQL:

```
postgres=# insert into customers values (2,'b','b','b@b.com');
INSERT 0 1
postgres=# select * from customers;
 id | first_name | last_name |  email
----+------------+-----------+---------
  1 | a          | a         | a@a.com
  2 | b          | b         | b@b.com
(2 rows)
```

- В Selena:

```
MySQL [test]> select * from customers;
+------+------------+-----------+---------+
| id   | first_name | last_name | email   |
+------+------------+-----------+---------+
|    1 | a          | a         | a@a.com |
|    2 | b          | b         | b@b.com |
+------+------------+-----------+---------+
2 rows in set (0.01 sec)
```

UPDATE

- В PostgreSQL:

```
postgres=# update customers set email='c@c.com';
UPDATE 2
postgres=# select * from customers;
 id | first_name | last_name |  email
----+------------+-----------+---------
  1 | a          | a         | c@c.com
  2 | b          | b         | c@c.com
(2 rows)
```

- В Selena:

```
MySQL [test]> select * from customers;
+------+------------+-----------+---------+
| id   | first_name | last_name | email   |
+------+------------+-----------+---------+
|    1 | a          | a         | c@c.com |
|    2 | b          | b         | c@c.com |
+------+------------+-----------+---------+
2 rows in set (0.00 sec)
```

DELETE

- В PostgreSQL:

```
postgres=# delete from customers where id=1;
DELETE 1
postgres=# select * from customers;
 id | first_name | last_name |  email
----+------------+-----------+---------
  2 | b          | b         | c@c.com
(1 row)
```

- В Selena:

```
MySQL [test]> select * from customers;
+------+------------+-----------+---------+
| id   | first_name | last_name | email   |
+------+------------+-----------+---------+
|    2 | b          | b         | c@c.com |
+------+------------+-----------+---------+
1 row in set (0.00 sec)
```

3.5.2 Загрузка данных с помощью Routine Load

> **ПОДСКАЗКА**
>
> Попробуйте Routine Load в этом Кратком руководстве

В этом разделе описывается, как создать задачу Routine Load для потоковой передачи сообщений Kafka (событий) в Selena, а также знакомит вас с основными концепциями Routine Load.

Для непрерывной загрузки сообщений потока в Selena вы можете сохранить поток сообщений в topic Kafka и создать задачу Routine Load для потребления сообщений. Задача Routine Load сохраняется в Selena, генерирует серию подзадач загрузки для потребления сообщений из всех или части partition в topic и загружает сообщения в Selena.

Задача Routine Load поддерживает семантику exactly-once доставки, гарантируя, что данные, загруженные в Selena, не будут потеряны или дублированы.

Routine Load поддерживает преобразование данных при загрузке и поддерживает изменения данных, выполняемые операциями UPSERT и DELETE во время загрузки данных. Для получения дополнительной информации см. Преобразование данных при загрузке и Изменение данных через загрузку.

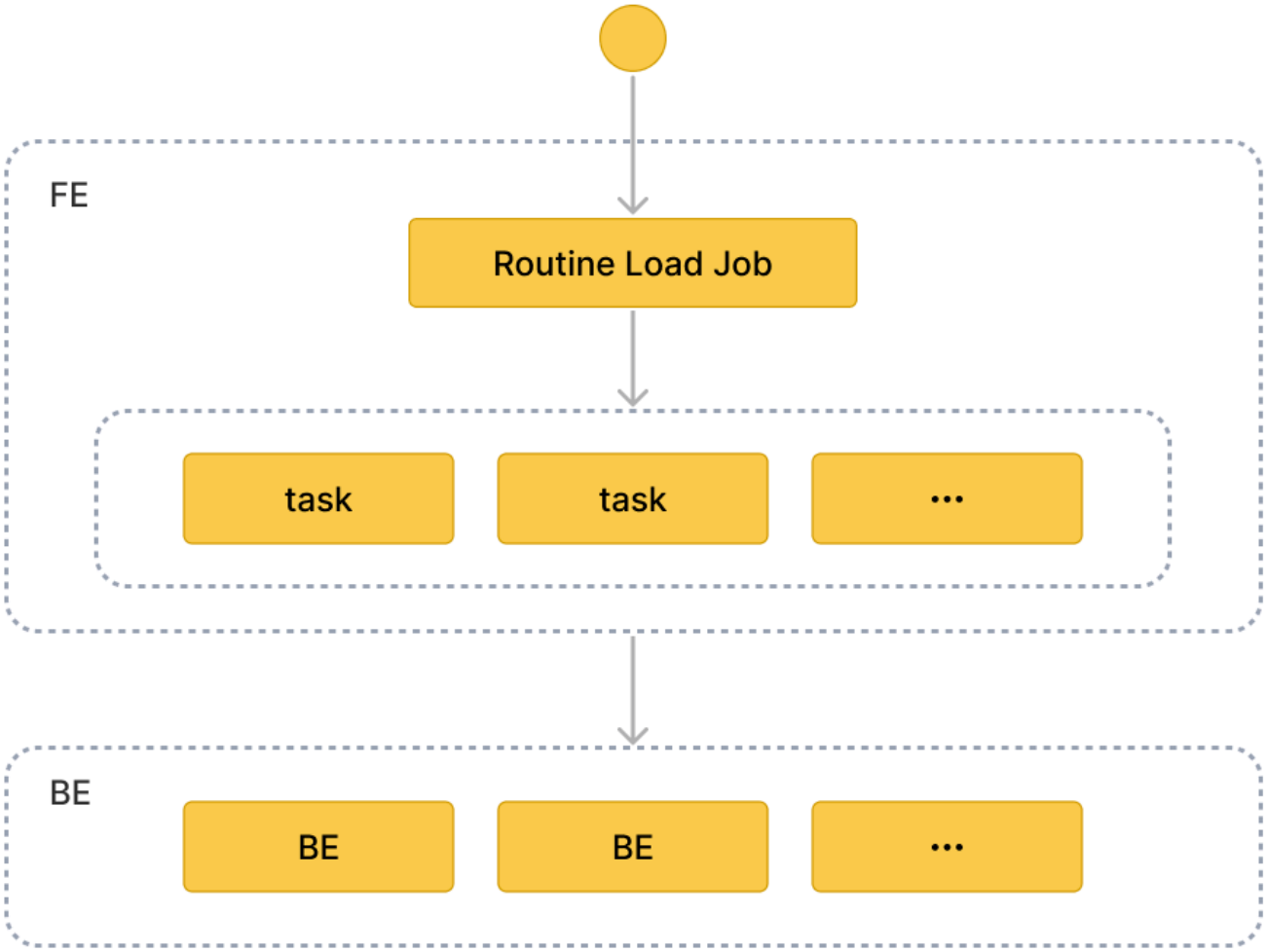
Вы можете загружать данные в таблицы Selena только как пользователь, имеющий привилегию INSERT на эти таблицы Selena. Если у вас нет привилегии INSERT, следуйте инструкциям в разделе GRANT, чтобы предоставить привилегию INSERT пользователю, которого вы используете для подключения к вашему cluster Selena. Синтаксис: `GRANT INSERT ON TABLE <table_name> IN DATABASE <database_name> TO { ROLE <role_name> | USER <user_identity>}`.

Поддерживаемые форматы данных

Routine Load в настоящее время поддерживает потребление данных в форматах CSV, JSON и Avro (поддерживается начиная с v1.5.2) из cluster Kafka.

> **ПРИМЕЧАНИЕ**
>
> Для данных CSV обратите внимание на следующие моменты:
>
> - Вы можете использовать строку UTF-8, такую как запятая (,), табуляция или вертикальная черта (|), длиной не более 50 байт в качестве текстового разделителя.
> - Null-значения обозначаются с помощью `\N`. Например, файл данных состоит из трёх столбцов, и запись из этого файла данных содержит данные в первом и третьем столбцах, но не содержит данных во втором столбце. В этой ситуации вам нужно использовать `\N` во втором столбце для обозначения null-значения. Это означает, что запись должна быть скомпилирована как `a,\N,b` вместо `a,,b`. `a,,b` означает, что второй столбец записи содержит пустую строку.

Основные концепции



Терминология

- **Задача загрузки (Load job)**

  Задача Routine Load — это долгосрочная задача. Пока её статус RUNNING, задача загрузки непрерывно генерирует одну или несколько параллельных подзадач загрузки, которые потребляют сообщения из topic cluster Kafka и загружают данные в Selena.

- **Подзадача загрузки (Load task)**

  Задача загрузки разделяется на несколько подзадач загрузки по определённым правилам. Подзадача загрузки — это базовая единица загрузки данных. Как отдельное событие, подзадача загрузки реализует механизм загрузки на основе Stream Load. Несколько подзадач загрузки одновременно потребляют сообщения из разных partition topic и загружают данные в Selena.

Рабочий процесс

1. **Создайте задачу Routine Load.** Для загрузки данных из Kafka вам нужно создать задачу Routine Load, выполнив оператор CREATE ROUTINE LOAD. FE анализирует оператор и создаёт задачу в соответствии с указанными вами свойствами.

2. **FE разделяет задачу на несколько подзадач загрузки.**

   FE разделяет задачу на несколько подзадач загрузки на основе определённых правил. Каждая подзадача загрузки является отдельной транзакцией. Правила разделения следующие:

   - FE вычисляет фактическое количество параллельных подзадач загрузки в соответствии с желаемым количеством параллелизма `desired_concurrent_number`, количеством partition в topic Kafka и количеством активных узлов BE.
   - FE разделяет задачу на подзадачи загрузки на основе вычисленного фактического количества параллелизма и размещает подзадачи в очереди задач.

   Каждый topic Kafka состоит из нескольких partition. Связь между partition topic и подзадачей загрузки следующая:

   - Partition однозначно назначается подзадаче загрузки, и все сообщения из partition потребляются подзадачей загрузки.
   - Подзадача загрузки может потреблять сообщения из одной или нескольких partition.
   - Все partition распределяются равномерно между подзадачами загрузки.

3. **Несколько подзадач загрузки выполняются параллельно для потребления сообщений из нескольких partition topic Kafka и загрузки данных в Selena**

   i. **FE планирует и отправляет подзадачи загрузки**: FE планирует подзадачи загрузки в очереди на регулярной основе и назначает их выбранным узлам Coordinator BE. Интервал между подзадачами загрузки определяется параметром конфигурации `max_batch_interval`. FE равномерно распределяет подзадачи загрузки по всем узлам BE. См. CREATE ROUTINE LOAD для получения дополнительной информации о `max_batch_interval`.

   ii. Coordinator BE запускает подзадачу загрузки, потребляет сообщения в partition, анализирует и фильтрует данные. Подзадача загрузки продолжается до тех пор,

пока не будет потреблено заранее определённое количество сообщений или не будет достигнут заранее определённый лимит времени. Размер пакета сообщений и лимит времени определяются в конфигурациях FE `max_routine_load_batch_size` и `routine_load_task_consume_second`. Для получения подробной информации см. Конфигурация FE. Затем Coordinator BE распределяет сообщения на Executor BE. Executor BE записывают сообщения на диски.

> **ПРИМЕЧАНИЕ**
>
> Selena поддерживает доступ к Kafka через протоколы безопасности, включая SASL_SSL, SAS_PLAINTEXT, SSL и PLAINTEXT. В этом разделе в качестве примера используется подключение к Kafka через PLAINTEXT. Если вам нужно подключиться к Kafka через другие протоколы безопасности, см. CREATE ROUTINE LOAD.

4. **FE генерирует новые подзадачи загрузки для непрерывной загрузки данных.** После того как Executor BE записали данные на диски, Coordinator BE сообщает результат подзадачи загрузки в FE. На основе результата FE затем генерирует новые подзадачи загрузки для непрерывной загрузки данных. Или FE повторяет неудачные подзадачи, чтобы гарантировать, что данные, загруженные в Selena, не будут потеряны или дублированы.

Создание задачи Routine Load

Следующие три примера описывают, как потреблять данные в форматах CSV, JSON и Avro из Kafka и загружать данные в Selena путём создания задачи Routine Load. Для получения подробного синтаксиса и описания параметров см. CREATE ROUTINE LOAD.

Загрузка данных в формате CSV

В этом разделе описывается, как создать задачу Routine Load для потребления данных в формате CSV из cluster Kafka и загрузки данных в Selena.

Подготовка набора данных

Предположим, что в topic `ordertest1` в cluster Kafka есть набор данных в формате CSV. Каждое сообщение в наборе данных включает шесть полей: идентификатор заказа, дата оплаты, имя клиента, национальность, пол и цена.

```
2020050802,2020-05-08,Johann Georg Faust,Deutschland,male,895
2020050802,2020-05-08,Julien Sorel,France,male,893
2020050803,2020-05-08,Dorian Grey,UK,male,1262
2020050901,2020-05-09,Anna Karenina",Russia,female,175
2020051001,2020-05-10,Tess Durbeyfield,US,female,986
2020051101,2020-05-11,Edogawa Conan,japan,male,8924
```

Создание таблицы

В соответствии с полями данных в формате CSV создайте таблицу `example_tbl1` в базе данных `example_db`. Следующий пример создаёт таблицу с 5 полями, исключая поле пола клиента из данных в формате CSV.

```
CREATE TABLE example_db.example_tbl1 (
    `order_id` bigint NOT NULL COMMENT "Order ID",
    `pay_dt` date NOT NULL COMMENT "Payment date",
    `customer_name` varchar(26) NULL COMMENT "Customer name",
    `nationality` varchar(26) NULL COMMENT "Nationality",
    `price`double NULL COMMENT "Price"
)
ENGINE=OLAP
DUPLICATE KEY (order_id,pay_dt)
DISTRIBUTED BY HASH(`order_id`);
```

> **ВНИМАНИЕ**
>
> Начиная с версии v1.5.2, Selena может автоматически устанавливать количество bucket (BUCKETS) при создании таблицы или добавлении partition. Вам больше не нужно вручную устанавливать количество bucket. Для получения подробной информации см. установка количества bucket.

Отправка задачи Routine Load

Выполните следующий оператор, чтобы отправить задачу Routine Load с именем `example_tbl1_ordertest1` для потребления сообщений из topic `ordertest1` и загрузки данных в таблицу `example_tbl1`. Подзадача загрузки потребляет сообщения с начального смещения в указанных partition topic.

```
CREATE ROUTINE LOAD example_db.example_tbl1_ordertest1 ON example_tbl1
COLUMNS TERMINATED BY ",",
COLUMNS (order_id, pay_dt, customer_name, nationality, temp_gender, price)
PROPERTIES
(
    "desired_concurrent_number" = "5"
)
FROM KAFKA
(
    "kafka_broker_list" = "<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "ordertest1",
    "kafka_partitions" = "0,1,2,3,4",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

После отправки задачи загрузки вы можете выполнить оператор SHOW ROUTINE LOAD, чтобы проверить статус задачи загрузки.

- **Имя задачи загрузки**

  На одной таблице может быть несколько задач загрузки. Поэтому мы рекомендуем называть задачу загрузки с соответствующим topic Kafka и временем отправки задачи загрузки. Это помогает различать задачи загрузки на каждой таблице.

- **Разделитель столбцов**

  Свойство `COLUMN TERMINATED BY` определяет разделитель столбцов данных в формате CSV. По умолчанию это `\t`.

- **Partition и смещение topic Kafka**

  Вы можете указать свойства `kafka_partitions` и `kafka_offsets`, чтобы указать partition и смещения для потребления сообщений. Например, если вы хотите, чтобы задача загрузки потребляла сообщения из partition Kafka `"0,1,2,3,4"` topic `ordertest1` все с начальными смещениями, вы можете указать свойства следующим образом: Если вы хотите, чтобы задача загрузки потребляла сообщения из partition Kafka `"0,1,2,3,4"` и вам нужно указать отдельное начальное смещение для каждой partition, вы можете настроить следующим образом:

```
"kafka_partitions" ="0,1,2,3,4",
"kafka_offsets" = "OFFSET_BEGINNING, OFFSET_END, 1000, 2000, 3000"
```

Вы также можете установить смещения по умолчанию для всех partition с помощью свойства `property.kafka_default_offsets`.

```
"kafka_partitions" ="0,1,2,3,4",
"property.kafka_default_offsets" = "OFFSET_BEGINNING"
```

Для получения подробной информации см. CREATE ROUTINE LOAD.

- **Сопоставление и преобразование данных**

Чтобы указать связь сопоставления и преобразования между данными в формате CSV и таблицей Selena, вам нужно использовать параметр `COLUMNS`.

Сопоставление данных:

- Selena извлекает столбцы из данных в формате CSV и сопоставляет их **последовательно** с полями, объявленными в параметре `COLUMNS`.
- Selena извлекает поля, объявленные в параметре `COLUMNS`, и сопоставляет их **по имени** со столбцами таблицы Selena.

Преобразование данных:

И поскольку пример исключает столбец пола клиента из данных в формате CSV, поле `temp_gender` в параметре `COLUMNS` используется в качестве заполнителя для этого поля. Остальные поля сопоставляются со столбцами таблицы Selena `example_tbl1` напрямую.

Для получения дополнительной информации о преобразовании данных см. Преобразование данных при загрузке.

> **ПРИМЕЧАНИЕ**
>
> Вам не нужно указывать параметр `COLUMNS`, если имена, количество и порядок столбцов в данных в формате CSV полностью соответствуют таковым в таблице

> Selena.

- **Параллелизм задач**

  Когда есть много partition topic Kafka и достаточно узлов BE, вы можете ускорить загрузку, увеличив параллелизм задач.

  Чтобы увеличить фактический параллелизм подзадач загрузки, вы можете увеличить желаемый параллелизм подзадач загрузки `desired_concurrent_number` при создании задачи routine load. Вы также можете установить динамический параметр конфигурации FE `max_routine_load_task_concurrent_num` (максимальный параллелизм подзадач загрузки по умолчанию) на большее значение. Для получения дополнительной информации о `max_routine_load_task_concurrent_num` см. Параметры конфигурации FE.

  Фактический параллелизм задач определяется минимальным значением среди количества активных узлов BE, количества предварительно указанных partition topic Kafka и значений `desired_concurrent_number` и `max_routine_load_task_concurrent_num`.

  В примере количество активных узлов BE равно `5`, количество предварительно указанных partition topic Kafka равно `5`, а значение `max_routine_load_task_concurrent_num` равно `5`. Чтобы увеличить фактический параллелизм подзадач загрузки, вы можете увеличить `desired_concurrent_number` со значения по умолчанию `3` до `5`.

  Для получения дополнительной информации о свойствах см. CREATE ROUTINE LOAD.

Загрузка данных в формате JSON

В этом разделе описывается, как создать задачу Routine Load для потребления данных в формате JSON из cluster Kafka и загрузки данных в Selena.

Подготовка набора данных

Предположим, что в topic `ordertest2` в cluster Kafka есть набор данных в формате JSON. Набор данных включает шесть ключей: идентификатор товара, имя клиента, национальность, время оплаты и цена. Кроме того, вы хотите преобразовать столбец времени оплаты в тип DATE и загрузить его в столбец `pay_dt` в таблице Selena.

```
{"commodity_id": "1", "customer_name": "Mark Twain", "country":
"US","pay_time": 1589191487,"price": 875}
{"commodity_id": "2", "customer_name": "Oscar Wilde", "country":
"UK","pay_time": 1589191487,"price": 895}
{"commodity_id": "3", "customer_name": "Antoine de Saint-
Exupéry","country": "France","pay_time": 1589191487,"price": 895}
```

> **ВНИМАНИЕ** Каждый JSON-объект в строке должен находиться в одном сообщении Kafka, иначе будет возвращена ошибка парсинга JSON.

Создание таблицы

В соответствии с ключами данных в формате JSON создайте таблицу `example_tbl2` в базе данных `example_db`.

```
CREATE TABLE `example_tbl2` (
    `commodity_id` varchar(26) NULL COMMENT "Commodity ID",
    `customer_name` varchar(26) NULL COMMENT "Customer name",
    `country` varchar(26) NULL COMMENT "Country",
    `pay_time` bigint(20) NULL COMMENT "Payment time",
    `pay_dt` date NULL COMMENT "Payment date",
    `price`double SUM NULL COMMENT "Price"
)
ENGINE=OLAP
AGGREGATE KEY(`commodity_id`,`customer_name`,`country`,`pay_time`,`pay_dt`)
DISTRIBUTED BY HASH(`commodity_id`);
```

> **ВНИМАНИЕ**
>
> Начиная с версии v1.5.2, Selena может автоматически устанавливать количество bucket (BUCKETS) при создании таблицы или добавлении partition. Вам больше не нужно вручную устанавливать количество bucket. Для получения подробной информации см. установка количества bucket.

Отправка задачи Routine Load

Выполните следующий оператор, чтобы отправить задачу Routine Load с именем `example_tbl2_ordertest2` для потребления сообщений из topic `ordertest2` и загрузки данных в таблицу `example_tbl2`. Подзадача загрузки потребляет сообщения с начального смещения в указанных partition topic.

```
CREATE ROUTINE LOAD example_db.example_tbl2_ordertest2 ON example_tbl2
COLUMNS(commodity_id, customer_name, country, pay_time, price,
pay_dt=from_unixtime(pay_time, '%Y%m%d'))
PROPERTIES
(
    "desired_concurrent_number" = "5",
    "format" = "json",
    "jsonpaths" = "
[\"$.commodity_id\",\"$.customer_name\",\"$.country\",\"$.pay_time\",\"$.pric
 )
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>",
    "kafka_topic" = "ordertest2",
    "kafka_partitions" ="0,1,2,3,4",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

После отправки задачи загрузки вы можете выполнить оператор SHOW ROUTINE LOAD, чтобы проверить статус задачи загрузки.

- **Формат данных**

  Вам нужно указать `"format" = "json"` в предложении `PROPERTIES`, чтобы определить, что формат данных — JSON.

- **Сопоставление и преобразование данных**

  Чтобы указать связь сопоставления и преобразования между данными в формате JSON и таблицей Selena, вам нужно указать параметр `COLUMNS` и свойство `jsonpaths`. Порядок полей, указанных в параметре `COLUMNS`, должен соответствовать порядку данных в формате JSON, а имена полей должны соответствовать именам в таблице Selena. Свойство `jsonpaths` используется для извлечения необходимых полей из данных JSON. Затем эти поля именуются свойством `COLUMNS`.

  Поскольку в примере необходимо преобразовать поле времени оплаты в тип данных DATE и загрузить данные в столбец `pay_dt` в таблице Selena, вам нужно использовать функцию from_unixtime. Остальные поля сопоставляются с полями таблицы `example_tbl2` напрямую.

Сопоставление данных:

- Selena извлекает ключи `name` и `code` из данных в формате JSON и сопоставляет их с ключами, объявленными в свойстве `jsonpaths`.
- Selena извлекает ключи, объявленные в свойстве `jsonpaths`, и сопоставляет их **последовательно** с полями, объявленными в параметре `COLUMNS`.
- Selena извлекает поля, объявленные в параметре `COLUMNS`, и сопоставляет их **по имени** со столбцами таблицы Selena.

Преобразование данных:

- Поскольку в примере необходимо преобразовать ключ `pay_time` в тип данных DATE и загрузить данные в столбец `pay_dt` в таблице Selena, вам нужно использовать функцию from_unixtime в параметре `COLUMNS`. Остальные поля сопоставляются с полями таблицы `example_tbl2` напрямую.
- И поскольку пример исключает столбец пола клиента из данных в формате JSON, поле `temp_gender` в параметре `COLUMNS` используется в качестве заполнителя для этого поля. Остальные поля сопоставляются со столбцами таблицы Selena `example_tbl1` напрямую.

  Для получения дополнительной информации о преобразовании данных см. Преобразование данных при загрузке.

  > **ПРИМЕЧАНИЕ**
  >
  > Вам не нужно указывать параметр `COLUMNS`, если имена и количество ключей в JSON-объекте полностью соответствуют именам и количеству полей в таблице Selena.

Загрузка данных в формате Avro

Начиная с версии v1.5.2, Selena поддерживает загрузку данных Avro с помощью Routine Load.

Подготовка набора данных

Схема Avro

1. Создайте следующий файл схемы Avro `avro_schema.avsc`:

```
{
    "type": "record",
    "name": "sensor_log",
    "fields" : [
        {"name": "id", "type": "long"},
        {"name": "name", "type": "string"},
        {"name": "checked", "type" : "boolean"},
        {"name": "data", "type": "double"},
        {"name": "sensor_type", "type": {"type": "enum", "name":
"sensor_type_enum", "symbols" : ["TEMPERATURE", "HUMIDITY", "AIR-
PRESSURE"]}}
    ]
}
```

2. Зарегистрируйте схему Avro в Schema Registry.

Данные Avro

Подготовьте данные Avro и отправьте их в topic Kafka `topic_0`.

Создание таблицы

В соответствии с полями данных Avro создайте таблицу `sensor_log` в целевой базе данных `example_db` в cluster Selena. Имена столбцов таблицы должны соответствовать именам полей в данных Avro. Для сопоставления типов данных между столбцами таблицы и полями данных Avro см. Сопоставление типов данных.

```
CREATE TABLE example_db.sensor_log (
    `id` bigint NOT NULL COMMENT "sensor id",
    `name` varchar(26) NOT NULL COMMENT "sensor name",
    `checked` boolean NOT NULL COMMENT "checked",
    `data` double NULL COMMENT "sensor data",
    `sensor_type` varchar(26) NOT NULL COMMENT "sensor type"
)
ENGINE=OLAP
DUPLICATE KEY (id)
DISTRIBUTED BY HASH(`id`);
```

> **ВНИМАНИЕ**
>
> Начиная с версии v1.5.2, Selena может автоматически устанавливать количество bucket (BUCKETS) при создании таблицы или добавлении partition. Вам больше не нужно вручную устанавливать количество bucket. Для получения подробной информации см. установка количества bucket.

Отправка задачи Routine Load

Выполните следующий оператор, чтобы отправить задачу Routine Load с именем `sensor_log_load_job` для потребления сообщений Avro из topic Kafka `topic_0` и загрузки данных в таблицу `sensor_log` в базе данных `sensor`. Задача загрузки потребляет сообщения с начального смещения в указанных partition topic.

```
CREATE ROUTINE LOAD example_db.sensor_log_load_job ON sensor_log
PROPERTIES
(
    "format" = "avro"
)
FROM KAFKA
(
    "kafka_broker_list" = "<kafka_broker1_ip>:<kafka_broker1_port>,
<kafka_broker2_ip>:<kafka_broker2_port>,...",
    "confluent.schema.registry.url" = "http://172.xx.xxx.xxx:8081",
    "kafka_topic" = "topic_0",
    "kafka_partitions" = "0,1,2,3,4,5",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

- Формат данных

  Вам нужно указать `"format = "avro"` в предложении `PROPERTIES`, чтобы определить, что формат данных — Avro.

- Schema Registry

  Вам нужно настроить `confluent.schema.registry.url`, чтобы указать URL Schema Registry, где зарегистрирована схема Avro. Selena получает схему Avro, используя этот URL. Формат следующий:

```
confluent.schema.registry.url = http[s]://[<schema-registry-api-key>:
<schema-registry-api-secret>@]<hostname|ip address>[:<port>]
```

- Сопоставление и преобразование данных

  Чтобы указать связь сопоставления и преобразования между данными в формате Avro и таблицей Selena, вам нужно указать параметр `COLUMNS` и свойство `jsonpaths`. Порядок полей, указанных в параметре `COLUMNS`, должен соответствовать порядку полей в свойстве `jsonpaths`, а имена полей должны соответствовать именам в таблице Selena. Свойство `jsonpaths` используется для извлечения необходимых полей из данных Avro. Затем эти поля именуются свойством `COLUMNS`.

  Для получения дополнительной информации о преобразовании данных см. Преобразование данных при загрузке.

  > ПРИМЕЧАНИЕ
  >
  > Вам не нужно указывать параметр `COLUMNS`, если имена и количество полей в записи Avro полностью соответствуют именам и количеству столбцов в таблице Selena.

После отправки задачи загрузки вы можете выполнить оператор SHOW ROUTINE LOAD, чтобы проверить статус задачи загрузки.

Сопоставление типов данных

Сопоставление типов данных между полями данных Avro, которые вы хотите загрузить, и столбцами таблицы Selena следующее:

Примитивные типы

Avro	Selena
nul	NULL
boolean	BOOLEAN
int	INT
long	BIGINT
float	FLOAT
double	DOUBLE
bytes	STRING

Avro	Selena
string	STRING

Сложные типы

Avro	Selena
record	Загружает всю RECORD или её подполя в Selena как JSON.
enums	STRING
arrays	ARRAY
maps	JSON
union(T, null)	NULLABLE(T)
fixed	STRING

Ограничения

- В настоящее время Selena не поддерживает эволюцию схемы.
- Каждое сообщение Kafka должно содержать только одну запись данных Avro.

Проверка задачи загрузки и подзадачи

Проверка задачи загрузки

Выполните оператор SHOW ROUTINE LOAD, чтобы проверить статус задачи загрузки `example_tbl2_ordertest2`. Selena возвращает состояние выполнения `State`, статистическую информацию (включая общее количество потреблённых строк и общее количество загруженных строк) `Statistics` и прогресс задачи загрузки `progress`.

Если состояние задачи загрузки автоматически изменилось на **PAUSED**, это возможно потому, что количество ошибочных строк превысило порог. Для получения подробных инструкций по установке этого порога см. CREATE ROUTINE LOAD. Вы можете проверить файлы `ReasonOfStateChanged` и `ErrorLogUrls`, чтобы выявить и устранить проблему. Устранив проблему, вы можете затем выполнить оператор RESUME ROUTINE LOAD, чтобы возобновить приостановленную (**PAUSED**) задачу загрузки.

Если состояние задачи загрузки — **CANCELLED**, возможно, задача загрузки столкнулась с исключением (например, таблица была удалена). Вы можете проверить файлы `ReasonOfStateChanged` и `ErrorLogUrls`, чтобы выявить и устранить проблему. Однако вы не можете возобновить отменённую (**CANCELLED**) задачу загрузки.

```
MySQL [example_db]> SHOW ROUTINE LOAD FOR example_tbl2_ordertest2 \G
*************************** 1. row ***************************
                  Id: 63013
                Name: example_tbl2_ordertest2
          CreateTime: 2022-08-10 17:09:00
           PauseTime: NULL
             EndTime: NULL
              DbName: default_cluster:example_db
           TableName: example_tbl2
               State: RUNNING
      DataSourceType: KAFKA
      CurrentTaskNum: 3
       JobProperties: {"partitions":"*","partial_update":"false","columnToCol
'%Y%m%d'),price","maxBatchIntervalS":"20","whereExpr":"*","dataFormat":"json"
[\"$.commodity_id\",\"$.customer_name\",\"$.country\",\"$.pay_time\",\"$.pric
DataSourceProperties: {"topic":"ordertest2","currentKafkaPartitions":"0,1,2,3
    CustomProperties: {"kafka_default_offsets":"OFFSET_BEGINNING"}
            Statistic: {"receivedBytes":230,"errorRows":0,"committedTaskNum":1
             Progress: {"0":"1","1":"OFFSET_ZERO","2":"OFFSET_ZERO","3":"OFFSE
ReasonOfStateChanged:
        ErrorLogUrls:
            OtherMsg:
```

> **ВНИМАНИЕ**
>
> Вы не можете проверить задачу загрузки, которая остановлена или ещё не запущена.

Проверка подзадачи загрузки

Выполните оператор SHOW ROUTINE LOAD TASK, чтобы проверить подзадачи загрузки задачи загрузки `example_tbl2_ordertest2`, такие как количество текущих выполняемых подзадач, partition topic Kafka, которые потребляются, и прогресс потребления `DataSourceProperties`, а также соответствующий узел Coordinator BE `BeId`.

```
MySQL [example_db]> SHOW ROUTINE LOAD TASK WHERE JobName =
"example_tbl2_ordertest2" \G
*************************** 1. row ***************************
              TaskId: 18c3a823-d73e-4a64-b9cb-b9eced026753
               TxnId: -1
           TxnStatus: UNKNOWN
               JobId: 63013
          CreateTime: 2022-08-10 17:09:05
   LastScheduledTime: 2022-08-10 17:47:27
    ExecuteStartTime: NULL
             Timeout: 60
                BeId: -1
DataSourceProperties: {"1":0,"4":0}
             Message: there is no new data in kafka, wait for 20 seconds to
schedule again
*************************** 2. row ***************************
              TaskId: f76c97ac-26aa-4b41-8194-a8ba2063eb00
               TxnId: -1
           TxnStatus: UNKNOWN
               JobId: 63013
          CreateTime: 2022-08-10 17:09:05
   LastScheduledTime: 2022-08-10 17:47:26
    ExecuteStartTime: NULL
             Timeout: 60
                BeId: -1
DataSourceProperties: {"2":0}
             Message: there is no new data in kafka, wait for 20 seconds to
schedule again
*************************** 3. row ***************************
              TaskId: 1a327a34-99f4-4f8d-8014-3cd38db99ec6
               TxnId: -1
           TxnStatus: UNKNOWN
               JobId: 63013
          CreateTime: 2022-08-10 17:09:26
   LastScheduledTime: 2022-08-10 17:47:27
    ExecuteStartTime: NULL
             Timeout: 60
                BeId: -1
DataSourceProperties: {"0":2,"3":0}
             Message: there is no new data in kafka, wait for 20 seconds to
schedule again
```

Приостановка задачи загрузки

Вы можете выполнить оператор PAUSE ROUTINE LOAD, чтобы приостановить задачу загрузки. Состояние задачи загрузки станет **PAUSED** после выполнения оператора. Однако она не остановлена. Вы можете выполнить оператор RESUME ROUTINE LOAD, чтобы возобновить её. Вы также можете проверить её статус с помощью оператора SHOW ROUTINE LOAD.

Следующий пример приостанавливает задачу загрузки `example_tbl2_ordertest2`:

```
PAUSE ROUTINE LOAD FOR example_tbl2_ordertest2;
```

Возобновление задачи загрузки

Вы можете выполнить оператор RESUME ROUTINE LOAD, чтобы возобновить приостановленную задачу загрузки. Состояние задачи загрузки временно станет **NEED_SCHEDULE** (поскольку задача загрузки перепланируется), а затем станет **RUNNING**. Вы можете проверить её статус с помощью оператора SHOW ROUTINE LOAD.

Следующий пример возобновляет приостановленную задачу загрузки `example_tbl2_ordertest2`:

```
RESUME ROUTINE LOAD FOR example_tbl2_ordertest2;
```

Изменение задачи загрузки

Перед изменением задачи загрузки вы должны приостановить её с помощью оператора PAUSE ROUTINE LOAD. Затем вы можете выполнить ALTER ROUTINE LOAD. После изменения вы можете выполнить оператор RESUME ROUTINE LOAD, чтобы возобновить её, и проверить её статус с помощью оператора SHOW ROUTINE LOAD.

Предположим, что количество активных узлов BE увеличилось до `6`, а partition topic Kafka для потребления — `"0,1,2,3,4,5,6,7"`. Если вы хотите увеличить фактический параллелизм подзадач загрузки, вы можете выполнить следующий оператор, чтобы увеличить количество желаемого параллелизма задач `desired_concurrent_number` до `6` (больше или равно количеству активных узлов BE) и указать partition topic Kafka и начальные смещения.

> **ПРИМЕЧАНИЕ**
>
> Поскольку фактический параллелизм задач определяется минимальным значением нескольких параметров, вы должны убедиться, что значение динамического параметра FE `max_routine_load_task_concurrent_num` больше или равно `6`.

```
ALTER ROUTINE LOAD FOR example_tbl2_ordertest2
PROPERTIES
(
    "desired_concurrent_number" = "6"
)
FROM kafka
(
    "kafka_partitions" = "0,1,2,3,4,5,6,7",
    "kafka_offsets" =
"OFFSET_BEGINNING,OFFSET_BEGINNING,OFFSET_BEGINNING,OFFSET_BEGINNING,OFFSET_E
);
```

Остановка задачи загрузки

Вы можете выполнить оператор STOP ROUTINE LOAD, чтобы остановить задачу загрузки. Состояние задачи загрузки станет **STOPPED** после выполнения оператора, и вы не сможете возобновить остановленную задачу загрузки. Вы не можете проверить статус остановленной задачи загрузки с помощью оператора SHOW ROUTINE LOAD.

Следующий пример останавливает задачу загрузки `example_tbl2_ordertest2`:

```
STOP ROUTINE LOAD FOR example_tbl2_ordertest2;
```

3.6 Загрузка из Apache Spark

3.6.1 Загрузка данных с помощью Spark connector (рекомендуется)

Selena предоставляет самостоятельно разработанный коннектор под названием Selena Connector for Apache Spark™ (сокращённо Spark connector), который помогает загружать данные в таблицу Selena с помощью Spark. Основной принцип заключается в накоплении данных и последующей загрузке всех данных сразу в Selena через STREAM LOAD. Spark connector реализован на основе Spark DataSource V2. DataSource можно создать с

помощью Spark DataFrames или Spark SQL. Поддерживаются как пакетный режим, так и режим Structured Streaming.

> **ВНИМАНИЕ**
>
> Только пользователи с привилегиями SELECT и INSERT на таблицу Selena могут загружать данные в эту таблицу. Вы можете следовать инструкциям в GRANT, чтобы предоставить эти привилегии пользователю.

Требования к версиям

Spark connector	Spark	Selena	Java	Scala
1.1.2	3.2, 3.3, 3.4, 3.5	2.5 и новее	8	2.12
1.1.1	3.2, 3.3 или 3.4	2.5 и новее	8	2.12
1.1.0	3.2, 3.3 или 3.4	2.5 и новее	8	2.12

> **ВНИМАНИЕ**
>
> - См. Обновление Spark connector для получения информации об изменениях поведения между различными версиями Spark connector.
> - Spark connector не предоставляет драйвер MySQL JDBC начиная с версии 1.1.1, и вам нужно вручную импортировать драйвер в classpath Spark. Вы можете найти драйвер на сайте MySQL или Maven Central.

Получение Spark connector

Вы можете получить JAR-файл Spark connector следующими способами:

- Напрямую загрузить скомпилированный JAR-файл Spark Connector.
- Добавить Spark connector как зависимость в ваш Maven-проект и затем загрузить JAR-файл.
- Скомпилировать исходный код Spark Connector в JAR-файл самостоятельно.

Формат именования JAR-файла Spark connector: `selena-spark-connector-${spark_version}_${scala_version}-${connector_version}.jar`.

Например, если вы установили Spark 3.2 и Scala 2.12 в вашей среде и хотите использовать Spark connector 1.1.0, вы можете использовать `selena-spark-connector-3.2_2.12-1.1.0.jar`.

> **ВНИМАНИЕ**
>
> В общем случае последняя версия Spark connector поддерживает совместимость только с тремя самыми последними версиями Spark.

Загрузка скомпилированного JAR-файла

Напрямую загрузите соответствующую версию JAR-файла Spark connector из Maven Central Repository.

Maven-зависимость

1. В файле `pom.xml` вашего Maven-проекта добавьте Spark connector как зависимость в следующем формате. Замените `spark_version`, `scala_version` и `connector_version` соответствующими версиями.

```
<dependency>
<groupId>com.selena</groupId>
<artifactId>selena-spark-connector-${spark_version}_${scala_version}
</artifactId>
<version>${connector_version}</version>
</dependency>
```

2. Например, если версия Spark в вашей среде — 3.2, версия Scala — 2.12, и вы выбираете Spark connector 1.1.0, вам нужно добавить следующую зависимость:

```
<dependency>
<groupId>com.selena</groupId>
<artifactId>selena-spark-connector-3.2_2.12</artifactId>
<version>1.1.0</version>
</dependency>
```

Компиляция самостоятельно

1. Загрузите пакет Spark connector.

2. Выполните следующую команду для компиляции исходного кода Spark connector в JAR-файл. Обратите внимание, что `spark_version` заменяется соответствующей версией Spark.

```
sh build.sh <spark_version>
```

Например, если версия Spark в вашей среде — 3.2, вам нужно выполнить следующую команду:

```
sh build.sh 3.2
```

3. Перейдите в каталог `target/`, чтобы найти JAR-файл Spark connector, например `selena-spark-connector-3.2_2.12-1.1.0-SNAPSHOT.jar`, сгенерированный при компиляции.

> **ПРИМЕЧАНИЕ**
>
> Имя Spark connector, который не выпущен официально, содержит суффикс `SNAPSHOT`.

Параметры

selena.fe.http.url

Обязательный: ДА
Значение по умолчанию: Нет
Описание: HTTP URL FE в вашем cluster Selena. Вы можете указать несколько URL, которые должны быть разделены запятой (,). Формат: `<fe_host1>:<fe_http_port1>,<fe_host2>:<fe_http_port2>`. Начиная с версии 1.1.1, вы также можете добавить префикс `http://` к URL, например `http://<fe_host1>:<fe_http_port1>,http://<fe_host2>:<fe_http_port2>`.

selena.fe.jdbc.url

Обязательный: ДА
Значение по умолчанию: Нет
Описание: Адрес, используемый для подключения к MySQL-серверу FE. Формат: `jdbc:mysql://<fe_host>:<fe_query_port>`.

selena.table.identifier

Обязательный: ДА
Значение по умолчанию: Нет
Описание: Имя таблицы Selena. Формат: `<database_name>.<table_name>`.

selena.user

Обязательный: ДА
Значение по умолчанию: Нет
Описание: Имя пользователя вашей учётной записи cluster Selena. Пользователю необходимы привилегии SELECT и INSERT на таблицу Selena.

selena.password

Обязательный: ДА
Значение по умолчанию: Нет
Описание: Пароль вашей учётной записи cluster Selena.

selena.write.label.prefix

Обязательный: НЕТ
Значение по умолчанию: spark-
Описание: Префикс метки, используемый Stream Load.

selena.write.enable.transaction-stream-load

Обязательный: НЕТ
Значение по умолчанию: TRUE
Описание: Использовать ли Stream Load transaction interface для загрузки данных. Требуется Selena v1.5.2 или новее. Эта функция может загружать больше данных в транзакции с меньшим использованием памяти и улучшить производительность.
ВНИМАНИЕ: Начиная с версии 1.1.1, этот параметр действует только когда значение `selena.write.max.retries` не положительное, поскольку Stream Load transaction interface не поддерживает повторы.

selena.write.buffer.size

Обязательный: НЕТ
Значение по умолчанию: 104857600
Описание: Максимальный размер данных, которые могут быть накоплены в памяти перед отправкой в Selena за один раз. Установка этого параметра на большее значение может улучшить производительность загрузки, но может увеличить задержку загрузки.

selena.write.buffer.rows

Обязательный: НЕТ
Значение по умолчанию: Integer.MAX_VALUE
Описание: Поддерживается начиная с версии 1.1.1. Максимальное количество строк, которые могут быть накоплены в памяти перед отправкой в Selena за один раз.

selena.write.flush.interval.ms

Обязательный: НЕТ
Значение по умолчанию: 300000
Описание: Интервал, с которым данные отправляются в Selena. Этот параметр используется для контроля задержки загрузки.

selena.write.max.retries

Обязательный: НЕТ
Значение по умолчанию: 3
Описание: Поддерживается начиная с версии 1.1.1. Количество повторов, которые коннектор выполняет для Stream Load для одного и того же пакета данных, если загрузка не удалась.
ВНИМАНИЕ: Поскольку Stream Load transaction interface не поддерживает повторы. Если этот параметр положительный, коннектор всегда использует интерфейс Stream Load и игнорирует значение `selena.write.enable.transaction-stream-load`.

selena.write.retry.interval.ms

Обязательный: НЕТ
Значение по умолчанию: 10000
Описание: Поддерживается начиная с версии 1.1.1. Интервал для повтора Stream Load для одного и того же пакета данных, если загрузка не удалась.

selena.columns

Обязательный: НЕТ
Значение по умолчанию: Нет
Описание: Столбец таблицы Selena, в который вы хотите загрузить данные. Вы можете указать несколько столбцов, которые должны быть разделены запятыми (,), например `"col0,col1,col2"`.

selena.column.types

Обязательный: НЕТ
Значение по умолчанию: Нет
Описание: Поддерживается начиная с версии 1.1.1. Настройка типов данных столбцов для Spark вместо использования значений по умолчанию, выведенных из таблицы Selena и сопоставления по умолчанию. Значение параметра — это схема в формате DDL, такая же как вывод Spark StructType#toDDL, например `col0 INT, col1 STRING, col2 BIGINT`. Обратите внимание, что вам нужно указать только столбцы, которые требуют настройки. Один из вариантов использования — загрузка данных в столбцы типа BITMAP или HLL.

selena.write.properties.*

Обязательный: НЕТ
Значение по умолчанию: Нет
Описание: Параметры, используемые для управления поведением Stream Load. Например, параметр `selena.write.properties.format` указывает формат данных для загрузки, например CSV или JSON. Для получения списка поддерживаемых параметров и их описаний см. STREAM LOAD.

selena.write.properties.format

Обязательный: НЕТ
Значение по умолчанию: CSV
Описание: Формат файла, на основе которого Spark connector преобразует каждый пакет данных перед отправкой данных в Selena. Допустимые значения: CSV и JSON.

selena.write.properties.row_delimiter

Обязательный: НЕТ
Значение по умолчанию: \n
Описание: Разделитель строк для данных в формате CSV.

selena.write.properties.column_separator

Обязательный: НЕТ
Значение по умолчанию: \t
Описание: Разделитель столбцов для данных в формате CSV.

selena.write.properties.partial_update

Обязательный: НЕТ
Значение по умолчанию: `FALSE`
Описание: Использовать ли частичные обновления. Допустимые значения: `TRUE` и `FALSE`. Значение по умолчанию: `FALSE`, что означает отключение этой функции.

selena.write.properties.partial_update_mode

Обязательный: НЕТ
Значение по умолчанию: `row`
Описание: Указывает режим для частичных обновлений. Допустимые значения: `row` и `column`.

- Значение `row` (по умолчанию) означает частичные обновления в строчном режиме, который более подходит для обновлений в реальном времени с множеством столбцов и небольшими пакетами.
- Значение `column` означает частичные обновления в колоночном режиме, который более подходит для пакетных обновлений с небольшим количеством столбцов и множеством строк. В таких сценариях включение колоночного режима обеспечивает более высокую скорость обновления. Например, в таблице со 100 столбцами, если обновляются только 10 столбцов (10% от общего количества) для всех строк, скорость обновления в колоночном режиме в 10 раз выше.

selena.write.num.partitions

Обязательный: НЕТ
Значение по умолчанию: Нет
Описание: Количество partition, в которые Spark может записывать данные параллельно. Когда объём данных небольшой, вы можете уменьшить количество partition, чтобы снизить параллелизм и частоту загрузки. Значение по умолчанию для этого параметра определяется Spark. Однако этот метод может вызвать накладные расходы на Spark Shuffle.

selena.write.partition.columns

Обязательный: НЕТ
Значение по умолчанию: Нет
Описание: Столбцы partition в Spark. Параметр действует только когда указан

`selena.write.num.partitions`. Если этот параметр не указан, все записываемые столбцы используются для партиционирования.

selena.timezone

Обязательный: НЕТ
Значение по умолчанию: Часовой пояс JVM по умолчанию
Описание: Поддерживается начиная с версии 1.1.1. Часовой пояс, используемый для преобразования Spark `TimestampType` в Selena `DATETIME`. По умолчанию — часовой пояс JVM, возвращаемый `ZoneId#systemDefault()`. Формат может быть названием часового пояса, например `Asia/Shanghai`, или смещением зоны, например `+08:00`.

Сопоставление типов данных между Spark и Selena

- Сопоставление типов данных по умолчанию следующее:

Тип данных Spark	Тип данных Selena
BooleanType	BOOLEAN
ByteType	TINYINT
ShortType	SMALLINT
IntegerType	INT
LongType	BIGINT
StringType	LARGEINT
FloatType	FLOAT
DoubleType	DOUBLE
DecimalType	DECIMAL
StringType	CHAR
StringType	VARCHAR
StringType	STRING
StringType	JSON
DateType	DATE
TimestampType	DATETIME
ArrayType	ARRAY **ПРИМЕЧАНИЕ:** **Поддерживается начиная с версии 1.1.1**. Для получения подробных шагов см. Загрузка данных в столбцы типа ARRAY.

- Вы также можете настроить сопоставление типов данных.

  Например, таблица Selena содержит столбцы BITMAP и HLL, но Spark не поддерживает эти два типа данных. Вам нужно настроить соответствующие типы данных в Spark. Для получения подробных шагов см. загрузку данных в столбцы BITMAP и HLL. **BITMAP и HLL поддерживаются начиная с версии 1.1.1**.

Обновление Spark connector

Обновление с версии 1.1.0 до 1.1.1

- Начиная с версии 1.1.1, Spark connector не предоставляет `mysql-connector-java`, который является официальным драйвером JDBC для MySQL, из-за ограничений лицензии GPL, используемой `mysql-connector-java`. Однако Spark connector всё ещё нуждается в драйвере MySQL JDBC для подключения к Selena для получения метаданных таблицы, поэтому вам нужно вручную добавить драйвер в classpath Spark. Вы можете найти драйвер на сайте MySQL или Maven Central.
- Начиная с версии 1.1.1, коннектор по умолчанию использует интерфейс Stream Load, а не Stream Load transaction interface, как в версии 1.1.0. Если вы всё ещё хотите использовать Stream Load transaction interface, вы можете установить опцию `selena.write.max.retries` в `0`. См. описание `selena.write.enable.transaction-stream-load` и `selena.write.max.retries` для подробностей.

Примеры

Следующие примеры показывают, как использовать Spark connector для загрузки данных в таблицу Selena с помощью Spark DataFrames или Spark SQL. Spark DataFrames поддерживает как пакетный режим, так и режим Structured Streaming.

Для получения дополнительных примеров см. Примеры Spark Connector.

Подготовка

Создание таблицы Selena

Создайте базу данных `test` и создайте таблицу с Primary Key `score_board`.

```
CREATE DATABASE `test`;

CREATE TABLE `test`.`score_board`
(
    `id` int(11) NOT NULL COMMENT "",
    `name` varchar(65533) NULL DEFAULT "" COMMENT "",
    `score` int(11) NOT NULL DEFAULT "0" COMMENT ""
)
ENGINE=OLAP
PRIMARY KEY(`id`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`id`);
```

Сетевая конфигурация

Убедитесь, что машина, на которой находится Spark, может получить доступ к узлам FE cluster Selena через `http_port` (по умолчанию: `8030`) и `query_port` (по умолчанию: `9030`), а к узлам BE через `be_http_port` (по умолчанию: `8040`).

Настройка среды Spark

Обратите внимание, что следующие примеры выполняются в Spark 3.2.4 и используют `spark-shell`, `pyspark` и `spark-sql`. Перед запуском примеров убедитесь, что JAR-файл Spark connector размещён в каталоге `$SPARK_HOME/jars`.

Загрузка данных с помощью Spark DataFrames

Следующие два примера объясняют, как загружать данные с помощью пакетного режима Spark DataFrames или режима Structured Streaming.

Пакетный режим

Создайте данные в памяти и загрузите данные в таблицу Selena.

1. Вы можете написать приложение Spark, используя Scala или Python.

Для Scala выполните следующий фрагмент кода в `spark-shell`:

```
// 1. Создайте DataFrame из последовательности.
val data = Seq((1, "selena", 100), (2, "spark", 100))
val df = data.toDF("id", "name", "score")

// 2. Запишите в Selena, настроив формат как "selena" и следующие опции.
// Вам нужно изменить опции в соответствии с вашей средой.
df.write.format("selena")
    .option("selena.fe.http.url", "127.0.0.1:8030")
    .option("selena.fe.jdbc.url", "jdbc:mysql://127.0.0.1:9030")
    .option("selena.table.identifier", "test.score_board")
    .option("selena.user", "root")
    .option("selena.password", "")
    .mode("append")
    .save()
```

Для Python выполните следующий фрагмент кода в `pyspark`:

```
from pyspark.sql import SparkSession

spark = SparkSession \
     .builder \
     .appName("Selena Example") \
     .getOrCreate()

 # 1. Создайте DataFrame из последовательности.
 data = [(1, "selena", 100), (2, "spark", 100)]
 df = spark.sparkContext.parallelize(data) \
         .toDF(["id", "name", "score"])

 # 2. Запишите в Selena, настроив формат как "selena" и следующие опции.
 # Вам нужно изменить опции в соответствии с вашей средой.
 df.write.format("selena") \
     .option("selena.fe.http.url", "127.0.0.1:8030") \
     .option("selena.fe.jdbc.url", "jdbc:mysql://127.0.0.1:9030") \
     .option("selena.table.identifier", "test.score_board") \
     .option("selena.user", "root") \
     .option("selena.password", "") \
     .mode("append") \
     .save()
```

2. Запросите данные в таблице Selena.

```
MySQL [test]> SELECT * FROM `score_board`;
+------+-----------+-------+
| id   | name      | score |
+------+-----------+-------+
|    1 | selena    |   100 |
|    2 | spark     |   100 |
+------+-----------+-------+
2 rows in set (0.00 sec)
```

Structured Streaming

Создайте потоковое чтение данных из CSV-файла и загрузите данные в таблицу Selena.

1. В каталоге `csv-data` создайте CSV-файл `test.csv` со следующими данными:

```
3,selena,100
4,spark,100
```

2. Вы можете написать приложение Spark, используя Scala или Python.

Для Scala выполните следующий фрагмент кода в `spark-shell`:

```
import org.apache.spark.sql.types.StructType

// 1. Создайте DataFrame из CSV.
val schema = (new StructType()
        .add("id", "integer")
        .add("name", "string")
        .add("score", "integer")
    )
val df = (spark.readStream
        .option("sep", ",")
        .schema(schema)
        .format("csv")
        // Замените на ваш путь к каталогу "csv-data".
        .load("/path/to/csv-data")
    )

// 2. Запишите в Selena, настроив формат как "selena" и следующие опции.
// Вам нужно изменить опции в соответствии с вашей средой.
val query = (df.writeStream.format("selena")
        .option("selena.fe.http.url", "127.0.0.1:8030")
        .option("selena.fe.jdbc.url", "jdbc:mysql://127.0.0.1:9030")
        .option("selena.table.identifier", "test.score_board")
        .option("selena.user", "root")
        .option("selena.password", "")
        // замените на ваш каталог контрольных точек
        .option("checkpointLocation", "/path/to/checkpoint")
        .outputMode("append")
        .start()
    )
```

Для Python выполните следующий фрагмент кода в `pyspark`:

```python
from pyspark.sql import SparkSession
from pyspark.sql.types import IntegerType, StringType, StructType,
StructField

spark = SparkSession \
     .builder \
     .appName("Selena SS Example") \
     .getOrCreate()

 # 1. Создайте DataFrame из CSV.
 schema = StructType([
         StructField("id", IntegerType()),
         StructField("name", StringType()),
         StructField("score", IntegerType())
     ])
 df = (
     spark.readStream
     .option("sep", ",")
     .schema(schema)
     .format("csv")
     # Замените на ваш путь к каталогу "csv-data".
     .load("/path/to/csv-data")
 )

 # 2. Запишите в Selena, настроив формат как "selena" и следующие опции.
 # Вам нужно изменить опции в соответствии с вашей средой.
 query = (
     df.writeStream.format("selena")
     .option("selena.fe.http.url", "127.0.0.1:8030")
     .option("selena.fe.jdbc.url", "jdbc:mysql://127.0.0.1:9030")
     .option("selena.table.identifier", "test.score_board")
     .option("selena.user", "root")
     .option("selena.password", "")
     # замените на ваш каталог контрольных точек
     .option("checkpointLocation", "/path/to/checkpoint")
     .outputMode("append")
     .start()
 )
```

3. Запросите данные в таблице Selena.

```
MySQL [test]> select * from score_board;
+------+-----------+-------+
| id   | name      | score |
+------+-----------+-------+
|    4 | spark     |   100 |
|    3 | selena    |   100 |
+------+-----------+-------+
2 rows in set (0.67 sec)
```

Загрузка данных с помощью Spark SQL

Следующий пример объясняет, как загружать данные с помощью Spark SQL, используя оператор `INSERT INTO` в Spark SQL CLI.

1. Выполните следующий SQL-оператор в `spark-sql`:

```
-- 1. Создайте таблицу, настроив источник данных как `selena` и
следующие опции.
-- Вам нужно изменить опции в соответствии с вашей средой.
CREATE TABLE `score_board`
USING selena
OPTIONS(
"selena.fe.http.url"="127.0.0.1:8030",
"selena.fe.jdbc.url"="jdbc:mysql://127.0.0.1:9030",
"selena.table.identifier"="test.score_board",
"selena.user"="root",
"selena.password"=""
);

-- 2. Вставьте две строки в таблицу.
INSERT INTO `score_board` VALUES (5, "selena", 100), (6, "spark", 100);
```

2. Запросите данные в таблице Selena.

```
MySQL [test]> select * from score_board;
+------+-----------+-------+
| id   | name      | score |
+------+-----------+-------+
|    6 | spark     |   100 |
|    5 | selena    |   100 |
+------+-----------+-------+
2 rows in set (0.00 sec)
```

Лучшие практики

Загрузка данных в таблицу с Primary Key

В этом разделе показано, как загружать данные в таблицу Selena с Primary Key для достижения частичных обновлений и условных обновлений. Вы можете посмотреть Изменение данных через загрузку для подробного введения в эти функции. Эти примеры используют Spark SQL.

Подготовка

Создайте базу данных `test` и создайте таблицу с Primary Key `score_board` в Selena.

```
CREATE DATABASE `test`;

CREATE TABLE `test`.`score_board`
(
    `id` int(11) NOT NULL COMMENT "",
    `name` varchar(65533) NULL DEFAULT "" COMMENT "",
    `score` int(11) NOT NULL DEFAULT "0" COMMENT ""
)
ENGINE=OLAP
PRIMARY KEY(`id`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`id`);
```

Частичные обновления

Этот пример покажет, как обновить данные только в столбце `name` через загрузку:

1. Вставьте начальные данные в таблицу Selena в клиенте MySQL.

```
mysql> INSERT INTO `score_board` VALUES (1, 'selena', 100), (2,
'spark', 100);

mysql> select * from score_board;
+------+-----------+-------+
| id   | name      | score |
+------+-----------+-------+
|    1 | selena    |   100 |
|    2 | spark     |   100 |
+------+-----------+-------+
2 rows in set (0.02 sec)
```

2. Создайте таблицу Spark `score_board` в клиенте Spark SQL.

   - Установите опцию `selena.write.properties.partial_update` в `true`, что указывает коннектору выполнять частичное обновление.
   - Установите опцию `selena.columns` в `"id,name"`, чтобы указать коннектору, какие столбцы записывать.

```
CREATE TABLE `score_board`
USING selena
OPTIONS(
    "selena.fe.http.url"="127.0.0.1:8030",
    "selena.fe.jdbc.url"="jdbc:mysql://127.0.0.1:9030",
    "selena.table.identifier"="test.score_board",
    "selena.user"="root",
    "selena.password"="",
    "selena.write.properties.partial_update"="true",
    "selena.columns"="id,name"
 );
```

3. Вставьте данные в таблицу в клиенте Spark SQL и обновите только столбец `name`.

```
INSERT INTO `score_board` VALUES (1, 'selena-update'), (2, 'spark-
update');
```

4. Запросите таблицу Selena в клиенте MySQL.

   Вы можете видеть, что изменились только значения для `name`, а значения для `score` не изменились.

```
mysql> select * from score_board;
+------+------------------+-------+
| id   | name             | score |
+------+------------------+-------+
|    1 | selena-update    |   100 |
|    2 | spark-update     |   100 |
+------+------------------+-------+
2 rows in set (0.02 sec)
```

Условные обновления

Этот пример покажет, как выполнять условные обновления в соответствии со значениями столбца `score`. Обновление для `id` вступает в силу только тогда, когда новое значение для `score` больше или равно старому значению.

1. Вставьте начальные данные в таблицу Selena в клиенте MySQL.

```
mysql> INSERT INTO `score_board` VALUES (1, 'selena', 100), (2,
'spark', 100);

mysql> select * from score_board;
+------+-----------+-------+
| id   | name      | score |
+------+-----------+-------+
|    1 | selena    |   100 |
|    2 | spark     |   100 |
+------+-----------+-------+
2 rows in set (0.02 sec)
```

2. Создайте таблицу Spark `score_board` следующим образом.

   - Установите опцию `selena.write.properties.merge_condition` в `score`, что указывает коннектору использовать столбец `score` в качестве условия.
   - Убедитесь, что Spark connector использует интерфейс Stream Load для загрузки данных, а не Stream Load transaction interface, поскольку последний не поддерживает эту функцию.

```
CREATE TABLE `score_board`
USING selena
OPTIONS(
    "selena.fe.http.url"="127.0.0.1:8030",
    "selena.fe.jdbc.url"="jdbc:mysql://127.0.0.1:9030",
    "selena.table.identifier"="test.score_board",
    "selena.user"="root",
    "selena.password"="",
    "selena.write.properties.merge_condition"="score"
 );
```

3. Вставьте данные в таблицу в клиенте Spark SQL и обновите строку с `id` равным 1 с меньшим значением score, а строку с `id` равным 2 с большим значением score.

```
INSERT INTO `score_board` VALUES (1, 'selena-update', 99), (2, 'spark-
update', 101);
```

4. Запросите таблицу Selena в клиенте MySQL.

   Вы можете видеть, что изменилась только строка с `id` равным 2, а строка с `id` равным 1 не изменилась.

```
mysql> select * from score_board;
+------+--------------+-------+
| id   | name         | score |
+------+--------------+-------+
|    1 | selena       |   100 |
|    2 | spark-update |   101 |
+------+--------------+-------+
2 rows in set (0.03 sec)
```

Загрузка данных в столбцы типа BITMAP

`BITMAP` часто используется для ускорения подсчёта уникальных значений (count distinct), например подсчёта UV, см. Использование Bitmap для точного Count Distinct. Здесь мы возьмём подсчёт UV в качестве примера, чтобы показать, как загружать данные в столбцы типа `BITMAP`. **`BITMAP` поддерживается начиная с версии 1.1.1**.

1. Создайте Aggregate-таблицу Selena.

В базе данных `test` создайте Aggregate-таблицу `page_uv`, где столбец `visit_users` определён как тип `BITMAP` и настроен с агрегатной функцией `BITMAP_UNION`.

```
CREATE TABLE `test`.`page_uv` (
  `page_id` INT NOT NULL COMMENT 'page ID',
  `visit_date` datetime NOT NULL COMMENT 'access time',
  `visit_users` BITMAP BITMAP_UNION NOT NULL COMMENT 'user ID'
) ENGINE=OLAP
AGGREGATE KEY(`page_id`, `visit_date`)
DISTRIBUTED BY HASH(`page_id`);
```

2. Создайте таблицу Spark.

   Схема таблицы Spark выводится из таблицы Selena, и Spark не поддерживает тип `BITMAP`. Поэтому вам нужно настроить соответствующий тип данных столбца в Spark, например как `BIGINT`, настроив опцию `"selena.column.types"="visit_users BIGINT"`. При использовании Stream Load для загрузки данных коннектор использует функцию `to_bitmap` для преобразования данных типа `BIGINT` в тип `BITMAP`.

   Выполните следующий DDL в `spark-sql`:

```
CREATE TABLE `page_uv`
USING selena
OPTIONS(
   "selena.fe.http.url"="127.0.0.1:8030",
   "selena.fe.jdbc.url"="jdbc:mysql://127.0.0.1:9030",
   "selena.table.identifier"="test.page_uv",
   "selena.user"="root",
   "selena.password"="",
   "selena.column.types"="visit_users BIGINT"
);
```

3. Загрузите данные в таблицу Selena.

   Выполните следующий DML в `spark-sql`:

```
INSERT INTO `page_uv` VALUES
   (1, CAST('2020-06-23 01:30:30' AS TIMESTAMP), 13),
   (1, CAST('2020-06-23 01:30:30' AS TIMESTAMP), 23),
   (1, CAST('2020-06-23 01:30:30' AS TIMESTAMP), 33),
   (1, CAST('2020-06-23 02:30:30' AS TIMESTAMP), 13),
   (2, CAST('2020-06-23 01:30:30' AS TIMESTAMP), 23);
```

4. Вычислите UV страниц из таблицы Selena.

```
MySQL [test]> SELECT `page_id`, COUNT(DISTINCT `visit_users`) FROM
`page_uv` GROUP BY `page_id`;
+---------+-----------------------------+
| page_id | count(DISTINCT visit_users) |
+---------+-----------------------------+
|       2 |                           1 |
|       1 |                           3 |
+---------+-----------------------------+
2 rows in set (0.01 sec)
```

> **ВНИМАНИЕ:**
>
> Коннектор использует функцию `to_bitmap` для преобразования данных типов `TINYINT`, `SMALLINT`, `INTEGER` и `BIGINT` в Spark в тип `BITMAP` в Selena, и использует функцию `bitmap_hash` или `bitmap_hash64` для других типов данных Spark.

Загрузка данных в столбцы типа HLL

`HLL` можно использовать для приблизительного подсчёта уникальных значений (count distinct), см. Использование HLL для приблизительного count distinct.

Здесь мы возьмём подсчёт UV в качестве примера, чтобы показать, как загружать данные в столбцы типа `HLL`. **`HLL` поддерживается начиная с версии 1.1.1**.

1. Создайте Aggregate-таблицу Selena.

   В базе данных `test` создайте Aggregate-таблицу `hll_uv`, где столбец `visit_users` определён как тип `HLL` и настроен с агрегатной функцией `HLL_UNION`.

```
CREATE TABLE `hll_uv` (
`page_id` INT NOT NULL COMMENT 'page ID',
`visit_date` datetime NOT NULL COMMENT 'access time',
`visit_users` HLL HLL_UNION NOT NULL COMMENT 'user ID'
) ENGINE=OLAP
AGGREGATE KEY(`page_id`, `visit_date`)
DISTRIBUTED BY HASH(`page_id`);
```

2. Создайте таблицу Spark.

   Схема таблицы Spark выводится из таблицы Selena, и Spark не поддерживает тип `HLL`. Поэтому вам нужно настроить соответствующий тип данных столбца в Spark, например как `BIGINT`, настроив опцию `"selena.column.types"="visit_users BIGINT"`. При использовании Stream Load для загрузки данных коннектор использует функцию `hll_hash` для преобразования данных типа `BIGINT` в тип `HLL`.

   Выполните следующий DDL в `spark-sql`:

```
CREATE TABLE `hll_uv`
USING selena
OPTIONS(
   "selena.fe.http.url"="127.0.0.1:8030",
   "selena.fe.jdbc.url"="jdbc:mysql://127.0.0.1:9030",
   "selena.table.identifier"="test.hll_uv",
   "selena.user"="root",
   "selena.password"="",
   "selena.column.types"="visit_users BIGINT"
);
```

3. Загрузите данные в таблицу Selena.

   Выполните следующий DML в `spark-sql`:

```
INSERT INTO `hll_uv` VALUES
   (3, CAST('2023-07-24 12:00:00' AS TIMESTAMP), 78),
   (4, CAST('2023-07-24 13:20:10' AS TIMESTAMP), 2),
   (3, CAST('2023-07-24 12:30:00' AS TIMESTAMP), 674);
```

4. Вычислите UV страниц из таблицы Selena.

```
MySQL [test]> SELECT `page_id`, COUNT(DISTINCT `visit_users`) FROM
`hll_uv` GROUP BY `page_id`;
+---------+----------------------------+
| page_id | count(DISTINCT visit_users) |
+---------+----------------------------+
|       4 |                          1 |
|       3 |                          2 |
+---------+----------------------------+
2 rows in set (0.01 sec)
```

Загрузка данных в столбцы типа ARRAY

Следующий пример объясняет, как загружать данные в столбцы типа `ARRAY`.

1. Создайте таблицу Selena.

   В базе данных `test` создайте таблицу с Primary Key `array_tbl`, которая включает один столбец `INT` и два столбца `ARRAY`.

```
CREATE TABLE `array_tbl` (
    `id` INT NOT NULL,
    `a0` ARRAY<STRING>,
    `a1` ARRAY<ARRAY<INT>>
 )
 ENGINE=OLAP
 PRIMARY KEY(`id`)
 DISTRIBUTED BY HASH(`id`)
 ;
```

2. Запишите данные в Selena.

   Поскольку некоторые версии Selena не предоставляют метаданные столбца `ARRAY`, коннектор не может вывести соответствующий тип данных Spark для этого столбца. Однако вы можете явно указать соответствующий тип данных Spark столбца в опции `selena.column.types`. В этом примере вы можете настроить опцию как `a0 ARRAY<STRING>,a1 ARRAY<ARRAY<INT>>`.

   Выполните следующий код в `spark-shell`:

```
 val data = Seq(
    |  (1, Seq("hello", "selena"), Seq(Seq(1, 2), Seq(3, 4))),
    |  (2, Seq("hello", "spark"), Seq(Seq(5, 6, 7), Seq(8, 9, 10)))
    | )
 val df = data.toDF("id", "a0", "a1")
 df.write
      .format("selena")
      .option("selena.fe.http.url", "127.0.0.1:8030")
      .option("selena.fe.jdbc.url", "jdbc:mysql://127.0.0.1:9030")
      .option("selena.table.identifier", "test.array_tbl")
      .option("selena.user", "root")
      .option("selena.password", "")
      .option("selena.column.types", "a0 ARRAY<STRING>,a1
ARRAY<ARRAY<INT>>")
      .mode("append")
      .save()
```

3. Запросите данные в таблице Selena.

```
MySQL [test]> SELECT * FROM `array_tbl`;
+------+-----------------------+--------------------+
| id   | a0                    | a1                 |
+------+-----------------------+--------------------+
|    1 | ["hello","selena"]    | [[1,2],[3,4]]      |
|    2 | ["hello","spark"]     | [[5,6,7],[8,9,10]] |
+------+-----------------------+--------------------+
2 rows in set (0.01 sec)
```

3.6.2 Массовая загрузка данных с помощью Spark Load

Этот метод загрузки использует внешние ресурсы Apache Spark™ для предварительной обработки импортируемых данных, что улучшает производительность импорта и экономит вычислительные ресурсы. Он в основном используется для **начальной миграции** и **импорта больших объёмов данных** в Selena (объём данных до уровня ТБ).

Spark load — это **асинхронный** метод импорта, который требует от пользователей создания задач импорта типа Spark через протокол MySQL и просмотра результатов импорта с помощью `SHOW LOAD`.

> **ВНИМАНИЕ**

> - Только пользователи с привилегией INSERT на таблицу Selena могут загружать данные в эту таблицу. Вы можете следовать инструкциям в GRANT, чтобы предоставить необходимую привилегию.
> - Spark Load нельзя использовать для загрузки данных в таблицу с Primary Key.

Объяснение терминологии

- **Spark ETL**: Отвечает главным образом за ETL данных в процессе импорта, включая построение глобального словаря (тип BITMAP), партиционирование, сортировку, агрегацию и т.д.
- **Broker**: Broker — это независимый процесс без состояния. Он инкапсулирует интерфейс файловой системы и предоставляет Selena возможность читать файлы из удалённых систем хранения.
- **Глобальный словарь**: Сохраняет структуру данных, которая сопоставляет данные от исходного значения к закодированному значению. Исходное значение может быть любого типа данных, в то время как закодированное значение является целым числом. Глобальный словарь в основном используется в сценариях, где предварительно вычисляется точный count distinct.

Основы

Пользователь отправляет задачу импорта типа Spark через MySQL-клиент; FE записывает метаданные и возвращает результат отправки.

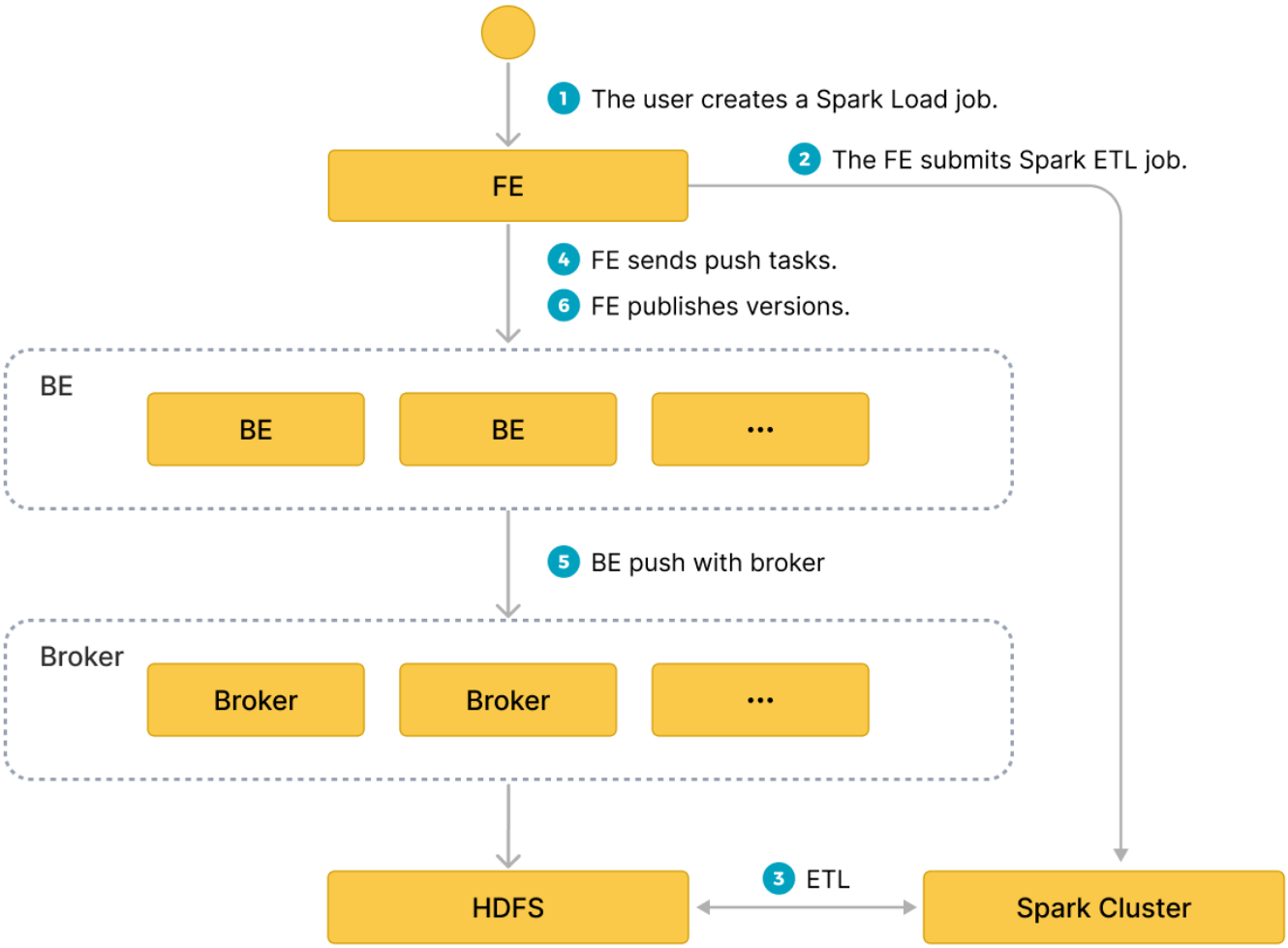
Выполнение задачи Spark load делится на следующие основные фазы.

1. Пользователь отправляет задачу Spark load в FE.
2. FE планирует отправку задачи ETL в cluster Apache Spark™ для выполнения.
3. Cluster Apache Spark™ выполняет задачу ETL, которая включает построение глобального словаря (тип BITMAP), партиционирование, сортировку, агрегацию и т.д.
4. После завершения задачи ETL FE получает путь к данным каждого предварительно обработанного среза и планирует соответствующие BE для выполнения задачи Push.
5. BE читает данные через процесс Broker из HDFS и конвертирует их в формат хранения Selena.

   > Если вы решите не использовать процесс Broker, BE читает данные из HDFS напрямую.

6. FE планирует эффективную версию и завершает задачу импорта.

Следующая диаграмма иллюстрирует основной поток Spark load.



Глобальный словарь

Применимые сценарии

В настоящее время столбец BITMAP в Selena реализован с использованием Roaringbitmap, который принимает только целые числа в качестве типа входных данных. Поэтому, если вы хотите реализовать предварительное вычисление для столбца BITMAP в процессе импорта, вам нужно преобразовать тип входных данных в целое число.

В существующем процессе импорта Selena структура данных глобального словаря реализована на основе таблицы Hive, которая сохраняет сопоставление от исходного значения к закодированному значению.

Процесс построения

1. Чтение данных из вышестоящего источника данных и генерация временной таблицы Hive с именем `hive-table`.
2. Извлечение значений дедуплицируемых полей из `hive-table` для генерации новой таблицы Hive с именем `distinct-value-table`.
3. Создание новой таблицы глобального словаря с именем `dict-table` с одним столбцом для исходных значений и одним столбцом для закодированных значений.
4. Left join между `distinct-value-table` и `dict-table`, а затем использование оконной функции для кодирования этого набора. Наконец, как исходное значение, так и закодированное значение дедуплицированного столбца записываются обратно в `dict-table`.
5. Join между `dict-table` и `hive-table` для завершения работы по замене исходного значения в `hive-table` на целочисленное закодированное значение.
6. `hive-table` будет прочитана при следующей предварительной обработке данных, а затем импортирована в Selena после вычисления.

Предварительная обработка данных

Базовый процесс предварительной обработки данных следующий:

1. Чтение данных из вышестоящего источника данных (файл HDFS или таблица Hive).
2. Выполнение сопоставления полей и вычисления для прочитанных данных, затем генерация `bucket-id` на основе информации о partition.
3. Генерация RollupTree на основе метаданных Rollup таблицы Selena.
4. Итерация по RollupTree и выполнение иерархических операций агрегации. Rollup следующей иерархии может быть вычислен из Rollup предыдущей иерархии.
5. Каждый раз после завершения вычисления агрегации данные разбиваются по bucket в соответствии с `bucket-id` и затем записываются в HDFS.
6. Последующий процесс Broker будет извлекать файлы из HDFS и импортировать их в узел BE Selena.

Базовые операции

Конфигурация cluster ETL

Apache Spark™ используется в качестве внешнего вычислительного ресурса в Selena для работы ETL. К Selena могут быть добавлены и другие внешние ресурсы, такие как Spark/GPU для запросов, HDFS/S3 для внешнего хранилища, MapReduce для ETL и т.д.

Поэтому мы вводим `Resource Management` для управления этими внешними ресурсами, используемыми Selena.

Перед отправкой задачи импорта Apache Spark™ настройте cluster Apache Spark™ для выполнения задач ETL. Синтаксис операции следующий:

```
-- создание ресурса Apache Spark™
CREATE EXTERNAL RESOURCE resource_name
PROPERTIES
(
 type = spark,
 spark_conf_key = spark_conf_value,
 working_dir = path,
 broker = broker_name,
 broker.property_key = property_value
);

-- удаление ресурса Apache Spark™
DROP RESOURCE resource_name;

-- просмотр ресурсов
SHOW RESOURCES
SHOW PROC "/resources";

-- привилегии
GRANT USAGE_PRIV ON RESOURCE resource_name TO user_identityGRANT USAGE_PRIV
ON RESOURCE resource_name TO ROLE role_name;
REVOKE USAGE_PRIV ON RESOURCE resource_name FROM user_identityREVOKE
USAGE_PRIV ON RESOURCE resource_name FROM ROLE role_name;
```

- Создание ресурса

Пример:

```
-- режим yarn cluster
CREATE EXTERNAL RESOURCE "spark0"
PROPERTIES
(
    "type" = "spark",
    "spark.master" = "yarn",
    "spark.submit.deployMode" = "cluster",
    "spark.jars" = "xxx.jar,yyy.jar",
    "spark.files" = "/tmp/aaa,/tmp/bbb",
    "spark.executor.memory" = "1g",
    "spark.yarn.queue" = "queue0",
    "spark.hadoop.yarn.resourcemanager.address" = "127.0.0.1:9999",
    "spark.hadoop.fs.defaultFS" = "hdfs://127.0.0.1:10000",
    "working_dir" = "hdfs://127.0.0.1:10000/tmp/selena",
    "broker" = "broker0",
    "broker.username" = "user0",
    "broker.password" = "password0"
);

-- режим yarn HA cluster
CREATE EXTERNAL RESOURCE "spark1"
PROPERTIES
(
    "type" = "spark",
    "spark.master" = "yarn",
    "spark.submit.deployMode" = "cluster",
    "spark.hadoop.yarn.resourcemanager.ha.enabled" = "true",
    "spark.hadoop.yarn.resourcemanager.ha.rm-ids" = "rm1,rm2",
    "spark.hadoop.yarn.resourcemanager.hostname.rm1" = "host1",
    "spark.hadoop.yarn.resourcemanager.hostname.rm2" = "host2",
    "spark.hadoop.fs.defaultFS" = "hdfs://127.0.0.1:10000",
    "working_dir" = "hdfs://127.0.0.1:10000/tmp/selena",
    "broker" = "broker1"
);
```

`resource-name` — это имя ресурса Apache Spark™, настроенного в Selena.

`PROPERTIES` включает параметры, относящиеся к ресурсу Apache Spark™, следующим образом:

> **Примечание**

> Для получения подробного описания PROPERTIES ресурса Apache Spark™ см. CREATE RESOURCE

- Параметры, связанные со Spark:
  - `type`: Тип ресурса, обязательный, в настоящее время поддерживается только `spark`.
  - `spark.master`: Обязательный, в настоящее время поддерживается только `yarn`.
    - `spark.submit.deployMode`: Режим развёртывания программы Apache Spark™, обязательный, в настоящее время поддерживаются как `cluster`, так и `client`.
    - `spark.hadoop.fs.defaultFS`: Обязательный, если master — yarn.
    - Параметры, связанные с yarn resource manager, обязательные.
      - один ResourceManager на одном узле `spark.hadoop.yarn.resourcemanager.address`: Адрес единственного resource manager.
      - ResourceManager HA

        > Вы можете выбрать указание hostname или адреса ResourceManager.

        - `spark.hadoop.yarn.resourcemanager.ha.enabled`: Включить HA resource manager, установите в `true`.
        - `spark.hadoop.yarn.resourcemanager.ha.rm-ids`: список логических идентификаторов resource manager.
        - `spark.hadoop.yarn.resourcemanager.hostname.rm-id`: Для каждого rm-id укажите hostname, соответствующий resource manager.
        - `spark.hadoop.yarn.resourcemanager.address.rm-id`: Для каждого rm-id укажите `host:port` для клиента для отправки задач.
- `*working_dir`: Каталог, используемый ETL. Обязательный, если Apache Spark™ используется в качестве ресурса ETL. Например: `hdfs://host:port/tmp/selena`.

- Параметры, связанные с Broker:
  - `broker`: Имя Broker. Обязательный, если Apache Spark™ используется в качестве ресурса ETL. Вам нужно заранее выполнить настройку с помощью команды `ALTER SYSTEM ADD BROKER`.
  - `broker.property_key`: Информация (например, информация для аутентификации), которая должна быть указана, когда процесс Broker читает

промежуточный файл, сгенерированный ETL.

Предостережение:

Выше приведено описание параметров для загрузки через процесс Broker. Если вы намерены загружать данные без процесса Broker, следует обратить внимание на следующее.

- Вам не нужно указывать `broker`.
- Если вам нужно настроить аутентификацию пользователя и HA для узлов NameNode, вам нужно настроить параметры в файле hdfs-site.xml в cluster HDFS, см. описания параметров в broker_properties. и вам нужно переместить файл **hdfs-site.xml** в **$FE_HOME/conf** для каждого FE и **$BE_HOME/conf** для каждого BE.

> Примечание
>
> Если файл HDFS доступен только определённому пользователю, вам всё ещё нужно указать имя пользователя HDFS в `broker.name` и пароль пользователя в `broker.password`.

- Просмотр ресурсов

Обычные учётные записи могут просматривать только ресурсы, к которым у них есть доступ `USAGE-PRIV`. Учётные записи root и admin могут просматривать все ресурсы.

- Разрешения ресурсов

Разрешения ресурсов управляются через `GRANT REVOKE`, который в настоящее время поддерживает только разрешения `USAGE-PRIV`. Вы можете предоставить разрешения `USAGE-PRIV` пользователю или роли.

```
-- Предоставить доступ к ресурсам spark0 пользователю user0
GRANT USAGE_PRIV ON RESOURCE "spark0" TO "user0"@"%";

-- Предоставить доступ к ресурсам spark0 роли role0
GRANT USAGE_PRIV ON RESOURCE "spark0" TO ROLE "role0";

-- Предоставить доступ ко всем ресурсам пользователю user0
GRANT USAGE_PRIV ON RESOURCE* TO "user0"@"%";

-- Предоставить доступ ко всем ресурсам роли role0
GRANT USAGE_PRIV ON RESOURCE* TO ROLE "role0";

-- Отозвать привилегии использования ресурсов spark0 у пользователя user0
REVOKE USAGE_PRIV ON RESOURCE "spark0" FROM "user0"@"%";
```

Конфигурация клиента Spark

Настройте клиент Spark для FE, чтобы последний мог отправлять задачи Spark, выполняя команду `spark-submit`. Рекомендуется использовать официальную версию Spark2 2.4.5 или выше (адрес загрузки spark). После загрузки используйте следующие шаги для завершения конфигурации.

- Настройка `SPARK-HOME`

Разместите клиент Spark в каталоге на той же машине, что и FE, и настройте `spark_home_default_dir` в файле конфигурации FE на этот каталог, который по умолчанию является путём `lib/spark2x` в корневом каталоге FE и не может быть пустым.

- **Настройка пакета зависимостей SPARK**

Для настройки пакета зависимостей заархивируйте все jar-файлы в папке jars под клиентом Spark и настройте элемент `spark_resource_path` в конфигурации FE на этот zip-файл. Если эта конфигурация пуста, FE попытается найти файл `lib/spark2x/jars/spark-2x.zip` в корневом каталоге FE. Если FE не найдёт его, будет выдана ошибка.

При отправке задачи Spark load заархивированные файлы зависимостей будут загружены в удалённый репозиторий. Путь репозитория по умолчанию находится в каталоге `working_dir/{cluster_id}` с именем `--spark-repository--{resource-name}`, что

означает, что ресурс в cluster соответствует удалённому репозиторию. Структура каталога приведена ниже:

```
---spark-repository--spark0/

   |---archive-1.0.0/

   |        |\---lib-990325d2c0d1d5e45bf675e54e44fb16-spark-dpp-1.0.0\-jar-with-dependencies.jar

   |        |\---lib-7670c29daf535efe3c9b923f778f61fc-spark-2x.zip

   |---archive-1.1.0/

   |        |\---lib-64d5696f99c379af2bee28c1c84271d5-spark-dpp-1.1.0\-jar-with-dependencies.jar

   |        |\---lib-1bbb74bb6b264a270bc7fca3e964160f-spark-2x.zip

   |---archive-1.2.0/

   |        |-...
```

Помимо зависимостей Spark (по умолчанию названных `spark-2x.zip`), FE также загружает зависимости DPP в удалённый репозиторий. Если все зависимости, отправленные Spark load, уже существуют в удалённом репозитории, то нет необходимости загружать зависимости снова, что экономит время на повторную загрузку большого количества файлов каждый раз.

Конфигурация клиента YARN

Настройте клиент yarn для FE, чтобы FE мог выполнять команды yarn для получения статуса запущенного приложения или его остановки. Рекомендуется использовать официальную версию Hadoop2 2.5.2 или выше (адрес загрузки hadoop). После загрузки используйте следующие шаги для завершения конфигурации:

- **Настройка пути к исполняемому файлу YARN**

Разместите загруженный клиент yarn в каталоге на той же машине, что и FE, и настройте элемент `yarn_client_path` в файле конфигурации FE на бинарный исполняемый файл

yarn, который по умолчанию является путём `lib/yarn-client/hadoop/bin/yarn` в корневом каталоге FE.

- **Настройка пути к файлу конфигурации, необходимому для генерации YARN (опционально)**

Когда FE через клиент yarn получает статус приложения или останавливает приложение, по умолчанию Selena генерирует файл конфигурации, необходимый для выполнения команды yarn, в пути `lib/yarn-config` корневого каталога FE. Этот путь можно изменить, настроив запись `yarn_config_dir` в файле конфигурации FE, которая в настоящее время включает `core-site.xml` и `yarn-site.xml`.

Создание задачи импорта

Синтаксис:

```
LOAD LABEL load_label
    (data_desc, ...)
WITH RESOURCE resource_name
[resource_properties]
[PROPERTIES (key1=value1, ... )]

* load_label:
    db_name.label_name

* data_desc:
    DATA INFILE ('file_path', ...)
    [NEGATIVE]
    INTO TABLE tbl_name
    [PARTITION (p1, p2)]
    [COLUMNS TERMINATED BY separator ]
    [(col1, ...)]
    [COLUMNS FROM PATH AS (col2, ...)]
    [SET (k1=f1(xx), k2=f2(xx))]
    [WHERE predicate]

    DATA FROM TABLE hive_external_tbl
    [NEGATIVE]
    INTO TABLE tbl_name
    [PARTITION (p1, p2)]
    [SET (k1=f1(xx), k2=f2(xx))]
    [WHERE predicate]

* resource_properties:
 (key2=value2, ...)
```

Пример 1: Случай, когда вышестоящий источник данных — HDFS

```
LOAD LABEL db1.label1
(
    DATA INFILE("hdfs://abc.com:8888/user/selena/test/ml/file1")
    INTO TABLE tbl1
    COLUMNS TERMINATED BY ","
    (tmp_c1,tmp_c2)
    SET
    (
        id=tmp_c2,
        name=tmp_c1
    ),
    DATA INFILE("hdfs://abc.com:8888/user/selena/test/ml/file2")
    INTO TABLE tbl2
    COLUMNS TERMINATED BY ","
    (col1, col2)
    where col1 > 1
)
WITH RESOURCE 'spark0'
(
    "spark.executor.memory" = "2g",
    "spark.shuffle.compress" = "true"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Пример 2: Случай, когда вышестоящий источник данных — Hive.

- Шаг 1: Создайте новый ресурс hive

```
CREATE EXTERNAL RESOURCE hive0
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:8080"
);
```

- Шаг 2: Создайте новую внешнюю таблицу hive

```
CREATE EXTERNAL TABLE hive_t1
(
    k1 INT,
    K2 SMALLINT,
    k3 varchar(50),
    uuid varchar(100)
)
ENGINE=hive
PROPERTIES
(
    "resource" = "hive0",
    "database" = "tmp",
    "table" = "t1"
);
```

- Шаг 3: Отправьте команду загрузки, требуя, чтобы столбцы в импортируемой таблице Selena существовали во внешней таблице hive.

```
LOAD LABEL db1.label1
(
    DATA FROM TABLE hive_t1
    INTO TABLE tbl1
    SET
    (
        uuid=bitmap_dict(uuid)
    )
)
WITH RESOURCE 'spark0'
(
    "spark.executor.memory" = "2g",
    "spark.shuffle.compress" = "true"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Введение в параметры в Spark load:

- **Label**

Label задачи импорта. Каждая задача импорта имеет Label, который уникален в пределах базы данных, следуя тем же правилам, что и broker load.

- **Параметры класса описания данных**

В настоящее время поддерживаемые источники данных — CSV и таблица Hive. Остальные правила такие же, как и для broker load.

- **Параметры задачи импорта**

Параметры задачи импорта относятся к параметрам, принадлежащим разделу `opt_properties` оператора импорта. Эти параметры применяются ко всей задаче импорта. Правила такие же, как и для broker load.

- **Параметры ресурса Spark**

Ресурсы Spark должны быть заранее настроены в Selena, и пользователям необходимо предоставить разрешения USAGE-PRIV, прежде чем они смогут применять ресурсы к Spark load. Параметры ресурса Spark можно установить, когда у пользователя есть временная потребность, например, добавление ресурсов для задачи и изменение конфигураций Spark. Настройка действует только для этой задачи и не влияет на существующие конфигурации в cluster Selena.

```
WITH RESOURCE 'spark0'
(
    "spark.driver.memory" = "1g",
    "spark.executor.memory" = "3g"
)
```

- **Импорт, когда источник данных — Hive**

В настоящее время для использования таблицы Hive в процессе импорта вам нужно создать внешнюю таблицу типа `Hive`, а затем указать её имя при отправке команды импорта.

- **Процесс импорта для построения глобального словаря**

В команде загрузки вы можете указать необходимые поля для построения глобального словаря в следующем формате: `имя поля Selena=bitmap_dict(имя поля таблицы`

`hive)`. Обратите внимание, что в настоящее время **глобальный словарь поддерживается только когда вышестоящий источник данных — таблица Hive**.

- **Загрузка данных бинарного типа**

Начиная с версии v1.5.2, Spark Load поддерживает функцию bitmap_from_binary, которая может конвертировать бинарные данные в данные bitmap. Если тип столбца таблицы Hive или файла HDFS — binary, а соответствующий столбец в таблице Selena — агрегатный столбец типа bitmap, вы можете указать поля в команде загрузки в следующем формате: `имя поля Selena=bitmap_from_binary(имя поля таблицы Hive)`. Это устраняет необходимость построения глобального словаря.

Просмотр задач импорта

Импорт Spark load является асинхронным, как и broker load. Пользователь должен записать label задачи импорта и использовать его в команде `SHOW LOAD` для просмотра результатов импорта. Команда просмотра импорта общая для всех методов импорта. Пример следующий.

Обратитесь к Broker Load для получения подробного объяснения возвращаемых параметров. Различия следующие.

```
mysql> show load order by createtime desc limit 1\G
*************************** 1. row ***************************
  JobId: 76391
  Label: label1
  State: FINISHED
 Progress: ETL:100%; LOAD:100%
  Type: SPARK
 EtlInfo: unselected.rows=4; dpp.abnorm.ALL=15; dpp.norm.ALL=28133376
 TaskInfo: cluster:cluster0; timeout(s):10800; max_filter_ratio:5.0E-5
 ErrorMsg: N/A
 CreateTime: 2019-07-27 11:46:42
 EtlStartTime: 2019-07-27 11:46:44
 EtlFinishTime: 2019-07-27 11:49:44
 LoadStartTime: 2019-07-27 11:49:44
LoadFinishTime: 2019-07-27 11:50:16
  URL: http://1.1.1.1:8089/proxy/application_1586619723848_0035/
 JobDetails:
{"ScannedRows":28133395,"TaskNumber":1,"FileNumber":1,"FileSize":200000}
```

- **State**

Текущая стадия импортируемой задачи. PENDING: Задача зафиксирована. ETL: Spark ETL зафиксирован. LOADING: FE планирует BE для выполнения операции push. FINISHED: Push завершён, и версия эффективна.

Есть две финальные стадии задачи импорта — `CANCELLED` и `FINISHED`, обе указывают на завершение задачи загрузки. `CANCELLED` указывает на неудачу импорта, а `FINISHED` указывает на успех импорта.

- **Progress**

Описание прогресса задачи импорта. Есть два типа прогресса — ETL и LOAD, которые соответствуют двум фазам процесса импорта, ETL и LOADING.

- Диапазон прогресса для LOAD — 0~100%.

`LOAD прогресс = количество текущих завершённых tablet всех replica импортов / общее количество tablet этой задачи импорта * 100%`.

- Если все таблицы были импортированы, прогресс LOAD составляет 99% и меняется на 100%, когда импорт входит в финальную фазу проверки.

- Прогресс импорта не линейный. Если прогресс не меняется в течение некоторого времени, это не означает, что импорт не выполняется.

- **Type**

Тип задачи импорта. SPARK для Spark load.

- **CreateTime/EtlStartTime/EtlFinishTime/LoadStartTime/LoadFinishTime**

Эти значения представляют время создания импорта, время начала фазы ETL, время завершения фазы ETL, время начала фазы LOADING и время завершения всей задачи импорта.

- **JobDetails**

Отображает подробный статус выполнения задачи, включая количество импортированных файлов, общий размер (в байтах), количество подзадач, количество обработанных

исходных строк и т.д. Например:

```
{"ScannedRows":139264,"TaskNumber":1,"FileNumber":1,"FileSize":940754064}
```

- **URL**

Вы можете скопировать ввод в ваш браузер для доступа к веб-интерфейсу соответствующего приложения.

Просмотр логов отправки Apache Spark™ Launcher

Иногда пользователям нужно просмотреть подробные логи, сгенерированные во время отправки задачи Apache Spark™. По умолчанию логи сохраняются в пути `log/spark_launcher_log` в корневом каталоге FE с именем `spark-launcher-{load-job-id}-{label}.log`. Логи сохраняются в этом каталоге в течение определённого времени и будут стёрты при очистке информации об импорте в метаданных FE. Время хранения по умолчанию — 3 дня.

Отмена импорта

Когда статус задачи Spark load не `CANCELLED` или `FINISHED`, она может быть отменена вручную пользователем путём указания Label задачи импорта.

Связанные системные конфигурации

Конфигурация FE: Следующая конфигурация является конфигурацией системного уровня для Spark load, которая применяется ко всем задачам импорта Spark load. Значения конфигурации можно настроить главным образом путём изменения `fe.conf`.

- enable-spark-load: Включить Spark load и создание ресурсов со значением по умолчанию false.
- spark-load-default-timeout-second: Таймаут по умолчанию для задачи — 259200 секунд (3 дня).
- spark-home-default-dir: Путь к клиенту Spark (`fe/lib/spark2x`).
- spark-resource-path: Путь к упакованному файлу зависимостей Spark (по умолчанию пустой).
- spark-launcher-log-dir: Каталог, где хранится лог отправки клиента Spark (`fe/log/spark-launcher-log`).

- yarn-client-path: Путь к бинарному исполняемому файлу yarn (`fe/lib/yarn-client/hadoop/bin/yarn`).
- yarn-config-dir: Путь к файлу конфигурации Yarn (`fe/lib/yarn-config`).

Лучшие практики

Наиболее подходящий сценарий для использования Spark load — когда исходные данные находятся в файловой системе (HDFS), а объём данных составляет от десятков ГБ до уровня ТБ. Используйте Stream Load или Broker Load для меньших объёмов данных.

Для полного примера импорта Spark load обратитесь к демо на github: https://github.com/Selena/demo/blob/master/docs/03_sparkLoad2Selena.md

FAQ

- `Error: When running with master 'yarn' either HADOOP-CONF-DIR or YARN-CONF-DIR must be set in the environment.`

Использование Spark Load без настройки переменной окружения `HADOOP-CONF-DIR` в `spark-env.sh` клиента Spark.

- `Error: Cannot run program "xxx/bin/spark-submit": error=2, No such file or directory`

Элемент конфигурации `spark_home_default_dir` не указывает корневой каталог клиента Spark при использовании Spark Load.

- `Error: File xxx/jars/spark-2x.zip does not exist.`

Элемент конфигурации `spark-resource-path` не указывает на упакованный zip-файл при использовании Spark load.

- `Error: yarn client does not exist in path: xxx/yarn-client/hadoop/bin/yarn`

Элемент конфигурации yarn-client-path не указывает исполняемый файл yarn при использовании Spark load.

- `ERROR: Cannot execute hadoop-yarn/bin/... /libexec/yarn-config.sh`

При использовании Hadoop с CDH вам нужно настроить переменную окружения `HADOOP_LIBEXEC_DIR`. Поскольку `hadoop-yarn` и каталоги hadoop разные, каталог

`libexec` по умолчанию будет искать `hadoop-yarn/bin/... /libexec`, в то время как `libexec` находится в каталоге hadoop. Команда ```yarn application status`` для получения статуса задачи Spark выдала ошибку, что привело к сбою задачи импорта.

3.7 Загрузка данных с помощью INSERT

В этой теме описывается, как загружать данные в Selena с помощью SQL оператора - INSERT.

Подобно MySQL и многим другим системам управления базами данных, Selena поддерживает загрузку данных во внутреннюю таблицу с помощью INSERT. Вы можете вставить одну или несколько строк напрямую с помощью предложения VALUES для тестирования функции или DEMO. Вы также можете вставить данные, определенные результатами запроса, во внутреннюю таблицу из внешней таблицы. Начиная с Selena v1.5.2, вы можете напрямую загружать данные из файлов в облачном хранилище, используя команду INSERT и табличную функцию FILES().

Selena v1.5.2 дополнительно поддерживает перезапись данных в таблицу с помощью INSERT OVERWRITE. Оператор INSERT OVERWRITE интегрирует следующие операции для реализации функции перезаписи:

1. Создает временные partitions в соответствии с partitions, которые хранят исходные данные.
2. Вставляет данные во временные partitions.
3. Меняет местами исходные partitions с временными partitions.

> **ПРИМЕЧАНИЕ**
>
> Если вам необходимо проверить данные перед их перезаписью, вместо использования INSERT OVERWRITE вы можете следовать вышеуказанным процедурам, чтобы перезаписать ваши данные и проверить их перед заменой partitions.

Начиная с v1.5.2, Selena поддерживает новую семантику - Dynamic Overwrite для INSERT OVERWRITE с partitioned таблицами. Для получения дополнительной информации см. Dynamic Overwrite.

3.7.1 Предостережения

- Вы можете отменить синхронную транзакцию INSERT только нажав клавиши **Ctrl** и **C** из вашего клиента MySQL.
- Вы можете отправить асинхронную задачу INSERT, используя SUBMIT TASK.
- Что касается текущей версии Selena, транзакция INSERT завершается ошибкой по умолчанию, если данные любых строк не соответствуют схеме таблицы. Например, транзакция INSERT завершается ошибкой, если длина поля в любой строке превышает ограничение длины для соответствующего поля в таблице. Вы можете установить переменную сеанса `enable_insert_strict` в `false`, чтобы разрешить транзакции продолжать работу, отфильтровывая строки, которые не соответствуют таблице.
- Если вы часто выполняете оператор INSERT для загрузки небольших батчей данных в Selena, создается избыточное количество версий данных. Это серьезно влияет на производительность запросов. Рекомендуется в производственной среде не загружать данные с помощью команды INSERT слишком часто или использовать её как рутинную процедуру для ежедневной загрузки данных. Если ваше приложение или аналитический сценарий требует решений для загрузки потоковых данных или небольших батчей данных отдельно, рекомендуется использовать Apache Kafka® в качестве источника данных и загружать данные через Routine Load.
- Если вы выполняете оператор INSERT OVERWRITE, Selena создает временные partitions для partitions, которые хранят исходные данные, вставляет новые данные во временные partitions и меняет местами исходные partitions с временными partitions. Все эти операции выполняются на узле FE Leader. Следовательно, если узел FE Leader выходит из строя во время выполнения команды INSERT OVERWRITE, вся транзакция загрузки завершится ошибкой, и временные partitions будут усечены.

3.7.2 Подготовка

Проверка привилегий

Вы можете загружать данные в таблицы Selena только как пользователь, имеющий привилегию INSERT на эти таблицы Selena. Если у вас нет привилегии INSERT, следуйте инструкциям в разделе GRANT, чтобы предоставить привилегию INSERT пользователю, которого вы используете для подключения к вашему cluster Selena. Синтаксис: `GRANT INSERT ON TABLE <table_name> IN DATABASE <database_name> TO { ROLE <role_name> | USER <user_identity>}`.

Создание объектов

Создайте базу данных с именем `load_test`, создайте таблицу `insert_wiki_edit` в качестве целевой таблицы и таблицу `source_wiki_edit` в качестве исходной таблицы.

> **ПРИМЕЧАНИЕ**
>
> Примеры, продемонстрированные в этой теме, основаны на таблице `insert_wiki_edit` и таблице `source_wiki_edit`. Если вы предпочитаете работать со своими собственными таблицами и данными, вы можете пропустить подготовку и перейти к следующему шагу.

```
CREATE DATABASE IF NOT EXISTS load_test;
USE load_test;
CREATE TABLE insert_wiki_edit
(
    event_time      DATETIME,
    channel         VARCHAR(32)      DEFAULT '',
    user            VARCHAR(128)     DEFAULT '',
    is_anonymous    TINYINT          DEFAULT '0',
    is_minor        TINYINT          DEFAULT '0',
    is_new          TINYINT          DEFAULT '0',
    is_robot        TINYINT          DEFAULT '0',
    is_unpatrolled  TINYINT          DEFAULT '0',
    delta           INT              DEFAULT '0',
    added           INT              DEFAULT '0',
    deleted         INT              DEFAULT '0'
)
DUPLICATE KEY(
    event_time,
    channel,
    user,
    is_anonymous,
    is_minor,
    is_new,
    is_robot,
    is_unpatrolled
)
PARTITION BY RANGE(event_time)(
    PARTITION p06 VALUES LESS THAN ('2015-09-12 06:00:00'),
    PARTITION p12 VALUES LESS THAN ('2015-09-12 12:00:00'),
    PARTITION p18 VALUES LESS THAN ('2015-09-12 18:00:00'),
    PARTITION p24 VALUES LESS THAN ('2015-09-13 00:00:00')
)
DISTRIBUTED BY HASH(user);

CREATE TABLE source_wiki_edit
(
    event_time      DATETIME,
    channel         VARCHAR(32)      DEFAULT '',
    user            VARCHAR(128)     DEFAULT '',
    is_anonymous    TINYINT          DEFAULT '0',
    is_minor        TINYINT          DEFAULT '0',
    is_new          TINYINT          DEFAULT '0',
    is_robot        TINYINT          DEFAULT '0',
    is_unpatrolled  TINYINT          DEFAULT '0',
    delta           INT              DEFAULT '0',
```

```
    added           INT                 DEFAULT '0',
    deleted         INT                 DEFAULT '0'
)
DUPLICATE KEY(
    event_time,
    channel,user,
    is_anonymous,
    is_minor,
    is_new,
    is_robot,
    is_unpatrolled
)
PARTITION BY RANGE(event_time)(
    PARTITION p06 VALUES LESS THAN ('2015-09-12 06:00:00'),
    PARTITION p12 VALUES LESS THAN ('2015-09-12 12:00:00'),
    PARTITION p18 VALUES LESS THAN ('2015-09-12 18:00:00'),
    PARTITION p24 VALUES LESS THAN ('2015-09-13 00:00:00')
)
DISTRIBUTED BY HASH(user);
```

> **ВАЖНО**
>
> Начиная с v1.5.2, Selena может автоматически устанавливать количество buckets (BUCKETS) при создании таблицы или добавлении partition. Вам больше не нужно вручную устанавливать количество buckets. Подробную информацию см. в установка количества buckets.

3.7.3 Вставка данных через INSERT INTO VALUES

Вы можете добавить одну или несколько строк в определенную таблицу, используя команду INSERT INTO VALUES. Несколько строк разделяются запятой (,). Подробные инструкции и справочные параметры см. в SQL Reference - INSERT.

> **ВНИМАНИЕ**
>
> Вставка данных через INSERT INTO VALUES применяется только к ситуации, когда вам необходимо проверить DEMO с небольшим набором данных. Это не рекомендуется для массового тестирования или производственной среды. Для загрузки массы данных в Selena см. Варианты загрузки для других вариантов, которые подходят для ваших сценариев.

Следующий пример вставляет две строки в таблицу источника данных `source_wiki_edit` с меткой `insert_load_wikipedia`. Label - это уникальный идентификационный label для каждой транзакции загрузки данных в базе данных.

```
INSERT INTO source_wiki_edit
WITH LABEL insert_load_wikipedia
VALUES
    ("2015-09-12 00:00:00","#en.wikipedia","AustinFF",0,0,0,0,0,21,5,0),
    ("2015-09-12 00:00:00","#ca.wikipedia","helloSR",0,1,0,1,0,3,23,0);
```

3.7.4 Вставка данных через INSERT INTO SELECT

Вы можете загрузить результат запроса к исходной таблице данных в целевую таблицу с помощью команды INSERT INTO SELECT. Команда INSERT INTO SELECT выполняет операции ETL на данных из исходной таблицы данных и загружает данные во внутреннюю таблицу в Selena. Источником данных может быть одна или несколько внутренних или внешних таблиц или даже файлы данных в облачном хранилище. Целевая таблица ДОЛЖНА быть внутренней таблицей в Selena. Подробные инструкции и справочные параметры см. в SQL Reference - INSERT.

Вставка данных из внутренней или внешней таблицы во внутреннюю таблицу

> **ПРИМЕЧАНИЕ**
>
> Вставка данных из внешней таблицы идентична вставке данных из внутренней таблицы. Для простоты мы демонстрируем только вставку данных из внутренней таблицы в следующих примерах.

- Следующий пример вставляет данные из исходной таблицы в целевую таблицу `insert_wiki_edit`.

```
INSERT INTO insert_wiki_edit
WITH LABEL insert_load_wikipedia_1
SELECT * FROM source_wiki_edit;
```

- Следующий пример вставляет данные из исходной таблицы в partitions `p06` и `p12` целевой таблицы `insert_wiki_edit`. Если partition не указан, данные будут

вставлены во все partitions. В противном случае данные будут вставлены только в указанные partition(s).

```
INSERT INTO insert_wiki_edit PARTITION(p06, p12)
WITH LABEL insert_load_wikipedia_2
SELECT * FROM source_wiki_edit;
```

Выполните запрос к целевой таблице, чтобы убедиться, что в ней есть данные.

```
MySQL > select * from insert_wiki_edit;
+---------------------+---------------+----------+--------------+----------
+--------+----------+----------------+-------+-------+---------+
| event_time          | channel       | user     | is_anonymous | is_minor
| is_new | is_robot | is_unpatrolled | delta | added | deleted |
+---------------------+---------------+----------+--------------+----------
+--------+----------+----------------+-------+-------+---------+
| 2015-09-12 00:00:00 | #en.wikipedia | AustinFF |            0 |        0
|      0 |        0 |              0 |    21 |     5 |       0 |
| 2015-09-12 00:00:00 | #ca.wikipedia | helloSR  |            0 |        1
|      0 |        1 |              0 |     3 |    23 |       0 |
+---------------------+---------------+----------+--------------+----------
+--------+----------+----------------+-------+-------+---------+
2 rows in set (0.00 sec)
```

Если вы усечете partitions `p06` и `p12`, данные не будут возвращены в запросе.

```
MySQL > TRUNCATE TABLE insert_wiki_edit PARTITION(p06, p12);
Query OK, 0 rows affected (0.01 sec)

MySQL > select * from insert_wiki_edit;
Empty set (0.00 sec)
```

- Следующий пример вставляет столбцы `event_time` и `channel` из исходной таблицы в целевую таблицу `insert_wiki_edit`. Значения по умолчанию используются в столбцах, которые здесь не указаны.

```
INSERT INTO insert_wiki_edit
WITH LABEL insert_load_wikipedia_3
(
    event_time,
    channel
)
SELECT event_time, channel FROM source_wiki_edit;
```

> **ПРИМЕЧАНИЕ**
>
> Начиная с v1.5.2, указание списка столбцов в операторе INSERT INTO для таблицы Primary Key будет выполнять Partial Updates (вместо Full Upsert в более ранних версиях). Если список столбцов не указан, система будет выполнять Full Upsert.

Вставка данных непосредственно из файлов во внешнем источнике с использованием FILES()

Начиная с v1.5.2, Selena поддерживает прямую загрузку данных из файлов в облачном хранилище, используя команду INSERT и функцию FILES(), поэтому вам не нужно сначала создавать внешний catalog или файловую внешнюю таблицу. Кроме того, FILES() может автоматически определять схему таблицы файлов, значительно упрощая процесс загрузки данных.

Следующий пример вставляет строки данных из файла Parquet **parquet/insert_wiki_edit_append.parquet** в bucket AWS S3 `inserttest` в таблицу `insert_wiki_edit`:

```
INSERT INTO insert_wiki_edit
    SELECT * FROM FILES(
        "path" = "s3://inserttest/parquet/insert_wiki_edit_append.parquet",
        "format" = "parquet",
        "aws.s3.access_key" = "XXXXXXXXXX",
        "aws.s3.secret_key" = "YYYYYYYYYY",
        "aws.s3.region" = "us-west-2"
);
```

3.7.5 Перезапись данных через INSERT OVERWRITE VALUES

Вы можете перезаписать определенную таблицу одной или несколькими строками, используя команду INSERT OVERWRITE VALUES. Несколько строк разделяются запятой (,). Подробные инструкции и справочные параметры см. в SQL Reference - INSERT.

> **ВНИМАНИЕ**
>
> Перезапись данных через INSERT OVERWRITE VALUES применяется только к ситуации, когда вам необходимо проверить DEMO с небольшим набором данных. Это не рекомендуется для массового тестирования или производственной среды. Для загрузки массы данных в Selena см. Варианты загрузки для других вариантов, которые подходят для ваших сценариев.

Выполните запрос к исходной таблице и целевой таблице, чтобы убедиться, что в них есть данные.

```
MySQL > SELECT * FROM source_wiki_edit;
+---------------------+---------------+----------+--------------+----------
+--------+----------+----------------+-------+-------+---------+
| event_time          | channel       | user     | is_anonymous | is_minor
| is_new | is_robot | is_unpatrolled | delta | added | deleted |
+---------------------+---------------+----------+--------------+----------
+--------+----------+----------------+-------+-------+---------+
| 2015-09-12 00:00:00 | #ca.wikipedia | helloSR  |            0 |        1
|      0 |        1 |              0 |     3 |    23 |       0 |
| 2015-09-12 00:00:00 | #en.wikipedia | AustinFF |            0 |        0
|      0 |        0 |              0 |    21 |     5 |       0 |
+---------------------+---------------+----------+--------------+----------
+--------+----------+----------------+-------+-------+---------+
2 rows in set (0.02 sec)

MySQL > SELECT * FROM insert_wiki_edit;
+---------------------+---------------+----------+--------------+----------
+--------+----------+----------------+-------+-------+---------+
| event_time          | channel       | user     | is_anonymous | is_minor
| is_new | is_robot | is_unpatrolled | delta | added | deleted |
+---------------------+---------------+----------+--------------+----------
+--------+----------+----------------+-------+-------+---------+
| 2015-09-12 00:00:00 | #ca.wikipedia | helloSR  |            0 |        1
|      0 |        1 |              0 |     3 |    23 |       0 |
| 2015-09-12 00:00:00 | #en.wikipedia | AustinFF |            0 |        0
|      0 |        0 |              0 |    21 |     5 |       0 |
+---------------------+---------------+----------+--------------+----------
+--------+----------+----------------+-------+-------+---------+
2 rows in set (0.01 sec)
```

Следующий пример перезаписывает исходную таблицу `source_wiki_edit` двумя новыми строками.

```
INSERT OVERWRITE source_wiki_edit
WITH LABEL insert_load_wikipedia_ow
VALUES
    ("2015-09-12
00:00:00","#cn.wikipedia","GELongstreet",0,0,0,0,0,36,36,0),
    ("2015-09-12 00:00:00","#fr.wikipedia","PereBot",0,1,0,1,0,17,17,0);
```

3.7.6 Перезапись данных через INSERT OVERWRITE SELECT

Вы можете перезаписать таблицу результатом запроса к исходной таблице данных с помощью команды INSERT OVERWRITE SELECT. Оператор INSERT OVERWRITE SELECT выполняет операции ETL на данных из одной или нескольких внутренних или внешних таблиц и перезаписывает внутреннюю таблицу данными. Подробные инструкции и справочные параметры см. в SQL Reference - INSERT.

> **ПРИМЕЧАНИЕ**
>
> Загрузка данных из внешней таблицы идентична загрузке данных из внутренней таблицы. Для простоты мы демонстрируем только перезапись целевой таблицы данными из внутренней таблицы в следующих примерах.

Выполните запрос к исходной таблице и целевой таблице, чтобы убедиться, что они содержат разные строки данных.

```
MySQL > SELECT * FROM source_wiki_edit;
+---------------------+---------------+--------------+--------------+------
----+--------+----------+----------------+-------+-------+---------+
| event_time          | channel       | user         | is_anonymous |
is_minor | is_new | is_robot | is_unpatrolled | delta | added | deleted |
+---------------------+---------------+--------------+--------------+------
----+--------+----------+----------------+-------+-------+---------+
| 2015-09-12 00:00:00 | #cn.wikipedia | GELongstreet |            0 |
0 |      0 |        0 |              0 |    36 |    36 |       0 |
| 2015-09-12 00:00:00 | #fr.wikipedia | PereBot      |            0 |
1 |      0 |        1 |              0 |    17 |    17 |       0 |
+---------------------+---------------+--------------+--------------+------
----+--------+----------+----------------+-------+-------+---------+
2 rows in set (0.02 sec)

MySQL > SELECT * FROM insert_wiki_edit;
+---------------------+---------------+----------+--------------+----------
+--------+----------+----------------+-------+-------+---------+
| event_time          | channel       | user     | is_anonymous | is_minor
| is_new | is_robot | is_unpatrolled | delta | added | deleted |
+---------------------+---------------+----------+--------------+----------
+--------+----------+----------------+-------+-------+---------+
| 2015-09-12 00:00:00 | #en.wikipedia | AustinFF |            0 |        0
|      0 |        0 |              0 |    21 |     5 |       0 |
| 2015-09-12 00:00:00 | #ca.wikipedia | helloSR  |            0 |        1
|      0 |        1 |              0 |     3 |    23 |       0 |
+---------------------+---------------+----------+--------------+----------
+--------+----------+----------------+-------+-------+---------+
2 rows in set (0.01 sec)
```

- Следующий пример перезаписывает таблицу `insert_wiki_edit` данными из исходной таблицы.

```
INSERT OVERWRITE insert_wiki_edit
WITH LABEL insert_load_wikipedia_ow_1
SELECT * FROM source_wiki_edit;
```

- Следующий пример перезаписывает partitions `p06` и `p12` таблицы `insert_wiki_edit` данными из исходной таблицы.

```
INSERT OVERWRITE insert_wiki_edit PARTITION(p06, p12)
WITH LABEL insert_load_wikipedia_ow_2
SELECT * FROM source_wiki_edit;
```

Выполните запрос к целевой таблице, чтобы убедиться, что в ней есть данные.

```
MySQL > select * from insert_wiki_edit;
+---------------------+---------------+--------------+--------------+------
----+--------+----------+----------------+-------+-------+---------+
| event_time          | channel       | user         | is_anonymous |
is_minor | is_new | is_robot | is_unpatrolled | delta | added | deleted |
+---------------------+---------------+--------------+--------------+------
----+--------+----------+----------------+-------+-------+---------+
| 2015-09-12 00:00:00 | #fr.wikipedia | PereBot      |            0 |
1 |      0 |        1 |              0 |    17 |    17 |       0 |
| 2015-09-12 00:00:00 | #cn.wikipedia | GELongstreet |            0 |
0 |      0 |        0 |              0 |    36 |    36 |       0 |
+---------------------+---------------+--------------+--------------+------
----+--------+----------+----------------+-------+-------+---------+
2 rows in set (0.01 sec)
```

Если вы усечете partitions `p06` и `p12`, данные не будут возвращены в запросе.

```
MySQL > TRUNCATE TABLE insert_wiki_edit PARTITION(p06, p12);
Query OK, 0 rows affected (0.01 sec)

MySQL > select * from insert_wiki_edit;
Empty set (0.00 sec)
```

ПРИМЕЧАНИЕ

Для таблиц, использующих стратегию `PARTITION BY column`, INSERT OVERWRITE поддерживает создание новых partitions в целевой таблице путем указания значения ключа partition. Существующие partitions перезаписываются как обычно.

Следующий пример создает partitioned таблицу `activity` и создает новый partition в таблице при вставке данных в нее:

```
CREATE TABLE activity (
id INT          NOT NULL,
dt VARCHAR(10)  NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(`id`)
PARTITION BY (`id`, `dt`)
DISTRIBUTED BY HASH(`id`);

INSERT OVERWRITE activity
PARTITION(id='4', dt='2022-01-01')
WITH LABEL insert_activity_auto_partition
VALUES ('4', '2022-01-01');
```

- Следующий пример перезаписывает целевую таблицу `insert_wiki_edit` столбцами `event_time` и `channel` из исходной таблицы. Значение по умолчанию присваивается столбцам, в которые данные не перезаписываются.

```
INSERT OVERWRITE insert_wiki_edit
WITH LABEL insert_load_wikipedia_ow_3
(
    event_time,
    channel
)
SELECT event_time, channel FROM source_wiki_edit;
```

Dynamic Overwrite

Начиная с v1.5.2, Selena поддерживает новую семантику - Dynamic Overwrite для INSERT OVERWRITE с partitioned таблицами.

В настоящее время поведение по умолчанию INSERT OVERWRITE следующее:

- При перезаписи partitioned таблицы в целом (то есть без указания предложения PARTITION), новые записи данных заменят данные в соответствующих им partitions. Если есть partitions, которые не задействованы, они будут усечены, в то время как другие будут перезаписаны.
- При перезаписи пустой partitioned таблицы (то есть без partitions в ней) и указании предложения PARTITION система возвращает ошибку `ERROR 1064 (HY000): Getting analyzing error. Detail message: Unknown partition 'xxx' in table 'yyy'`.
- При перезаписи partitioned таблицы и указании несуществующего partition в предложении PARTITION система возвращает ошибку `ERROR 1064 (HY000): Getting analyzing error. Detail message: Unknown partition 'xxx' in table 'yyy'`.
- При перезаписи partitioned таблицы записями данных, которые не соответствуют ни одному из указанных partitions в предложении PARTITION, система либо возвращает ошибку `ERROR 1064 (HY000): Insert has filtered data in strict mode` (если включен строгий режим), либо фильтрует несоответствующие записи данных (если строгий режим отключен).

Поведение новой семантики Dynamic Overwrite сильно отличается:

При перезаписи partitioned таблицы в целом новые записи данных заменят данные в соответствующих им partitions. Если есть partitions, которые не задействованы, они будут оставлены без изменений, вместо того чтобы быть усеченными или удаленными. И если есть новые записи данных, соответствующие несуществующему partition, система создаст partition.

Семантика Dynamic Overwrite отключена по умолчанию. Чтобы включить её, необходимо установить системную переменную `dynamic_overwrite` в `true`.

Включить Dynamic Overwrite в текущем сеансе:

```
SET dynamic_overwrite = true;
```

Вы также можете установить её в подсказке оператора INSERT OVERWRITE, чтобы она вступила в силу только для этого оператора:

Пример:

```
INSERT /*+set_var(dynamic_overwrite = true)*/ OVERWRITE insert_wiki_edit
SELECT * FROM source_wiki_edit;
```

3.7.7 Вставка данных в таблицу с generated столбцами

Generated столбец - это специальный столбец, значение которого получается из предопределенного выражения или вычисления на основе других столбцов. Generated столбцы особенно полезны, когда ваш запрос включает вычисления дорогостоящих выражений, например, запрос определенного поля из значения JSON или вычисление данных ARRAY. Selena вычисляет выражение и сохраняет результаты в generated столбцах во время загрузки данных в таблицу, тем самым избегая вычисления выражения во время запросов и улучшая производительность запросов.

Вы можете загружать данные в таблицу с generated столбцами, используя INSERT.

Следующий пример создает таблицу `insert_generated_columns` и вставляет в нее строку. Таблица содержит два generated столбца: `avg_array` и `get_string`. `avg_array` вычисляет среднее значение данных ARRAY в `data_array`, а `get_string` извлекает строки из пути JSON `a` в `data_json`.

```
CREATE TABLE insert_generated_columns (
  id            INT(11)            NOT NULL     COMMENT "ID",
  data_array    ARRAY<INT(11)>     NOT NULL     COMMENT "ARRAY",
  data_json     JSON               NOT NULL     COMMENT "JSON",
  avg_array     DOUBLE             NULL
      AS array_avg(data_array)                  COMMENT "Get the average of
ARRAY",
  get_string    VARCHAR(65533)     NULL
      AS get_json_string(json_string(data_json), '$.a') COMMENT "Extract
JSON string"
) ENGINE=OLAP
PRIMARY KEY(id)
DISTRIBUTED BY HASH(id);

INSERT INTO insert_generated_columns
VALUES (1, [1,2], parse_json('{"a" : 1, "b" : 2}'));
```

> **ПРИМЕЧАНИЕ**
>
> Прямая загрузка данных в generated столбцы не поддерживается.

Вы можете запросить таблицу, чтобы проверить данные в ней.

```
mysql> SELECT * FROM insert_generated_columns;
+------+------------+------------------+-----------+------------+
| id   | data_array | data_json        | avg_array | get_string |
+------+------------+------------------+-----------+------------+
|    1 | [1,2]      | {"a": 1, "b": 2} |       1.5 | 1          |
+------+------------+------------------+-----------+------------+
1 row in set (0.02 sec)
```

3.7.8 Вставка данных с PROPERTIES

Начиная с v1.5.2, операторы INSERT поддерживают настройку PROPERTIES, которые могут служить для различных целей. PROPERTIES переопределяют соответствующие им переменные.

Включение строгого режима

Начиная с v1.5.2, вы можете включить строгий режим и установить `max_filter_ratio` для INSERT из FILES(). Строгий режим для INSERT из FILES() имеет такое же поведение, как и для других методов загрузки.

Если вы хотите загрузить набор данных с некоторыми несоответствующими строками, вы можете либо отфильтровать эти несоответствующие строки, либо загрузить их и присвоить NULL значения несоответствующим столбцам. Вы можете достичь этого, используя свойства `strict_mode` и `max_filter_ratio`.

- Для фильтрации несоответствующих строк: установите `strict_mode` в `true`, а `max_filter_ratio` в желаемое значение.
- Для загрузки всех несоответствующих строк со значениями NULL: установите `strict_mode` в `false`.

Следующий пример вставляет строки данных из файла Parquet **parquet/insert_wiki_edit_append.parquet** в bucket AWS S3 `inserttest` в таблицу `insert_wiki_edit`, включает строгий режим для фильтрации несоответствующих записей данных и допускает не более 10% ошибочных данных:

```
INSERT INTO insert_wiki_edit
PROPERTIES(
    "strict_mode" = "true",
    "max_filter_ratio" = "0.1"
)
SELECT * FROM FILES(
    "path" = "s3://inserttest/parquet/insert_wiki_edit_append.parquet",
    "format" = "parquet",
    "aws.s3.access_key" = "XXXXXXXXXX",
    "aws.s3.secret_key" = "YYYYYYYYYY",
    "aws.s3.region" = "us-west-2"
);
```

ПРИМЕЧАНИЕ

`strict_mode` и `max_filter_ratio` поддерживаются только для INSERT из FILES(). INSERT из таблиц не поддерживает эти свойства.

Установка длительности тайм-аута

Начиная с v1.5.2, вы можете установить длительность тайм-аута для операторов INSERT с помощью свойств.

Следующий пример вставляет данные из исходной таблицы `source_wiki_edit` в целевую таблицу `insert_wiki_edit` с длительностью тайм-аута, установленной на `2` секунды.

```
INSERT INTO insert_wiki_edit
PROPERTIES(
    "timeout" = "2"
)
SELECT * FROM source_wiki_edit;
```

ПРИМЕЧАНИЕ

Начиная с v1.5.2, вы также можете установить длительность тайм-аута INSERT, используя системную переменную `insert_timeout`, которая применяется к операциям, включающим INSERT (например, UPDATE, DELETE, CTAS, обновление материализованных представлений, сбор статистики и PIPE). В версиях ранее v1.5.2 соответствующая переменная - `query_timeout`.

Сопоставление столбцов по имени

По умолчанию INSERT сопоставляет столбцы в исходной и целевой таблицах по их позициям, то есть сопоставлению столбцов в операторе.

Следующий пример явно сопоставляет каждый столбец в исходной и целевой таблицах по их позициям:

```
INSERT INTO insert_wiki_edit (
    event_time,
    channel,
    user
)
SELECT event_time, channel, user FROM source_wiki_edit;
```

Сопоставление столбцов изменится, если вы измените порядок `channel` и `user` либо в списке столбцов, либо в операторе SELECT.

```
INSERT INTO insert_wiki_edit (
    event_time,
    channel,
    user
)
SELECT event_time, user, channel FROM source_wiki_edit;
```

Здесь загруженные данные, вероятно, не то, что вы хотите, потому что `channel` в целевой таблице `insert_wiki_edit` будет заполнен данными из `user` в исходной таблице `source_wiki_edit`.

Добавив предложение `BY NAME` в оператор INSERT, система обнаружит имена столбцов в исходной и целевой таблицах и сопоставит столбцы с одинаковым именем.

> **ПРИМЕЧАНИЕ**
>
> - Вы не можете указать список столбцов, если указано `BY NAME`.
> - Если `BY NAME` не указано, система сопоставляет столбцы по позиции столбцов в списке столбцов и операторе SELECT.

Следующий пример сопоставляет каждый столбец в исходной и целевой таблицах по их именам:

```
INSERT INTO insert_wiki_edit BY NAME
SELECT event_time, user, channel FROM source_wiki_edit;
```

В этом случае изменение порядка `channel` и `user` не изменит сопоставление столбцов.

3.7.9 Асинхронная загрузка данных с использованием INSERT

Загрузка данных с помощью INSERT отправляет синхронную транзакцию, которая может завершиться ошибкой из-за прерывания сеанса или тайм-аута. Вы можете отправить асинхронную транзакцию INSERT, используя SUBMIT TASK. Эта функция поддерживается с Selena v1.5.2.

- Следующий пример асинхронно вставляет данные из исходной таблицы в целевую таблицу `insert_wiki_edit`.

```
SUBMIT TASK AS INSERT INTO insert_wiki_edit
SELECT * FROM source_wiki_edit;
```

- Следующий пример асинхронно перезаписывает таблицу `insert_wiki_edit` данными из исходной таблицы.

```
SUBMIT TASK AS INSERT OVERWRITE insert_wiki_edit
SELECT * FROM source_wiki_edit;
```

- Следующий пример асинхронно перезаписывает таблицу `insert_wiki_edit` данными из исходной таблицы и увеличивает тайм-аут запроса до `100000` секунд, используя подсказку.

```
SUBMIT /*+set_var(insert_timeout=100000)*/ TASK AS
INSERT OVERWRITE insert_wiki_edit
SELECT * FROM source_wiki_edit;
```

- Следующий пример асинхронно перезаписывает таблицу `insert_wiki_edit` данными из исходной таблицы и указывает имя задачи как `async`.

```
SUBMIT TASK async
AS INSERT OVERWRITE insert_wiki_edit
SELECT * FROM source_wiki_edit;
```

Вы можете проверить статус асинхронной задачи INSERT, запросив представление метаданных `task_runs` в Information Schema.

Следующий пример проверяет статус задачи INSERT `async`.

```
SELECT * FROM information_schema.task_runs WHERE task_name = 'async';
```

3.7.10 Проверка статуса задания INSERT

Проверка через результат

Синхронная транзакция INSERT возвращает различный статус в соответствии с результатом транзакции.

- **Транзакция успешна**

Selena возвращает следующее, если транзакция успешна:

```
Query OK, 2 rows affected (0.05 sec)
{'label':'insert_load_wikipedia', 'status':'VISIBLE', 'txnId':'1006'}
```

- **Транзакция завершена неудачно**

Если все строки данных не удалось загрузить в целевую таблицу, транзакция INSERT завершается ошибкой. Selena возвращает следующее, если транзакция завершается ошибкой:

```
ERROR 1064 (HY000): Insert has filtered data in strict mode,
tracking_url=http://x.x.x.x:yyyy/api/_load_error_log?
file=error_log_9f0a4fd0b64e11ec_906bbede076e9d08
```

Вы можете определить проблему, проверив лог с помощью `tracking_url`.

Проверка через Information Schema

Вы можете использовать оператор SELECT для запроса результатов одного или нескольких заданий загрузки из таблицы `loads` в базе данных `information_schema`. Эта функция поддерживается с v1.5.2.

Пример 1: Запрос результатов заданий загрузки, выполненных в базе данных `load_test`, сортировка результатов по времени создания (`CREATE_TIME`) в порядке убывания и возврат только верхнего результата.

```
SELECT * FROM information_schema.loads
WHERE database_name = 'load_test'
ORDER BY create_time DESC
LIMIT 1\G
```

Пример 2: Запрос результата задания загрузки (с меткой `insert_load_wikipedia`), выполненного в базе данных `load_test`:

```
SELECT * FROM information_schema.loads
WHERE database_name = 'load_test' and label = 'insert_load_wikipedia'\G
```

Возвращается следующее:

```
*************************** 1. row ***************************
              JOB_ID: 21319
               LABEL: insert_load_wikipedia
       DATABASE_NAME: load_test
               STATE: FINISHED
            PROGRESS: ETL:100%; LOAD:100%
                TYPE: INSERT
            PRIORITY: NORMAL
           SCAN_ROWS: 0
       FILTERED_ROWS: 0
     UNSELECTED_ROWS: 0
           SINK_ROWS: 2
            ETL_INFO:
           TASK_INFO: resource:N/A; timeout(s):300; max_filter_ratio:0.0
         CREATE_TIME: 2023-08-09 10:42:23
      ETL_START_TIME: 2023-08-09 10:42:23
     ETL_FINISH_TIME: 2023-08-09 10:42:23
     LOAD_START_TIME: 2023-08-09 10:42:23
    LOAD_FINISH_TIME: 2023-08-09 10:42:24
         JOB_DETAILS: {"All backends":{"5ebf11b5-365e-11ee-9e4a-7a563fb695da"
[10006]},"FileNumber":0,"FileSize":0,"InternalTableLoadBytes":175,"InternalTa
backends":{"5ebf11b5-365e-11ee-9e4a-7a563fb695da":[]}}
           ERROR_MSG: NULL
        TRACKING_URL: NULL
        TRACKING_SQL: NULL
REJECTED_RECORD_PATH: NULL
1 row in set (0.01 sec)
```

Для получения информации о полях в результатах см. Information Schema > loads.

Проверка через команду curl

Вы можете проверить статус транзакции INSERT, используя команду curl.

Запустите терминал и выполните следующую команду:

```
curl --location-trusted -u <username>:<password> \
  http://<fe_address>:<fe_http_port>/api/<db_name>/_load_info?label=
<label_name>
```

Следующий пример проверяет статус транзакции с меткой `insert_load_wikipedia`.

```
curl --location-trusted -u <username>:<password> \
  http://x.x.x.x:8030/api/load_test/_load_info?label=insert_load_wikipedia
```

> **ПРИМЕЧАНИЕ**
>
> Если вы используете учетную запись, для которой не установлен пароль, вам необходимо ввести только `<username>:`.

Возвращается следующее:

```
{
   "jobInfo":{
      "dbName":"load_test",
      "tblNames":[
         "source_wiki_edit"
      ],
      "label":"insert_load_wikipedia",
      "state":"FINISHED",
      "failMsg":"",
      "trackingUrl":""
   },
   "status":"OK",
   "msg":"Success"
}
```

3.7.11 Конфигурация

Вы можете установить следующие параметры конфигурации для транзакции INSERT:

- **Конфигурация FE**

Конфигурация FE	Описание
insert_load_default_timeout_second	Тайм-аут по умолчанию для транзакции INSERT. Единица измерения: секунда. Если текущая транзакция INSERT не завершена в течение времени, установленного этим параметром, она будет отменена системой, и статус будет CANCELLED. Что касается текущей версии Selena, вы можете указать только единый тайм-аут для всех транзакций INSERT, используя этот параметр, и вы не можете установить другой тайм-аут для конкретной транзакции INSERT. По умолчанию 3600 секунд (1 час). Если транзакция INSERT не может быть завершена в указанное время, вы можете увеличить тайм-аут, настроив этот параметр.

- **Переменные сеанса**

Переменная сеанса	Описание
enable_insert_strict	Значение переключателя для управления тем, является ли транзакция INSERT толерантной к недопустимым строкам данных. Когда оно установлено в `true`, транзакция завершается ошибкой, если любая из строк данных является недопустимой. Когда оно установлено в `false`, транзакция успешна, когда по крайней мере одна строка данных была загружена правильно, и будет возвращена метка. По умолчанию `true`. Вы можете установить эту переменную с помощью команды `SET enable_insert_strict = {true or false};`.
insert_timeout	Тайм-аут для оператора INSERT. Единица измерения: секунда. Вы можете установить эту переменную с помощью команды `SET insert_timeout = xxx;`.

3.8 SQL-транзакции

> *Beta* **BETA FEATURE**
>
> Рекомендации по использованию бета-функций

Запустите простую SQL-транзакцию для фиксации нескольких DML-операторов в пакете.

3.8.1 Обзор

Начиная с версии v1.5.2, Selena поддерживает SQL-транзакции для обеспечения атомарности обновляемых таблиц при манипулировании данными в нескольких таблицах.

Транзакция состоит из нескольких SQL-операторов, которые обрабатываются в одной атомарной единице. Операторы в транзакции либо применяются, либо отменяются вместе, что гарантирует свойства ACID (атомарность, согласованность, изоляция и надёжность) транзакции.

В настоящее время SQL-транзакции в Selena поддерживают следующие операции:

- INSERT INTO
- UPDATE
- DELETE

> **ПРИМЕЧАНИЕ**
>
> - INSERT OVERWRITE в настоящее время не поддерживается.
> - Несколько операторов INSERT для одной и той же таблицы в рамках транзакции поддерживаются только в cluster с shared-data начиная с версии v2.0.0.
> - UPDATE и DELETE поддерживаются только в cluster с shared-data начиная с версии v2.0.0.

Начиная с версии v2.0.0, в рамках одной SQL-транзакции:

- **Несколько операторов INSERT** для одной таблицы поддерживаются.
- **Только один оператор UPDATE *ИЛИ* DELETE** для одной таблицы разрешён.
- **Оператор UPDATE *ИЛИ* DELETE после** операторов INSERT для той же таблицы **не разрешён**.

Свойства ACID транзакции гарантируются только на ограниченном уровне изоляции READ COMMITTED, то есть:

- Оператор работает только с данными, которые были зафиксированы до начала выполнения оператора.
- Два последовательных оператора в одной транзакции могут работать с разными данными, если другая транзакция была зафиксирована между выполнением первого и второго операторов.
- Изменения данных, внесённые предшествующими DML-операторами, невидимы для последующих операторов в той же транзакции.

Транзакция связана с одной сессией. Несколько сессий не могут разделять одну и ту же транзакцию.

3.8.2 Использование

1. Транзакция должна быть начата выполнением оператора START TRANSACTION. Selena также поддерживает синоним BEGIN.

   ```
   { START TRANSACTION | BEGIN [ WORK ] }
   ```

2. После запуска транзакции вы можете определить несколько DML-операторов в транзакции. Для получения подробной информации см. Примечания по

использованию.

3. Транзакция должна быть явно завершена выполнением `COMMIT` или `ROLLBACK`.

   - Чтобы применить (зафиксировать) транзакцию, используйте следующий синтаксис:

     ```
     COMMIT [ WORK ]
     ```

   - Чтобы отменить (откатить) транзакцию, используйте следующий синтаксис:

     ```
     ROLLBACK [ WORK ]
     ```

3.8.3 Пример

1. Создайте демонстрационную таблицу `desT` в cluster с shared-data и загрузите в неё данные.

   > **ПРИМЕЧАНИЕ**
   >
   > Если вы хотите попробовать этот пример в cluster с shared-nothing, вы должны пропустить шаг 3 и определить только один оператор INSERT на шаге 4.

   ```
   CREATE TABLE desT (
       k int,
       v int
   ) PRIMARY KEY(k);

   INSERT INTO desT VALUES
   (1,1),
   (2,2),
   (3,3);
   ```

2. Запустите транзакцию.

   ```
   START TRANSACTION;
   ```

Или

```
BEGIN WORK;
```

3. Определите оператор UPDATE или DELETE.

```
UPDATE desT SET v = v + 1 WHERE k = 1,
```

Или

```
DELETE FROM desT WHERE k = 1;
```

4. Определите несколько операторов INSERT.

```
-- Вставка данных с указанными значениями.
INSERT INTO desT VALUES (4,4);
-- Вставка данных из одной нативной таблицы в другую.
INSERT INTO desT SELECT * FROM srcT;
-- Вставка данных из удалённого хранилища.
INSERT INTO desT
    SELECT * FROM FILES(
        "path" = "s3://inserttest/parquet/srcT.parquet",
        "format" = "parquet",
        "aws.s3.access_key" = "XXXXXXXXXX",
        "aws.s3.secret_key" = "YYYYYYYYYY",
        "aws.s3.region" = "us-west-2"
);
```

5. Примените или отмените транзакцию.

   - Чтобы применить SQL-операторы в транзакции.

     ```
     COMMIT WORK;
     ```

   - Чтобы отменить SQL-операторы в транзакции.

```
ROLLBACK WORK;
```

3.8.4 Примечания по использованию

- В настоящее время Selena поддерживает операторы SELECT, INSERT, UPDATE и DELETE в SQL-транзакциях. UPDATE и DELETE поддерживаются только в cluster с shared-data начиная с версии v2.0.0.
- Операторы SELECT для таблиц, данные которых были изменены в той же транзакции, не разрешены.
- Несколько операторов INSERT для одной и той же таблицы в рамках транзакции поддерживаются только в cluster с shared-data начиная с версии v2.0.0.
- В рамках транзакции вы можете определить только один оператор UPDATE или DELETE для каждой таблицы, и он должен предшествовать операторам INSERT.
- Последующие DML-операторы не могут читать незафиксированные изменения, внесённые предшествующими операторами в той же транзакции. Например, целевая таблица предшествующего оператора INSERT не может быть исходной таблицей последующих операторов. В противном случае система вернёт ошибку.
- Все целевые таблицы DML-операторов в транзакции должны находиться в одной базе данных. Межбазовые операции не разрешены.
- В настоящее время INSERT OVERWRITE не поддерживается.
- Вложенные транзакции не разрешены. Вы не можете указывать BEGIN WORK внутри пары BEGIN-COMMIT/ROLLBACK.
- Если сессия, к которой принадлежит выполняющаяся транзакция, завершается или закрывается, транзакция автоматически откатывается.
- Selena поддерживает только ограниченный уровень изоляции транзакций READ COMMITTED, как описано выше.
- Проверки конфликтов записи не поддерживаются. Когда две транзакции одновременно записывают в одну и ту же таблицу, обе транзакции могут быть успешно зафиксированы. Видимость (порядок) изменений данных зависит от порядка выполнения операторов COMMIT WORK.

3.9 Загрузка данных с использованием транзакционного интерфейса Stream Load

Начиная с версии v1.5.2, Selena предоставляет транзакционный интерфейс Stream Load для реализации двухфазной фиксации (2PC) для транзакций, которые выполняются для загрузки данных из внешних систем, таких как Apache Flink и Apache Kafka.

Транзакционный интерфейс Stream Load помогает повысить производительность потоковых загрузок с высокой параллельностью.

Начиная с версии v2.0.0, транзакционный интерфейс Stream Load поддерживает Multi-table Transaction, то есть загрузку данных в несколько таблиц в рамках одной базы данных.

В этой теме описывается транзакционный интерфейс Stream Load и способы загрузки данных в Selena с использованием этого интерфейса.

3.9.1 Описание

Транзакционный интерфейс Stream Load поддерживает использование инструмента или языка, совместимого с протоколом HTTP, для вызова операций API. В этой теме в качестве примера используется curl для объяснения использования этого интерфейса. Этот интерфейс предоставляет различные функции, такие как управление транзакциями, запись данных, предварительная фиксация транзакций, дедупликация транзакций и управление тайм-аутом транзакций.

> **ПРИМЕЧАНИЕ**
>
> Stream Load поддерживает форматы файлов CSV и JSON. Этот метод рекомендуется, если вы хотите загрузить данные из небольшого количества файлов, размер каждого из которых не превышает 10 ГБ. Stream Load не поддерживает формат файлов Parquet. Если вам нужно загрузить данные из файлов Parquet, используйте INSERT+files().

Управление транзакциями

Транзакционный интерфейс Stream Load предоставляет следующие операции API, которые используются для управления транзакциями:

- `/api/transaction/begin`: запускает новую транзакцию.
- `/api/transaction/prepare`: предварительно фиксирует текущую транзакцию и делает изменения данных временно постоянными. После предварительной фиксации транзакции вы можете продолжить её фиксацию или откат. Если ваш cluster выходит из строя после предварительной фиксации транзакции, вы всё равно можете продолжить фиксацию транзакции после восстановления cluster.

- `/api/transaction/commit`: фиксирует текущую транзакцию, чтобы сделать изменения данных постоянными.
- `/api/transaction/rollback`: откатывает текущую транзакцию, чтобы отменить изменения данных.

> **ПРИМЕЧАНИЕ**
>
> После предварительной фиксации транзакции не продолжайте записывать данные, используя транзакцию. Если вы продолжите записывать данные, используя транзакцию, ваш запрос на запись вернёт ошибки.

На следующей диаграмме показана связь между состояниями транзакции и операциями:

```
stateDiagram-v2
    direction LR
    [*] --> PREPARE : begin
    PREPARE --> PREPARED : prepare
    PREPARE --> ABORTED : rollback
    PREPARED --> COMMITTED : commit
    PREPARED --> ABORTED : rollback
```

Запись данных

Транзакционный интерфейс Stream Load предоставляет операцию `/api/transaction/load`, которая используется для записи данных. Вы можете вызывать эту операцию несколько раз в рамках одной транзакции.

Начиная с версии v2.0.0, вы можете вызывать операции `/api/transaction/load` для разных таблиц, чтобы загружать данные в несколько таблиц в рамках одной базы данных.

Дедупликация транзакций

Транзакционный интерфейс Stream Load переносит механизм меток Selena. Вы можете привязать уникальную метку к каждой транзакции для достижения гарантий "не более одного раза" для транзакций.

Управление тайм-аутом транзакций

При запуске транзакции вы можете использовать поле `timeout` в заголовке HTTP-запроса для указания периода тайм-аута (в секундах) для транзакции из состояния `PREPARE` в состояние `PREPARED`. Если транзакция не была подготовлена после этого периода, она будет автоматически прервана. Если это поле не указано, значение по умолчанию определяется конфигурацией FE `stream_load_default_timeout_second` (по умолчанию: 600 секунд).

При запуске транзакции вы также можете использовать поле `idle_transaction_timeout` в заголовке HTTP-запроса для указания периода тайм-аута (в секундах), в течение которого транзакция может оставаться бездействующей. Если данные не записываются в течение этого периода, транзакция будет автоматически откачена.

При подготовке транзакции вы можете использовать поле `prepared_timeout` в заголовке HTTP-запроса для указания периода тайм-аута (в секундах) для транзакции из состояния `PREPARED` в состояние `COMMITTED`. Если транзакция не была зафиксирована после этого периода, она будет автоматически прервана. Если это поле не указано, значение по умолчанию определяется конфигурацией FE `prepared_transaction_default_timeout_second` (по умолчанию: 86400 секунд). `prepared_timeout` поддерживается начиная с версии v1.5.2.

3.9.2 Преимущества

Транзакционный интерфейс Stream Load приносит следующие преимущества:

- **Семантика "ровно один раз"**

  Транзакция разделена на две фазы — предварительную фиксацию и фиксацию, что упрощает загрузку данных между системами. Например, этот интерфейс может гарантировать семантику "ровно один раз" для загрузки данных из Flink.

- **Улучшенная производительность загрузки**

  Если вы запускаете задание загрузки с помощью программы, транзакционный интерфейс Stream Load позволяет объединять несколько мини-пакетов данных по требованию, а затем отправлять их все сразу в рамках одной транзакции, вызывая операцию `/api/transaction/commit`. Таким образом, требуется загрузить меньше версий данных, и производительность загрузки улучшается.

3.9.3 Ограничения

Транзакционный интерфейс Stream Load имеет следующие ограничения:

- Транзакции **с одной базой данных и несколькими таблицами** поддерживаются начиная с версии v2.0.0. Поддержка транзакций **с несколькими базами данных и несколькими таблицами** находится в разработке.

- Поддерживаются только **параллельные записи данных от одного клиента**. Поддержка **параллельных записей данных от нескольких клиентов** находится в разработке.

- Операция `/api/transaction/load` может вызываться несколько раз в рамках одной транзакции. В этом случае настройки параметров (кроме `table`), указанные для всех вызываемых операций `/api/transaction/load`, должны быть одинаковыми.

- При загрузке данных в формате CSV с использованием транзакционного интерфейса Stream Load убедитесь, что каждая запись данных в вашем файле данных заканчивается разделителем строк.

3.9.4 Меры предосторожности

- Если вызванная операция `/api/transaction/begin`, `/api/transaction/load` или `/api/transaction/prepare` возвращает ошибки, транзакция завершается неудачей и автоматически откатывается.
- При вызове операции `/api/transaction/begin` для запуска новой транзакции необходимо указать метку. Обратите внимание, что последующие операции `/api/transaction/load`, `/api/transaction/prepare` и `/api/transaction/commit` должны использовать ту же метку, что и операция `/api/transaction/begin`.
- Если вы используете метку текущей транзакции для вызова операции `/api/transaction/begin` для запуска новой транзакции, предыдущая транзакция завершится неудачей и будет откачена.
- Если вы используете транзакцию с несколькими таблицами для загрузки данных в разные таблицы, вы должны указать параметр `-H "transaction_type:multi"` для всех операций, участвующих в транзакции.
- Разделитель столбцов и разделитель строк по умолчанию, которые Selena поддерживает для данных в формате CSV, — это `\t` и `\n`. Если ваш файл данных не использует разделитель столбцов или разделитель строк по умолчанию, вы должны

использовать `"column_separator: <column_separator>"` или `"row_delimiter: <row_delimiter>"` для указания разделителя столбцов или разделителя строк, фактически используемого в вашем файле данных, при вызове операции `/api/transaction/load`.

3.9.5 Перед началом работы

Проверка привилегий

Вы можете загружать данные в таблицы Selena только как пользователь, имеющий привилегию INSERT на эти таблицы Selena. Если у вас нет привилегии INSERT, следуйте инструкциям в разделе GRANT, чтобы предоставить привилегию INSERT пользователю, которого вы используете для подключения к вашему cluster Selena. Синтаксис: `GRANT INSERT ON TABLE <table_name> IN DATABASE <database_name> TO { ROLE <role_name> | USER <user_identity>}`.

Проверка сетевой конфигурации

Убедитесь, что машина, на которой находятся загружаемые данные, имеет доступ к узлам FE и BE cluster Selena через порты `http_port` (по умолчанию: `8030`) и `be_http_port` (по умолчанию: `8040`) соответственно.

3.9.6 Основные операции

Подготовка примера данных

В этой теме в качестве примера используются данные в формате CSV.

1. В пути `/home/disk1/` вашей локальной файловой системы создайте CSV-файл с именем `example1.csv`. Файл состоит из трёх столбцов, которые представляют идентификатор пользователя, имя пользователя и оценку пользователя соответственно.

```
1,Lily,23
2,Rose,23
3,Alice,24
4,Julia,25
```

2. В базе данных Selena `test_db` создайте таблицу с первичным ключом с именем `table1`. Таблица состоит из трёх столбцов: `id`, `name` и `score`, где `id` является

первичным ключом.

```
CREATE TABLE `table1`
(
    `id` int(11) NOT NULL COMMENT "user ID",
    `name` varchar(65533) NULL COMMENT "user name",
    `score` int(11) NOT NULL COMMENT "user score"
)
ENGINE=OLAP
PRIMARY KEY(`id`)
DISTRIBUTED BY HASH(`id`) BUCKETS 10;
```

Запуск транзакции

Синтаксис

```
curl --location-trusted -u <username>:<password> -H "label:<label_name>" \
    -H "Expect:100-continue" \
    [-H "transaction_type:multi"]\  # Необязательно. Инициирует транзакцию
с несколькими таблицами.
    -H "db:<database_name>" -H "table:<table_name>" \
    -XPOST http://<fe_host>:<fe_http_port>/api/transaction/begin
```

> **ПРИМЕЧАНИЕ**
>
> Укажите `-H "transaction_type:multi"` в команде, если вы хотите загрузить данные в разные таблицы в рамках транзакции.

Пример

```
curl --location-trusted -u <jack>:<123456> -H
"label:streamload_txn_example1_table1" \
    -H "Expect:100-continue" \
    -H "db:test_db" -H "table:table1" \
    -XPOST http://<fe_host>:<fe_http_port>/api/transaction/begin
```

> **ПРИМЕЧАНИЕ**
>
> В этом примере `streamload_txn_example1_table1` указана как метка транзакции.

Возвращаемый результат

- Если транзакция успешно запущена, возвращается следующий результат:

```
{
    "Status": "OK",
    "Message": "",
    "Label": "streamload_txn_example1_table1",
    "TxnId": 9032,
    "BeginTxnTimeMs": 0
}
```

- Если транзакция связана с дублирующей меткой, возвращается следующий результат:

```
{
    "Status": "LABEL_ALREADY_EXISTS",
    "ExistingJobStatus": "RUNNING",
    "Message": "Label [streamload_txn_example1_table1] has already been
used."
}
```

- Если возникают ошибки, отличные от дублирующей метки, возвращается следующий результат:

```
{
    "Status": "FAILED",
    "Message": ""
}
```

Запись данных

Синтаксис

```
curl --location-trusted -u <username>:<password> -H "label:<label_name>" \
    -H "Expect:100-continue" \
    [-H "transaction_type:multi"]\  # Необязательно. Загружает данные через транзакцию с несколькими таблицами.
    -H "db:<database_name>" -H "table:<table_name>" \
    -T <file_path> \
    -XPUT http://<fe_host>:<fe_http_port>/api/transaction/load
```

> **ПРИМЕЧАНИЕ**
>
> - При вызове операции `/api/transaction/load` вы должны использовать `<file_path>` для указания пути сохранения файла данных, который вы хотите загрузить.
> - Вы можете вызывать операции `/api/transaction/load` с различными значениями параметра `table` для загрузки данных в разные таблицы в рамках одной базы данных. В этом случае вы должны указать `-H "transaction_type:multi"` в команде.

Пример

```
curl --location-trusted -u <jack>:<123456> -H "label:streamload_txn_example1_table1" \
    -H "Expect:100-continue" \
    -H "db:test_db" -H "table:table1" \
    -T /home/disk1/example1.csv \
    -H "column_separator: ," \
    -XPUT http://<fe_host>:<fe_http_port>/api/transaction/load
```

> **ПРИМЕЧАНИЕ**
>
> В этом примере разделитель столбцов, используемый в файле данных `example1.csv`, — запятые (`,`) вместо разделителя столбцов Selena по умолчанию (`\t`). Поэтому при вызове операции `/api/transaction/load` необходимо использовать `"column_separator: <column_separator>"` для указания запятых (`,`) в качестве разделителя столбцов.

Возвращаемый результат

- Если запись данных успешна, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Seq": 0,
    "Label": "streamload_txn_example1_table1",
    "Status": "OK",
    "Message": "",
    "NumberTotalRows": 5265644,
    "NumberLoadedRows": 5265644,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 10737418067,
    "LoadTimeMs": 418778,
    "StreamLoadPutTimeMs": 68,
    "ReceivedDataTimeMs": 38964,
}
```

- Если транзакция считается неизвестной, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "TXN_NOT_EXISTS"
}
```

- Если транзакция считается в недопустимом состоянии, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "Transcation State Invalid"
}
```

- Если возникают ошибки, отличные от неизвестной транзакции и недопустимого состояния, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": ""
}
```

Предварительная фиксация транзакции

Синтаксис

```
curl --location-trusted -u <username>:<password> -H "label:<label_name>" \
    -H "Expect:100-continue" \
    [-H "transaction_type:multi"]\  # Необязательно. Предварительно
фиксирует транзакцию с несколькими таблицами.
    -H "db:<database_name>" \
    [-H "prepared_timeout:<timeout_seconds>"] \
    -XPOST http://<fe_host>:<fe_http_port>/api/transaction/prepare
```

> **ПРИМЕЧАНИЕ**
>
> Укажите `-H "transaction_type:multi"` в команде, если транзакция, которую вы хотите предварительно зафиксировать, является транзакцией с несколькими таблицами.

Пример

```
curl --location-trusted -u <jack>:<123456> -H
"label:streamload_txn_example1_table1" \
    -H "Expect:100-continue" \
    -H "db:test_db" \
    -H "prepared_timeout:300" \
    -XPOST http://<fe_host>:<fe_http_port>/api/transaction/prepare
```

> **ПРИМЕЧАНИЕ**

> Поле `prepared_timeout` является необязательным. Если оно не указано, значение по умолчанию определяется конфигурацией FE `prepared_transaction_default_timeout_second` (по умолчанию: 86400 секунд). `prepared_timeout` поддерживается начиная с версии v1.5.2.

Возвращаемый результат

- Если предварительная фиксация успешна, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "OK",
    "Message": "",
    "NumberTotalRows": 5265644,
    "NumberLoadedRows": 5265644,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 10737418067,
    "LoadTimeMs": 418778,
    "StreamLoadPutTimeMs": 68,
    "ReceivedDataTimeMs": 38964,
    "WriteDataTimeMs": 417851
    "CommitAndPublishTimeMs": 1393
}
```

- Если транзакция считается несуществующей, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "Transcation Not Exist"
}
```

- Если предварительная фиксация истекает по таймауту, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "commit timeout",
}
```

- Если возникают ошибки, отличные от несуществующей транзакции и тайм-аута предварительной фиксации, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "publish timeout"
}
```

Фиксация транзакции

Синтаксис

```
curl --location-trusted -u <username>:<password> -H "label:<label_name>" \
    -H "Expect:100-continue" \
    [-H "transaction_type:multi"]\  # Необязательно. Фиксирует транзакцию с несколькими таблицами.
    -H "db:<database_name>" \
    -XPOST http://<fe_host>:<fe_http_port>/api/transaction/commit
```

> **ПРИМЕЧАНИЕ**
>
> Укажите `-H "transaction_type:multi"` в команде, если транзакция, которую вы хотите зафиксировать, является транзакцией с несколькими таблицами.

Пример

```
curl --location-trusted -u <jack>:<123456> -H
"label:streamload_txn_example1_table1" \
    -H "Expect:100-continue" \
    -H "db:test_db" \
    -XPOST http://<fe_host>:<fe_http_port>/api/transaction/commit
```

Возвращаемый результат

- Если фиксация успешна, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "OK",
    "Message": "",
    "NumberTotalRows": 5265644,
    "NumberLoadedRows": 5265644,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 10737418067,
    "LoadTimeMs": 418778,
    "StreamLoadPutTimeMs": 68,
    "ReceivedDataTimeMs": 38964,
    "WriteDataTimeMs": 417851
    "CommitAndPublishTimeMs": 1393
}
```

- Если транзакция уже была зафиксирована, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "OK",
    "Message": "Transaction already commited",
}
```

- Если транзакция считается несуществующей, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "Transcation Not Exist"
}
```

- Если фиксация истекает по таймауту, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "commit timeout",
}
```

- Если публикация данных истекает по таймауту, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "publish timeout",
    "CommitAndPublishTimeMs": 1393
}
```

- Если возникают ошибки, отличные от несуществующей транзакции и тайм-аута, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": ""
}
```

Откат транзакции

Синтаксис

```
curl --location-trusted -u <username>:<password> -H "label:<label_name>" \
    -H "Expect:100-continue" \
    [-H "transaction_type:multi"]\  # Необязательно. Откатывает транзакцию
с несколькими таблицами.
    -H "db:<database_name>" \
    -XPOST http://<fe_host>:<fe_http_port>/api/transaction/rollback
```

> **ПРИМЕЧАНИЕ**
>
> Укажите `-H "transaction_type:multi"` в команде, если транзакция, которую вы хотите откатить, является транзакцией с несколькими таблицами.

Пример

```
curl --location-trusted -u <jack>:<123456> -H
"label:streamload_txn_example1_table1" \
    -H "Expect:100-continue" \
    -H "db:test_db" \
    -XPOST http://<fe_host>:<fe_http_port>/api/transaction/rollback
```

Возвращаемый результат

- Если откат успешен, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "OK",
    "Message": ""
}
```

- Если транзакция считается несуществующей, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": "Transcation Not Exist"
}
```

- Если возникают ошибки, отличные от несуществующей транзакции, возвращается следующий результат:

```
{
    "TxnId": 1,
    "Label": "streamload_txn_example1_table1",
    "Status": "FAILED",
    "Message": ""
}
```

3.9.7 Справочные материалы

Информацию о подходящих сценариях применения и поддерживаемых форматах файлов данных Stream Load, а также о принципе работы Stream Load см. в Загрузка из локальной файловой системы через Stream Load.

Информацию о синтаксисе и параметрах для создания заданий Stream Load см. в STREAM LOAD.

3.10 Синхронизация в реальном времени из MySQL

Selena поддерживает несколько методов синхронизации данных из MySQL в Selena в реальном времени, обеспечивая аналитику больших объемов данных с низкой задержкой.

В этой теме описывается, как синхронизировать данные из MySQL в Selena в реальном времени (в течение секунд) через Apache Flink®.

Вы можете загружать данные в таблицы Selena только как пользователь, имеющий привилегию INSERT на эти таблицы Selena. Если у вас нет привилегии INSERT, следуйте инструкциям в разделе GRANT, чтобы предоставить привилегию INSERT пользователю, которого вы используете для подключения к вашему cluster Selena. Синтаксис: `GRANT`

`INSERT ON TABLE <table_name> IN DATABASE <database_name> TO { ROLE <role_name> | USER <user_identity>}`.

3.10.1 Как это работает

ПОДСКАЗКА

Flink CDC используется для синхронизации из MySQL в Flink. В этой теме используется Flink CDC версии ниже 3.0, поэтому SMT используется для синхронизации схем таблиц. Однако, если используется Flink CDC 3.0, не требуется использовать SMT для синхронизации схем таблиц в Selena. Flink CDC 3.0 может даже синхронизировать схемы всей базы данных MySQL, шардированные базы данных и таблицы, а также поддерживает синхронизацию изменений схемы. Подробное описание использования см. в Streaming ELT from MySQL to Selena.

На следующем рисунке показан весь процесс синхронизации.

Синхронизация в реальном времени из MySQL через Flink в Selena реализуется в два этапа: синхронизация схемы базы данных и таблиц и синхронизация данных. Сначала SMT преобразует схему базы данных и таблиц MySQL в операторы создания таблиц для Selena. Затем cluster Flink запускает задания Flink для синхронизации полных и инкрементальных данных MySQL в Selena.

К СВЕДЕНИЮ

Процесс синхронизации гарантирует exactly-once семантику.

Процесс синхронизации:

1. Синхронизация схемы базы данных и таблиц.

SMT читает схему базы данных и таблиц MySQL для синхронизации и генерирует SQL файлы для создания целевой базы данных и таблиц в Selena. Эта операция основана на информации MySQL и Selena в конфигурационном файле SMT.

2. Синхронизация данных.

   a. SQL клиент Flink выполняет оператор загрузки данных `INSERT INTO SELECT` для отправки одного или нескольких заданий Flink в cluster Flink.

   b. Cluster Flink выполняет задания Flink для получения данных. Коннектор Flink CDC сначала читает полные исторические данные из исходной базы данных, затем бесшовно переключается на инкрементальное чтение и отправляет данные в flink-connector-selena.

   c. flink-connector-selena накапливает данные в мини-батчах и синхронизирует каждый батч данных в Selena.

> **К СВЕДЕНИЮ**
>
> Только операции языка манипулирования данными (DML) в MySQL могут быть синхронизированы в Selena. Операции языка определения данных (DDL) не могут быть синхронизированы.

3.10.2 Сценарии использования

Синхронизация в реальном времени из MySQL имеет широкий спектр применения, где данные постоянно изменяются. Возьмем в качестве примера реальный сценарий "ранжирование продаж товаров в реальном времени".

Flink рассчитывает ранжирование продаж товаров в реальном времени на основе исходной таблицы заказов в MySQL и синхронизирует рейтинг в таблицу Primary Key Selena в реальном времени. Пользователи могут подключить инструмент визуализации к Selena для просмотра рейтинга в реальном времени и получения оперативной информации по требованию.

3.10.3 Подготовка

Загрузка и установка инструментов синхронизации

Для синхронизации данных из MySQL необходимо установить следующие инструменты: SMT, Flink, коннектор Flink CDC и flink-connector-selena.

1. Загрузите и установите Flink, затем запустите cluster Flink. Вы также можете выполнить этот шаг, следуя инструкциям в официальной документации Flink.

   a. Установите Java 8 или Java 11 в вашей операционной системе перед запуском Flink. Вы можете выполнить следующую команду для проверки установленной версии Java.

   ```
   # Просмотр версии Java.
   java -version

   # Java 8 установлена, если возвращается следующий вывод.
   java version "1.8.0_301"
   Java(TM) SE Runtime Environment (build 1.8.0_301-b09)
   Java HotSpot(TM) 64-Bit Server VM (build 25.301-b09, mixed mode)
   ```

   b. Загрузите установочный пакет Flink и распакуйте его. Рекомендуется использовать Flink 1.14 или более поздней версии. Минимальная допустимая версия - Flink 1.11. В этой теме используется Flink 1.14.5.

   ```
   # Загрузка Flink.
   wget https://archive.apache.org/dist/flink/flink-1.14.5/flink-
1.14.5-bin-scala_2.11.tgz
   # Распаковка Flink.
   tar -xzf flink-1.14.5-bin-scala_2.11.tgz
   # Переход в каталог Flink.
   cd flink-1.14.5
   ```

   c. Запуск cluster Flink.

```
# Запуск cluster Flink.
./bin/start-cluster.sh

# Cluster Flink запущен, если возвращается следующий вывод.
Starting cluster.
Starting standalonesession daemon on host.
Starting taskexecutor daemon on host.
```

2. Загрузите коннектор Flink CDC. В этой теме MySQL используется в качестве источника данных, поэтому загружается `flink-sql-connector-mysql-cdc-x.x.x.jar`. Версия коннектора должна соответствовать версии Flink. В этой теме используется Flink 1.14.5, и вы можете загрузить `flink-sql-connector-mysql-cdc-2.2.0.jar`.

```
wget https://repo1.maven.org/maven2/com/ververica/flink-sql-connector-mysql-cdc/2.1.1/flink-sql-connector-mysql-cdc-2.2.0.jar
```

3. Загрузите flink-connector-selena. Версия должна соответствовать версии Flink.

> Пакет flink-connector-selena `x.x.x_flink-y.yy _ z.zz.jar` содержит три номера версий:
>
> - `x.x.x` - номер версии flink-connector-selena.
> - `y.yy` - поддерживаемая версия Flink.
> - `z.zz` - версия Scala, поддерживаемая Flink. Если версия Flink 1.14.x или более ранняя, необходимо загрузить пакет с указанием версии Scala.
>
> В этой теме используется Flink 1.14.5 и Scala 2.11. Следовательно, вы можете загрузить следующий пакет: `1.2.3_flink-14_2.11.jar`.

4. Переместите JAR пакеты коннектора Flink CDC (`flink-sql-connector-mysql-cdc-2.2.0.jar`) и flink-connector-selena (`1.2.3_flink-1.14_2.11.jar`) в каталог `lib` Flink.

> **Примечание**
>
> Если cluster Flink уже запущен в вашей системе, необходимо остановить cluster Flink и перезапустить его для загрузки и проверки JAR пакетов.

```
$ ./bin/stop-cluster.sh
$ ./bin/start-cluster.sh
```

5. Загрузите и распакуйте пакет SMT (обратитесь в техническую поддержку для получения дистрибутива) и поместите его в каталог `flink-1.14.5`. Selena предоставляет пакеты SMT для Linux x86 и macOS ARM64. Вы можете выбрать один в зависимости от вашей операционной системы и процессора.

```
# для Linux x86
wget https://releases.selena.io/resources/smt.tar.gz
# для macOS ARM64
wget https://releases.selena.io/resources/smt_darwin_arm64.tar.gz
```

Включение бинарного лога MySQL

Для синхронизации данных из MySQL в реальном времени система должна читать данные из бинарного лога MySQL (binlog), анализировать данные, а затем синхронизировать их в Selena. Убедитесь, что бинарный лог MySQL включен.

1. Отредактируйте конфигурационный файл MySQL `my.cnf` (путь по умолчанию: `/etc/my.cnf`) для включения бинарного лога MySQL.

```
# Включение MySQL Binlog.
log_bin = ON
# Настройка пути сохранения для Binlog.
log_bin =/var/lib/mysql/mysql-bin
# Настройка server_id.
# Если server_id не настроен для MySQL 5.7.3 или более поздней версии,
служба MySQL не может быть использована.
server_id = 1
# Установка формата Binlog в ROW.
binlog_format = ROW
# Базовое имя файла Binlog. Идентификатор добавляется для идентификации
каждого файла Binlog.
log_bin_basename =/var/lib/mysql/mysql-bin
# Индексный файл файлов Binlog, который управляет каталогом всех файлов
Binlog.
log_bin_index =/var/lib/mysql/mysql-bin.index
```

2. Выполните одну из следующих команд для перезапуска MySQL, чтобы измененный конфигурационный файл вступил в силу.

```
# Использование service для перезапуска MySQL.
service mysqld restart
# Использование скрипта mysqld для перезапуска MySQL.
/etc/init.d/mysqld restart
```

3. Подключитесь к MySQL и проверьте, включен ли бинарный лог MySQL.

```
-- Подключение к MySQL.
mysql -h xxx.xx.xxx.xx -P 3306 -u root -pxxxxxx

-- Проверка, включен ли бинарный лог MySQL.
mysql> SHOW VARIABLES LIKE 'log_bin';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_bin       | ON    |
+---------------+-------+
1 row in set (0.00 sec)
```

3.10.4 Синхронизация схемы базы данных и таблиц

1. Отредактируйте конфигурационный файл SMT. Перейдите в каталог SMT `conf` и отредактируйте конфигурационный файл `config_prod.conf`, такие как информация о подключении MySQL, правила сопоставления базы данных и таблиц для синхронизации и конфигурационная информация flink-connector-selena.

```
[db]
type = mysql
host = xxx.xx.xxx.xx
port = 3306
user = user1
password = xxxxxx

[other]
# Количество BE в Selena
be_num = 3
# `decimal_v3` поддерживается с Selena-1.18.1.
use_decimal_v3 = true
# Файл для сохранения преобразованного DDL SQL
output_dir = ./result

[table-rule.1]
# Шаблон для сопоставления баз данных для установки свойств
database = ^demo.*$
# Шаблон для сопоставления таблиц для установки свойств
table = ^.*$

############################################
### Конфигурации Flink sink
### НЕ устанавливайте `connector`, `table-name`, `database-name`. Они
генерируются автоматически.
############################################
flink.selena.jdbc-url=jdbc:mysql://<fe_host>:<fe_query_port>
flink.selena.load-url= <fe_host>:<fe_http_port>
flink.selena.username=user2
flink.selena.password=xxxxxx
flink.selena.sink.properties.format=csv
flink.selena.sink.properties.column_separator=\x01
flink.selena.sink.properties.row_delimiter=\x02
flink.selena.sink.buffer-flush.interval-ms=15000
```

- `[db]`: информация, используемая для доступа к исходной базе данных.
  - `type`: тип исходной базы данных. В этой теме исходная база данных - `mysql`.
  - `host`: IP-адрес сервера MySQL.
  - `port`: номер порта базы данных MySQL, по умолчанию `3306`
  - `user`: имя пользователя для доступа к базе данных MySQL
  - `password`: пароль имени пользователя

- `[table-rule]`: правила сопоставления базы данных и таблиц и соответствующая конфигурация flink-connector-selena.
  - `Database`, `table`: имена базы данных и таблицы в MySQL. Поддерживаются регулярные выражения.
  - `flink.selena.*`: конфигурационная информация flink-connector-selena. Для получения дополнительных конфигураций и информации см. flink-connector-selena.

  > Если вам необходимо использовать разные конфигурации flink-connector-selena для разных таблиц. Например, если некоторые таблицы часто обновляются и вы хотите ускорить загрузку данных, см. Использование разных конфигураций flink-connector-selena для разных таблиц. Если вам необходимо загрузить несколько таблиц, полученных из шардирования MySQL, в одну таблицу Selena, см. Синхронизация нескольких таблиц после шардирования MySQL в одну таблицу в Selena.

- `[other]`: другая информация
  - `be_num`: Количество BE в вашем cluster Selena (Этот параметр будет использоваться для установки разумного количества tablets при последующем создании таблицы Selena).
  - `use_decimal_v3`: Включить ли Decimal V3. После включения Decimal V3 данные decimal MySQL будут преобразованы в данные Decimal V3 при синхронизации данных в Selena.
  - `output_dir`: Путь для сохранения SQL файлов, которые будут сгенерированы. SQL файлы будут использоваться для создания базы данных и таблицы в Selena и отправки задания Flink в cluster Flink. Путь по умолчанию - `./result`, и рекомендуется сохранить настройки по умолчанию.

2. Запустите SMT для чтения схемы базы данных и таблиц в MySQL и генерации SQL файлов в каталоге `./result` на основе конфигурационного файла. Файл `selena-create.all.sql` используется для создания базы данных и таблицы в Selena, а файл `flink-create.all.sql` используется для отправки задания Flink в cluster Flink.

```
# Запуск SMT.
./selena-migrate-tool

# Переход в каталог result и проверка файлов в этом каталоге.
cd result
ls result
flink-create.1.sql    smt.tar.gz              selena-create.all.sql
flink-create.all.sql  selena-create.1.sql
```

3. Выполните следующую команду для подключения к Selena и выполнения файла `selena-create.all.sql` для создания базы данных и таблицы в Selena. Рекомендуется использовать оператор создания таблицы по умолчанию в SQL файле для создания таблицы Primary Key table.

> **Примечание**
>
> Вы также можете изменить оператор создания таблицы в соответствии с вашими бизнес-потребностями и создать таблицу, которая не использует Primary Key table. Однако операция DELETE в исходной базе данных MySQL не может быть синхронизирована с таблицей, не являющейся Primary Key table. Будьте осторожны при создании такой таблицы.

```
mysql -h <fe_host> -P <fe_query_port> -u user2 -pxxxxxx < selena-
create.all.sql
```

Если данные должны быть обработаны Flink перед записью в целевую таблицу Selena, схема таблицы будет отличаться между исходной и целевой таблицами. В этом случае необходимо изменить оператор создания таблицы. В этом примере целевая таблица требует только столбцы `product_id` и `product_name` и ранжирование продаж товаров в реальном времени. Вы можете использовать следующий оператор создания таблицы.

```
CREATE DATABASE IF NOT EXISTS `demo`;

CREATE TABLE IF NOT EXISTS `demo`.`orders` (
`product_id` INT(11) NOT NULL COMMENT "",
`product_name` STRING NOT NULL COMMENT "",
`sales_cnt` BIGINT NOT NULL COMMENT ""
) ENGINE=olap
PRIMARY KEY(`product_id`)
DISTRIBUTED BY HASH(`product_id`)
PROPERTIES (
"replication_num" = "3"
);
```

> **ВАЖНО**
>
> Начиная с v1.5.2, Selena может автоматически устанавливать количество buckets (BUCKETS) при создании таблицы или добавлении partition. Вам больше не нужно вручную устанавливать количество buckets. Подробную информацию см. в установка количества buckets.

3.10.5 Синхронизация данных

Запустите cluster Flink и отправьте задание Flink для непрерывной синхронизации полных и инкрементальных данных из MySQL в Selena.

1. Перейдите в каталог Flink и выполните следующую команду для запуска файла `flink-create.all.sql` на вашем SQL клиенте Flink.

   ```
   ./bin/sql-client.sh -f flink-create.all.sql
   ```

   Этот SQL файл определяет динамические таблицы `source table` и `sink table`, оператор запроса `INSERT INTO SELECT` и указывает коннектор, исходную базу данных и целевую базу данных. После выполнения этого файла задание Flink отправляется в cluster Flink для запуска синхронизации данных.

   > **Примечание**
   >
   > - Убедитесь, что cluster Flink был запущен. Вы можете запустить cluster Flink, выполнив `flink/bin/start-cluster.sh`.

> - Если ваша версия Flink более ранняя, чем 1.13, вы можете не иметь возможности напрямую запустить SQL файл `flink-create.all.sql`. Необходимо выполнять SQL операторы один за другим в этом файле в интерфейсе командной строки (CLI) SQL клиента. Вам также необходимо экранировать символ `\`.
>
> ```
> 'sink.properties.column_separator' = '\\x01'
> 'sink.properties.row_delimiter' = '\\x02'
> ```

Обработка данных во время синхронизации:

Если вам необходимо обработать данные во время синхронизации, такие как выполнение GROUP BY или JOIN на данных, вы можете изменить файл `flink-create.all.sql`. Следующий пример вычисляет ранжирование продаж товаров в реальном времени, выполняя COUNT (*) и GROUP BY.

```
    $ ./bin/sql-client.sh -f flink-create.all.sql
    No default environment is specified.
    Searching for '/home/disk1/flink-1.13.6/conf/sql-client-
defaults.yaml'...not found.
    [INFO] Executing SQL from file.

    Flink SQL> CREATE DATABASE IF NOT EXISTS `default_catalog`.`demo`;
    [INFO] Execute statement succeed.

    -- Создание динамической таблицы `source table` на основе таблицы
заказов в MySQL.
    Flink SQL>
    CREATE TABLE IF NOT EXISTS `default_catalog`.`demo`.`orders_src`
(`order_id` BIGINT NOT NULL,
    `product_id` INT NULL,
    `order_date` TIMESTAMP NOT NULL,
    `customer_name` STRING NOT NULL,
    `product_name` STRING NOT NULL,
    `price` DECIMAL(10, 5) NULL,
    PRIMARY KEY(`order_id`)
    NOT ENFORCED
    ) with ('connector' = 'mysql-cdc',
    'hostname' = 'xxx.xx.xxx.xxx',
    'port' = '3306',
    'username' = 'root',
    'password' = '',
    'database-name' = 'demo',
    'table-name' = 'orders'
    );
    [INFO] Execute statement succeed.

    -- Создание динамической таблицы `sink table`.
    Flink SQL>
    CREATE TABLE IF NOT EXISTS `default_catalog`.`demo`.`orders_sink`
(`product_id` INT NOT NULL,
    `product_name` STRING NOT NULL,
    `sales_cnt` BIGINT NOT NULL,
    PRIMARY KEY(`product_id`)
    NOT ENFORCED
    ) with ('sink.max-retries' = '10',
    'jdbc-url' = 'jdbc:mysql://<fe_host>:<fe_query_port>',
    'password' = '',
    'sink.properties.strip_outer_array' = 'true',
    'sink.properties.format' = 'json',
    'load-url' = '<fe_host>:<fe_http_port>',
```

```
    'username' = 'root',
    'sink.buffer-flush.interval-ms' = '15000',
    'connector' = 'selena',
    'database-name' = 'demo',
    'table-name' = 'orders'
    );
    [INFO] Execute statement succeed.

    -- Реализация ранжирования продаж товаров в реальном времени, где
`sink table` динамически обновляется для отражения изменений данных в
`source table`.
    Flink SQL>
    INSERT INTO `default_catalog`.`demo`.`orders_sink` select
product_id,product_name, count(*) as cnt from
`default_catalog`.`demo`.`orders_src` group by product_id,product_name;
    [INFO] Submitting SQL update statement to the cluster...
    [INFO] SQL update statement has been successfully submitted to the
cluster:
    Job ID: 5ae005c4b3425d8bb13fe660260a35da
```

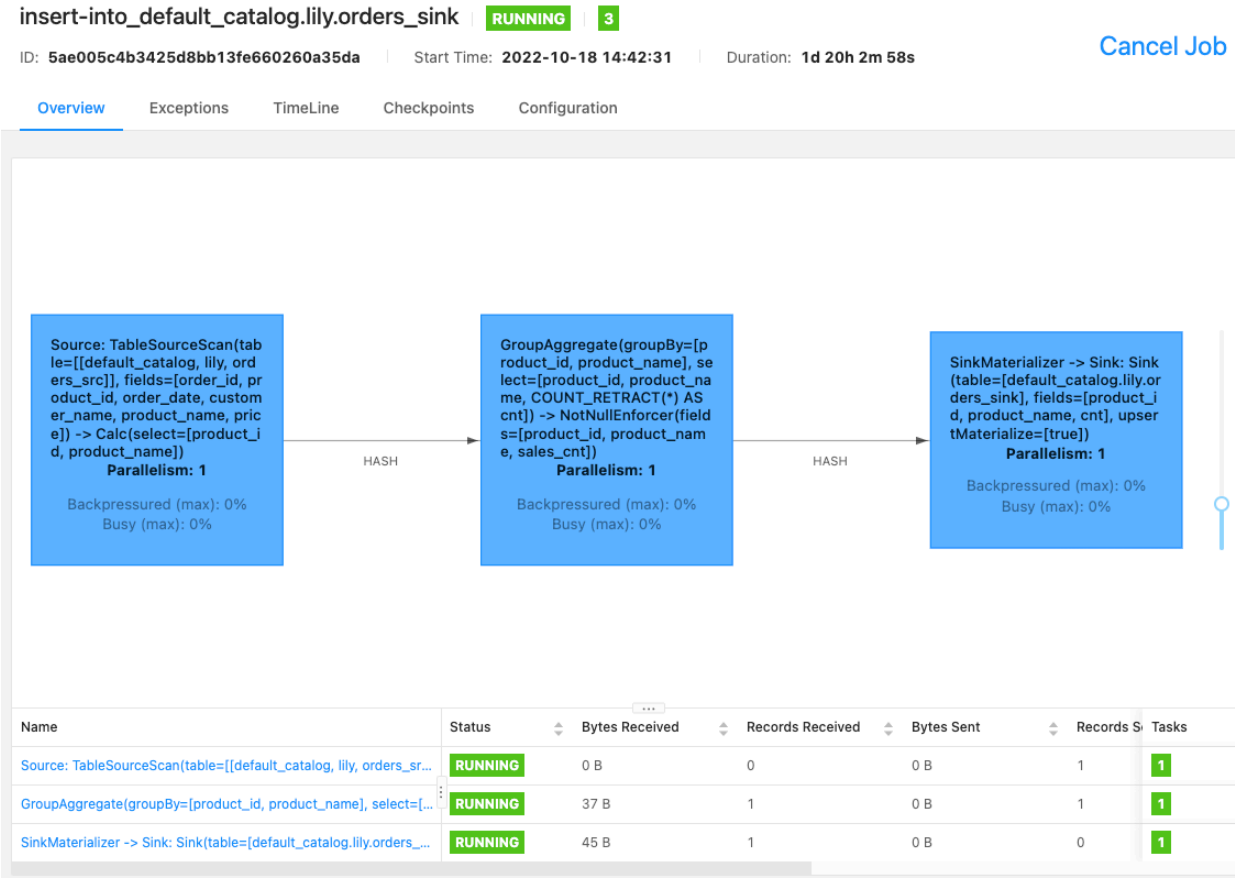
Если вам необходимо синхронизировать только часть данных, таких как данные, время оплаты которых позже 21 декабря 2021 года, вы можете использовать предложение `WHERE` в `INSERT INTO SELECT` для установки условия фильтрации, например `WHERE pay_dt > '2021-12-21'`. Данные, которые не соответствуют этому условию, не будут синхронизированы в Selena.

Если возвращается следующий результат, задание Flink было отправлено для полной и инкрементальной синхронизации.

```
[INFO] Submitting SQL update statement to the cluster...
[INFO] SQL update statement has been successfully submitted to the
cluster:
Job ID: 5ae005c4b3425d8bb13fe660260a35da
```

2. Вы можете использовать Flink WebUI или выполнить команду `bin/flink list -running` на вашем SQL клиенте Flink для просмотра заданий Flink, которые выполняются в cluster Flink, и идентификаторов заданий.

- Flink WebUI



- `bin/flink list -running`

```
    $ bin/flink list -running
    Waiting for response...
    ------------------ Running/Restarting Jobs -------------------
    13.10.2022 15:03:54 : 040a846f8b58e82eb99c8663424294d5 : insert-
into_default_catalog.lily.example_tbl1_sink (RUNNING)
    --------------------------------------------------------------
```

> **Примечание**
>
> Если задание является аномальным, вы можете выполнить устранение неполадок, используя Flink WebUI или просмотрев файл лога в каталоге `/log` Flink 1.14.5.

3.10.6 FAQ

Использование разных конфигураций flink-connector-selena для разных таблиц

Если некоторые таблицы в источнике данных часто обновляются и вы хотите ускорить скорость загрузки flink-connector-selena, необходимо установить отдельную конфигурацию flink-connector-selena для каждой таблицы в конфигурационном файле SMT `config_prod.conf`.

```
[table-rule.1]
# Шаблон для сопоставления баз данных для установки свойств
database = ^order.*$
# Шаблон для сопоставления таблиц для установки свойств
table = ^.*$

#############################################
### Конфигурации Flink sink
### НЕ устанавливайте `connector`, `table-name`, `database-name`. Они
генерируются автоматически
#############################################
flink.selena.jdbc-url=jdbc:mysql://<fe_host>:<fe_query_port>
flink.selena.load-url= <fe_host>:<fe_http_port>
flink.selena.username=user2
flink.selena.password=xxxxxx
flink.selena.sink.properties.format=csv
flink.selena.sink.properties.column_separator=\x01
flink.selena.sink.properties.row_delimiter=\x02
flink.selena.sink.buffer-flush.interval-ms=15000

[table-rule.2]
# Шаблон для сопоставления баз данных для установки свойств
database = ^order2.*$
# Шаблон для сопоставления таблиц для установки свойств
table = ^.*$

#############################################
### Конфигурации Flink sink
### НЕ устанавливайте `connector`, `table-name`, `database-name`. Они
генерируются автоматически
#############################################
flink.selena.jdbc-url=jdbc:mysql://<fe_host>:<fe_query_port>
flink.selena.load-url= <fe_host>:<fe_http_port>
flink.selena.username=user2
flink.selena.password=xxxxxx
flink.selena.sink.properties.format=csv
flink.selena.sink.properties.column_separator=\x01
flink.selena.sink.properties.row_delimiter=\x02
flink.selena.sink.buffer-flush.interval-ms=10000
```

Синхронизация нескольких таблиц после шардирования MySQL в одну таблицу в Selena

После выполнения шардирования данные в одной таблице MySQL могут быть разделены на несколько таблиц или даже распределены по нескольким базам данных. Все таблицы имеют одинаковую схему. В этом случае вы можете установить `[table-rule]` для синхронизации этих таблиц в одну таблицу Selena. Например, MySQL имеет две базы данных `edu_db_1` и `edu_db_2`, каждая из которых имеет две таблицы `course_1 и course_2`, и схема всех таблиц одинакова. Вы можете использовать следующую конфигурацию `[table-rule]` для синхронизации всех таблиц в одну таблицу Selena.

> **Примечание**
>
> Имя таблицы Selena по умолчанию - `course__auto_shard`. Если вам необходимо использовать другое имя, вы можете изменить его в SQL файлах `selena-create.all.sql` и `flink-create.all.sql`

```
[table-rule.1]
# Шаблон для сопоставления баз данных для установки свойств
database = ^edu_db_[0-9]*$
# Шаблон для сопоставления таблиц для установки свойств
table = ^course_[0-9]*$

#############################################
### Конфигурации Flink sink
### НЕ устанавливайте `connector`, `table-name`, `database-name`. Они
генерируются автоматически
#############################################
flink.selena.jdbc-url = jdbc: mysql://xxx.xxx.x.x:xxxx
flink.selena.load-url = xxx.xxx.x.x:xxxx
flink.selena.username = user2
flink.selena.password = xxxxxx
flink.selena.sink.properties.format=csv
flink.selena.sink.properties.column_separator =\x01
flink.selena.sink.properties.row_delimiter =\x02
flink.selena.sink.buffer-flush.interval-ms = 5000
```

Импорт данных в формате JSON

Данные в предыдущем примере импортируются в формате CSV. Если вы не можете выбрать подходящий разделитель, необходимо заменить следующие параметры

`flink.selena.*` в `[table-rule]`.

```
flink.selena.sink.properties.format=csv
flink.selena.sink.properties.column_separator =\x01
flink.selena.sink.properties.row_delimiter =\x02
```

Данные импортируются в формате JSON после передачи следующих параметров.

```
flink.selena.sink.properties.format=json
flink.selena.sink.properties.strip_outer_array=true
```

> **Примечание**
>
> Этот метод немного замедляет скорость загрузки.

Выполнение нескольких операторов INSERT INTO как одного задания Flink

Вы можете использовать синтаксис STATEMENT SET в файле `flink-create.all.sql` для выполнения нескольких операторов INSERT INTO как одного задания Flink, что предотвращает использование слишком большого количества ресурсов задания Flink несколькими операторами и повышает эффективность выполнения нескольких запросов.

> **Примечание**
>
> Flink поддерживает синтаксис STATEMENT SET начиная с версии 1.13.

1. Откройте файл `result/flink-create.all.sql`.
2. Измените SQL операторы в файле. Переместите все операторы INSERT INTO в конец файла. Поместите `EXECUTE STATEMENT SET BEGIN` перед первым оператором INSERT INTO и поместите `END;` после последнего оператора INSERT INTO.

> **Примечание**
>
> Позиции CREATE DATABASE и CREATE TABLE остаются неизменными.

```
CREATE DATABASE IF NOT EXISTS db;
CREATE TABLE IF NOT EXISTS db.a1;
CREATE TABLE IF NOT EXISTS db.b1;
CREATE TABLE IF NOT EXISTS db.a2;
CREATE TABLE IF NOT EXISTS db.b2;
EXECUTE STATEMENT SET
BEGIN-- один или несколько операторов INSERT INTO
INSERT INTO db.a1 SELECT * FROM db.b1;
INSERT INTO db.a2 SELECT * FROM db.b2;
END;
```

3.11 Непрерывная загрузка данных из Apache Flink®

Selena предоставляет самостоятельно разработанный connector под названием Selena Connector for Apache Flink® (сокращённо Flink connector) для загрузки данных в таблицу Selena с использованием Flink. Основной принцип заключается в накоплении данных с последующей загрузкой всех данных за один раз в Selena через STREAM LOAD.

Flink connector поддерживает DataStream API, Table API & SQL и Python API. Он имеет более высокую и стабильную производительность по сравнению с flink-connector-jdbc, предоставляемым Apache Flink®.

> **ВНИМАНИЕ**
>
> Загрузка данных в таблицы Selena с помощью Flink connector требует привилегий SELECT и INSERT на целевую таблицу Selena. Если у вас нет этих привилегий, следуйте инструкциям, представленным в GRANT, чтобы предоставить эти привилегии пользователю, которого вы используете для подключения к вашему cluster Selena.

3.11.1 Требования к версиям

Connector	Flink	Selena	Java	Scala
1.2.11	1.15,1.16,1.17,1.18,1.19,1.20	2.1 и позже	8	2.11,2.12
1.2.10	1.15,1.16,1.17,1.18,1.19	2.1 и позже	8	2.11,2.12
1.2.9	1.15,1.16,1.17,1.18	2.1 и позже	8	2.11,2.12
1.2.8	1.13,1.14,1.15,1.16,1.17	2.1 и позже	8	2.11,2.12
1.2.7	1.11,1.12,1.13,1.14,1.15	2.1 и позже	8	2.11,2.12

3.11.2 Получение Flink connector

Вы можете получить JAR-файл Flink connector следующими способами:

- Напрямую скачать скомпилированный JAR-файл Flink connector.
- Добавить Flink connector в качестве зависимости в ваш проект Maven, а затем скачать JAR-файл.
- Скомпилировать исходный код Flink connector в JAR-файл самостоятельно.

Формат наименования JAR-файла Flink connector следующий:

- Начиная с Flink 1.15, это `flink-connector-selena-${connector_version}_flink-${flink_version}.jar`. Например, если вы установили Flink 1.15 и хотите использовать Flink connector 1.2.7, вы можете использовать `flink-connector-selena-1.2.7_flink-1.15.jar`.

- До Flink 1.15 это `flink-connector-selena-${connector_version}_flink-${flink_version}_${scala_version}.jar`. Например, если вы установили Flink 1.14 и Scala 2.12 в вашей среде, и хотите использовать Flink connector 1.2.7, вы можете использовать `flink-connector-selena-1.2.7_flink-1.14_2.12.jar`.

> **ВНИМАНИЕ**
>
> В общем случае последняя версия Flink connector поддерживает совместимость только с тремя последними версиями Flink.

Скачивание скомпилированного Jar-файла

Напрямую скачайте соответствующую версию JAR-файла Flink connector из Maven Central Repository.

Maven Dependency

В файле `pom.xml` вашего проекта Maven добавьте Flink connector в качестве зависимости в соответствии со следующим форматом. Замените `flink_version`, `scala_version` и `connector_version` на соответствующие версии.

- В Flink 1.15 и позже

```
<dependency>
    <groupId>com.selena</groupId>
    <artifactId>flink-connector-selena</artifactId>
    <version>${connector_version}_flink-${flink_version}</version>
</dependency>
```

- В версиях до Flink 1.15

```
<dependency>
    <groupId>com.selena</groupId>
    <artifactId>flink-connector-selena</artifactId>

<version>${connector_version}_flink-${flink_version}_${scala_version}
</version>
</dependency>
```

Компиляция самостоятельно

1. Скачайте исходный код Flink connector.

2. Выполните следующую команду для компиляции исходного кода Flink connector в JAR-файл. Обратите внимание, что `flink_version` заменяется на соответствующую версию Flink.

   ```
   sh build.sh <flink_version>
   ```

   Например, если версия Flink в вашей среде 1.15, вам нужно выполнить следующую команду:

   ```
   sh build.sh 1.15
   ```

3. Перейдите в каталог `target/`, чтобы найти JAR-файл Flink connector, такой как `flink-connector-selena-1.2.7_flink-1.15-SNAPSHOT.jar`, сгенерированный после компиляции.

> **ПРИМЕЧАНИЕ**

> Имя Flink connector, который не выпущен официально, содержит суффикс `SNAPSHOT`.

3.11.3 Параметры

connector

Обязательный: Да
Значение по умолчанию: NONE
Описание: Connector, который вы хотите использовать. Значение должно быть "selena".

jdbc-url

Обязательный: Да
Значение по умолчанию: NONE
Описание: Адрес, который используется для подключения к серверу MySQL FE. Вы можете указать несколько адресов, которые должны быть разделены запятой (,). Формат: `jdbc:mysql://<fe_host1>:<fe_query_port1>,<fe_host2>:<fe_query_port2>,<fe_host3>:<fe_query_port3>`.

load-url

Обязательный: Да
Значение по умолчанию: NONE
Описание: Адрес, который используется для подключения к HTTP-серверу FE. Вы можете указать несколько адресов, которые должны быть разделены точкой с запятой (;). Формат: `<fe_host1>:<fe_http_port1>;<fe_host2>:<fe_http_port2>`.

database-name

Обязательный: Да
Значение по умолчанию: NONE
Описание: Имя базы данных Selena, в которую вы хотите загрузить данные.

table-name

Обязательный: Да
Значение по умолчанию: NONE
Описание: Имя таблицы, которую вы хотите использовать для загрузки данных в Selena.

username

Обязательный: Да
Значение по умолчанию: NONE
Описание: Имя пользователя учетной записи, которую вы хотите использовать для загрузки данных в Selena. Учетная запись должна иметь привилегии SELECT и INSERT на целевую таблицу Selena.

password

Обязательный: Да
Значение по умолчанию: NONE
Описание: Пароль предыдущей учетной записи.

sink.version

Обязательный: Нет
Значение по умолчанию: AUTO
Описание: Интерфейс, используемый для загрузки данных. Этот параметр поддерживается начиная с версии Flink connector 1.2.4.

- `V1`: Использовать интерфейс Stream Load для загрузки данных. Connectors до 1.2.4 поддерживают только этот режим.
- `V2`: Использовать интерфейс Stream Load transaction для загрузки данных. Требуется Selena версии не ниже 2.4. Рекомендуется `V2`, так как он оптимизирует использование памяти и обеспечивает более стабильную реализацию exactly-once.
- `AUTO`: Если версия Selena поддерживает транзакционный Stream Load, автоматически выберет `V2`, иначе выберет `V1`

sink.label-prefix

Обязательный: Нет
Значение по умолчанию: NONE
Описание: Префикс метки, используемый Stream Load. Рекомендуется настроить его, если вы используете exactly-once с connector 1.2.8 и позже. См. примечания по использованию exactly-once.

sink.semantic

Обязательный: Нет
Значение по умолчанию: at-least-once
Описание: Семантика, гарантируемая sink. Допустимые значения: **at-least-once** и **exactly-once**.

sink.buffer-flush.max-bytes

Обязательный: Нет
Значение по умолчанию: 94371840(90M)
Описание: Максимальный размер данных, которые могут быть накоплены в памяти перед отправкой в Selena за один раз. Максимальное значение варьируется от 64 МБ до 10 ГБ. Установка этого параметра на большее значение может улучшить производительность загрузки, но может увеличить задержку загрузки. Этот параметр вступает в силу только тогда, когда `sink.semantic` установлен на `at-least-once`. Если `sink.semantic` установлен на `exactly-once`, данные в памяти сбрасываются при запуске checkpoint Flink. В этом случае этот параметр не действует.

sink.buffer-flush.max-rows

Обязательный: Нет
Значение по умолчанию: 500000
Описание: Максимальное количество строк, которые могут быть накоплены в памяти перед отправкой в Selena за один раз. Этот параметр доступен только тогда, когда `sink.version` равен `V1`, а `sink.semantic` равен `at-least-once`. Допустимые значения: от 64000 до 5000000.

sink.buffer-flush.interval-ms

Обязательный: Нет
Значение по умолчанию: 300000
Описание: Интервал сброса данных. Этот параметр доступен только тогда, когда `sink.semantic` равен `at-least-once`. Допустимые значения: от 1000 до 3600000. Единица измерения: мс.

sink.max-retries

Обязательный: Нет
Значение по умолчанию: 3

Описание: Количество раз, когда система повторяет попытку выполнить задачу Stream Load. Этот параметр доступен только тогда, когда вы устанавливаете `sink.version` в `V1`. Допустимые значения: от 0 до 10.

sink.connect.timeout-ms

Обязательный: Нет
Значение по умолчанию: 30000
Описание: Тайм-аут для установки HTTP-соединения. Допустимые значения: от 100 до 60000. Единица измерения: мс. До Flink connector v1.2.9 значение по умолчанию было `1000`.

sink.socket.timeout-ms

Обязательный: Нет
Значение по умолчанию: -1
Описание: Поддерживается с версии 1.2.10. Продолжительность времени, в течение которого HTTP-клиент ожидает данных. Единица измерения: мс. Значение по умолчанию `-1` означает, что тайм-аута нет.

sink.sanitize-error-log

Обязательный: Нет
Значение по умолчанию: false
Описание: Поддерживается с версии 1.2.12. Следует ли очищать конфиденциальные данные в журнале ошибок для производственной безопасности. Когда этот элемент установлен в `true`, конфиденциальные данные строк и значения столбцов в журналах ошибок Stream Load редактируются как в журналах connector, так и в журналах SDK. Значение по умолчанию - `false` для обратной совместимости.

sink.wait-for-continue.timeout-ms

Обязательный: Нет
Значение по умолчанию: 10000
Описание: Поддерживается с версии 1.2.7. Тайм-аут ожидания ответа HTTP 100-continue от FE. Допустимые значения: от `3000` до `60000`. Единица измерения: мс

sink.ignore.update-before

Обязательный: Нет
Значение по умолчанию: true
Описание: Поддерживается с версии 1.2.8. Следует ли игнорировать записи `UPDATE_BEFORE` из Flink при загрузке данных в таблицы Primary Key. Если этот параметр установлен в false, запись рассматривается как операция удаления в таблице Selena.

sink.parallelism

Обязательный: Нет
Значение по умолчанию: NONE
Описание: Параллелизм загрузки. Доступно только для Flink SQL. Если этот параметр не указан, планировщик Flink определяет параллелизм. **В сценарии с мультипараллелизмом пользователи должны гарантировать, что данные записываются в правильном порядке.**

sink.properties.*

Обязательный: Нет
Значение по умолчанию: NONE
Описание: Параметры, которые используются для управления поведением Stream Load. Например, параметр `sink.properties.format` указывает формат, используемый для Stream Load, такой как CSV или JSON. Список поддерживаемых параметров и их описания см. в STREAM LOAD.

sink.properties.format

Обязательный: Нет
Значение по умолчанию: csv
Описание: Формат, используемый для Stream Load. Flink connector преобразует каждый пакет данных в формат перед их отправкой в Selena. Допустимые значения: `csv` и `json`.

sink.properties.column_separator

Обязательный: Нет
Значение по умолчанию: \t
Описание: Разделитель столбцов для данных в формате CSV.

sink.properties.row_delimiter

Обязательный: Нет
Значение по умолчанию: \n
Описание: Разделитель строк для данных в формате CSV.

sink.properties.max_filter_ratio

Обязательный: Нет
Значение по умолчанию: 0
Описание: Максимальная толерантность к ошибкам Stream Load. Это максимальный процент записей данных, которые могут быть отфильтрованы из-за недостаточного качества данных. Допустимые значения: от `0` до `1`. Значение по умолчанию: `0`. Подробности см. в Stream Load.

sink.properties.partial_update

Обязательный: НЕТ
Значение по умолчанию: `FALSE`
Описание: Следует ли использовать частичные обновления. Допустимые значения: `TRUE` и `FALSE`. Значение по умолчанию: `FALSE`, указывающее на отключение этой функции.

sink.properties.partial_update_mode

Обязательный: НЕТ
Значение по умолчанию: `row`
Описание: Указывает режим частичных обновлений. Допустимые значения: `row` и `column`.

- Значение `row` (по умолчанию) означает частичные обновления в режиме строк, что более подходит для обновлений в реальном времени с большим количеством столбцов и небольшими пакетами.
- Значение `column` означает частичные обновления в режиме столбцов, что более подходит для пакетных обновлений с небольшим количеством столбцов и большим количеством строк. В таких сценариях включение режима столбцов обеспечивает более быструю скорость обновления. Например, в таблице с 100 столбцами, если обновляются только 10 столбцов (10% от общего числа) для всех строк, скорость обновления в режиме столбцов в 10 раз быстрее.

sink.properties.strict_mode

Обязательный: Нет
Значение по умолчанию: false
Описание: Указывает, следует ли включать строгий режим для Stream Load. Это влияет на поведение загрузки при наличии неквалифицированных строк, таких как несогласованные значения столбцов. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`. Подробности см. в Stream Load.

sink.properties.compression

Обязательный: Нет
Значение по умолчанию: NONE
Описание: Алгоритм сжатия, используемый для Stream Load. Допустимые значения: `lz4_frame`. Сжатие для формата JSON требует Flink connector 1.2.10+ и Selena v1.5.2+. Сжатие для формата CSV требует только Flink connector 1.2.11+.

sink.properties.prepared_timeout

Обязательный: Нет
Значение по умолчанию: NONE
Описание: Поддерживается с версии 1.2.12 и действует только тогда, когда `sink.version` установлен в `V2`. Требуется Selena 3.5.4 или позже. Устанавливает тайм-аут в секундах для фазы Transaction Stream Load от `PREPARED` до `COMMITTED`. Обычно требуется только для exactly-once; at-least-once обычно не требует установки этого (connector по умолчанию использует 300 с). Если не установлено в exactly-once, применяется конфигурация Selena FE `prepared_transaction_default_timeout_second` (по умолчанию 86400 с). См. управление тайм-аутом транзакций Selena.

3.11.4 Сопоставление типов данных между Flink и Selena

Тип данных Flink	Тип данных Selena
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INTEGER	INTEGER
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE

Тип данных Flink	Тип данных Selena
DECIMAL	DECIMAL
BINARY	INT
CHAR	STRING
VARCHAR	STRING
STRING	STRING
DATE	DATE
TIMESTAMP_WITHOUT_TIME_ZONE(N)	DATETIME
TIMESTAMP_WITH_LOCAL_TIME_ZONE(N)	DATETIME
ARRAY<T>	ARRAY<T>
MAP<KT,VT>	JSON STRING
ROW<arg T...>	JSON STRING

3.11.5 Примечания по использованию

Exactly Once

- Если вы хотите, чтобы sink гарантировал семантику exactly-once, мы рекомендуем вам обновить Selena до версии 2.5 или позже, а Flink connector до версии 1.2.4 или позже
    - Начиная с Flink connector 1.2.4, exactly-once был переработан на основе интерфейса транзакций Stream Load, предоставленного Selena с версии 2.4. По сравнению с предыдущей реализацией на основе нетранзакционного интерфейса Stream Load, новая реализация снижает использование памяти и накладные расходы на checkpoint, тем самым повышая производительность в реальном времени и стабильность загрузки.
    - Если версия Selena ранее 2.4 или версия Flink connector ранее 1.2.4, sink автоматически выберет реализацию на основе нетранзакционного интерфейса Stream Load.
- Конфигурации для гарантии exactly-once
    - Значение `sink.semantic` должно быть `exactly-once`.
    - Если версия Flink connector 1.2.8 и позже, рекомендуется указать значение `sink.label-prefix`. Обратите внимание, что префикс метки должен быть

уникальным среди всех типов загрузки в Selena, таких как задачи Flink, Routine Load и Broker Load.

  - Если префикс метки указан, Flink connector будет использовать префикс метки для очистки оставшихся транзакций, которые могут быть сгенерированы в некоторых сценариях сбоя Flink, таких как сбой задачи Flink, когда checkpoint все еще выполняется. Эти оставшиеся транзакции обычно находятся в статусе `PREPARED`, если вы используете `SHOW PROC '/transactions/<db_id>/running';` для их просмотра в Selena. Когда задача Flink восстанавливается из checkpoint, Flink connector найдет эти оставшиеся транзакции в соответствии с префиксом метки и некоторой информацией в checkpoint и прервет их. Flink connector не может прервать их при выходе задачи Flink из-за механизма двухфазной фиксации для реализации exactly-once. Когда задача Flink завершается, Flink connector еще не получил уведомление от координатора checkpoint Flink о том, должны ли транзакции быть включены в успешный checkpoint, и это может привести к потере данных, если эти транзакции будут прерваны в любом случае. Вы можете получить обзор о том, как достичь сквозного exactly-once в Flink в этом blogpost.

  - Если префикс метки не указан, оставшиеся транзакции будут очищены Selena только после их истечения. Однако количество выполняющихся транзакций может достичь ограничения Selena `max_running_txn_num_per_db`, если задачи Flink часто терпят неудачу до истечения времени ожидания транзакций. Вы можете установить меньший тайм-аут для транзакций `PREPARED` чтобы они истекали быстрее, когда префикс метки не указан. См. ниже о том, как установить подготовленный тайм-аут.

- Если вы уверены, что задача Flink в конечном итоге восстановится из checkpoint или savepoint после длительного простоя из-за остановки или непрерывного отказа, пожалуйста, соответствующим образом настройте следующие конфигурации Selena, чтобы избежать потери данных.

  - Настройте тайм-аут транзакций `PREPARED`. См. ниже о том, как установить тайм-аут.

Тайм-аут должен быть больше времени простоя задачи Flink. В противном случае оставшиеся транзакции, которые включены в успешный checkpoint, могут быть прерваны из-за тайм-аута до перезапуска задачи Flink, что приводит к потере данных.

Обратите внимание, что когда вы устанавливаете большее значение для этой конфигурации, лучше указать значение `sink.label-prefix`, чтобы оставшиеся транзакции могли быть очищены в соответствии с префиксом метки и некоторой информацией в checkpoint, вместо того чтобы из-за тайм-аута (что может привести к потере данных).

- `label_keep_max_second` и `label_keep_max_num`: конфигурации Selena FE, значения по умолчанию - `259200` и `1000` соответственно. Подробности см. в конфигурациях FE. Значение `label_keep_max_second` должно быть больше времени простоя задачи Flink. В противном случае Flink connector не сможет проверить состояние транзакций в Selena, используя метки транзакций, сохраненные в savepoint или checkpoint Flink, и выяснить, зафиксированы ли эти транзакции или нет, что в конечном итоге может привести к потере данных.

- Как установить тайм-аут для транзакций PREPARED
  - Для Connector 1.2.12+ и Selena 3.5.4+ вы можете установить тайм-аут, настроив параметр connector `sink.properties.prepared_timeout`. По умолчанию значение не установлено, и оно возвращается к глобальной конфигурации Selena FE `prepared_transaction_default_timeout_second` (значение по умолчанию - `86400`).
  - Для других версий Connector или Selena вы можете установить тайм-аут, настроив глобальную конфигурацию Selena FE `prepared_transaction_default_timeout_second` (значение по умолчанию - `86400`).

Политика сброса

Flink connector будет буферизовать данные в памяти и сбрасывать их пакетами в Selena через Stream Load. Способ запуска сброса отличается между at-least-once и exactly-once.

Для at-least-once сброс будет запущен, когда выполняется любое из следующих условий:

- количество байтов буферизованных строк достигает предела `sink.buffer-flush.max-bytes`
- количество буферизованных строк достигает предела `sink.buffer-flush.max-rows`. (Действительно только для версии sink V1)
- время, прошедшее с момента последнего сброса, достигает предела `sink.buffer-flush.interval-ms`
- запущен checkpoint

Для exactly-once сброс происходит только при запуске checkpoint.

Мониторинг метрик загрузки

Flink connector предоставляет следующие метрики для мониторинга загрузки.

Метрика	Тип	Описание
totalFlushBytes	counter	успешно сброшенные байты.
totalFlushRows	counter	количество успешно сброшенных строк.
totalFlushSucceededTimes	counter	количество раз, когда данные были успешно сброшены.
totalFlushFailedTimes	counter	количество раз, когда сброс данных не удался.
totalFilteredRows	counter	количество отфильтрованных строк, которое также включено в totalFlushRows.

3.11.6 Примеры

Следующие примеры показывают, как использовать Flink connector для загрузки данных в таблицу Selena с помощью Flink SQL или Flink DataStream.

Подготовка

Создание таблицы Selena

Создайте базу данных `test` и создайте таблицу Primary Key `score_board`.

```
CREATE DATABASE `test`;

CREATE TABLE `test`.`score_board`
(
    `id` int(11) NOT NULL COMMENT "",
    `name` varchar(65533) NULL DEFAULT "" COMMENT "",
    `score` int(11) NOT NULL DEFAULT "0" COMMENT ""
)
ENGINE=OLAP
PRIMARY KEY(`id`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`id`);
```

Настройка среды Flink

- Скачайте бинарный файл Flink Flink 1.15.2 и распакуйте его в каталог `flink-1.15.2`.
- Скачайте Flink connector 1.2.7 и поместите его в каталог `flink-1.15.2/lib`.
- Выполните следующие команды для запуска cluster Flink:

  ```
  cd flink-1.15.2
  ./bin/start-cluster.sh
  ```

Конфигурация сети

Убедитесь, что машина, на которой находится Flink, может получить доступ к узлам FE cluster Selena через `http_port` (по умолчанию: `8030`) и `query_port` (по умолчанию: `9030`), а к узлам BE через `be_http_port` (по умолчанию: `8040`).

Запуск с Flink SQL

- Выполните следующую команду для запуска клиента Flink SQL.

  ```
  ./bin/sql-client.sh
  ```

- Создайте таблицу Flink `score_board` и вставьте значения в таблицу через Flink SQL Client. Обратите внимание, что вы должны определить первичный ключ в Flink DDL,

если хотите загрузить данные в таблицу Primary Key Selena. Это необязательно для других типов таблиц Selena.

```
CREATE TABLE `score_board` (
    `id` INT,
    `name` STRING,
    `score` INT,
    PRIMARY KEY (id) NOT ENFORCED
) WITH (
    'connector' = 'selena',
    'jdbc-url' = 'jdbc:mysql://127.0.0.1:9030',
    'load-url' = '127.0.0.1:8030',
    'database-name' = 'test',

    'table-name' = 'score_board',
    'username' = 'root',
    'password' = ''
);

INSERT INTO `score_board` VALUES (1, 'selena', 100), (2, 'flink', 100);
```

Запуск с Flink DataStream

Существует несколько способов реализации задачи Flink DataStream в зависимости от типа входных записей, таких как CSV Java `String`, JSON Java `String` или пользовательский объект Java.

- Входные записи представляют собой `String` в формате CSV. Полный пример см. в LoadCsvRecords.

```
/**
 * Генерация записей в формате CSV. Каждая запись имеет три значения,
разделенных "\t".
 * Эти значения будут загружены в столбцы `id`, `name` и `score` в
таблице Selena.
 */
String[] records = new String[]{
        "1\tselena-csv\t100",
        "2\tflink-csv\t100"
};
DataStream<String> source = env.fromElements(records);

/**
 * Настройте connector с необходимыми свойствами.
 * Вам также необходимо добавить свойства "sink.properties.format" и
"sink.properties.column_separator"
 * чтобы сообщить connector, что входные записи в формате CSV, а
разделитель столбцов - "\t".
 * Вы также можете использовать другие разделители столбцов в записях
формата CSV,
 * но не забудьте соответственно изменить
"sink.properties.column_separator".
 */
SelenaSinkOptions options = SelenaSinkOptions.builder()
        .withProperty("jdbc-url", jdbcUrl)
        .withProperty("load-url", loadUrl)
        .withProperty("database-name", "test")
        .withProperty("table-name", "score_board")
        .withProperty("username", "root")
        .withProperty("password", "")
        .withProperty("sink.properties.format", "csv")
        .withProperty("sink.properties.column_separator", "\t")
        .build();
// Создайте sink с параметрами.
SinkFunction<String> selenaink = SelenaSink.sink(options);
source.addSink(selenaink);
```

- Входные записи представляют собой `String` в формате JSON. Полный пример см. в LoadJsonRecords.

```
/**
 * Генерация записей в формате JSON.
 * Каждая запись имеет три пары ключ-значение, соответствующие столбцам
`id`, `name` и `score` в таблице Selena.
 */
String[] records = new String[]{
        "{\"id\":1, \"name\":\"selena-json\", \"score\":100}",
        "{\"id\":2, \"name\":\"flink-json\", \"score\":100}",
};
DataStream<String> source = env.fromElements(records);

/**
 * Настройте connector с необходимыми свойствами.
 * Вам также необходимо добавить свойства "sink.properties.format" и
"sink.properties.strip_outer_array"
 * чтобы сообщить connector, что входные записи в формате JSON и нужно
удалить внешнюю структуру массива.
 */
SelenaSinkOptions options = SelenaSinkOptions.builder()
        .withProperty("jdbc-url", jdbcUrl)
        .withProperty("load-url", loadUrl)
        .withProperty("database-name", "test")
        .withProperty("table-name", "score_board")
        .withProperty("username", "root")
        .withProperty("password", "")
        .withProperty("sink.properties.format", "json")
        .withProperty("sink.properties.strip_outer_array", "true")
        .build();
// Создайте sink с параметрами.
SinkFunction<String> selenaink = SelenaSink.sink(options);
source.addSink(selenaink);
```

- Входные записи представляют собой пользовательские объекты Java. Полный пример см. в LoadCustomJavaRecords.
    - В этом примере входная запись представляет собой простой POJO `RowData`.

```
public static class RowData {
        public int id;
        public String name;
        public int score;

        public RowData() {}

        public RowData(int id, String name, int score) {
            this.id = id;
            this.name = name;
            this.score = score;
        }
  }
```

- Основная программа выглядит следующим образом:

```
// Генерация записей, которые используют RowData в качестве
контейнера.
RowData[] records = new RowData[]{
        new RowData(1, "selena-rowdata", 100),
        new RowData(2, "flink-rowdata", 100),
    };
DataStream<RowData> source = env.fromElements(records);

// Настройте connector с необходимыми свойствами.
SelenaSinkOptions options = SelenaSinkOptions.builder()
        .withProperty("jdbc-url", jdbcUrl)
        .withProperty("load-url", loadUrl)
        .withProperty("database-name", "test")
        .withProperty("table-name", "score_board")
        .withProperty("username", "root")
        .withProperty("password", "")
        .build();

/**
 * Flink connector будет использовать массив объектов Java
(Object[]) для представления строки, которая будет загружена в
таблицу Selena,
 * и каждый элемент является значением для столбца.
 * Вам необходимо определить схему Object[], которая соответствует
схеме таблицы Selena.
 */
TableSchema schema = TableSchema.builder()
        .field("id", DataTypes.INT().notNull())
        .field("name", DataTypes.STRING())
        .field("score", DataTypes.INT())
        // Когда таблица Selena является таблицей Primary Key, вы
должны указать notNull(), например, DataTypes.INT().notNull(), для
первичного ключа `id`.
        .primaryKey("id")
        .build();
// Преобразуйте RowData в Object[] в соответствии со схемой.
RowDataTransformer transformer = new RowDataTransformer();
// Создайте sink со схемой, параметрами и трансформером.
SinkFunction<RowData> selenaink = SelenaSink.sink(schema, options,
transformer);
source.addSink(selenaink);
```

- `RowDataTransformer` в основной программе определен следующим образом:

```
private static class RowDataTransformer implements
SelenaSinkRowBuilder<RowData> {

    /**
     * Установите каждый элемент массива объектов в соответствии с
входными RowData.
     * Схема массива соответствует схеме таблицы Selena.
     */
    @Override
    public void accept(Object[] internalRow, RowData rowData) {
        internalRow[0] = rowData.id;
        internalRow[1] = rowData.name;
        internalRow[2] = rowData.score;
        // Когда таблица Selena является таблицей Primary Key, вам
нужно установить последний элемент, чтобы указать, является ли
загрузка данных операцией UPSERT или DELETE.
        internalRow[internalRow.length - 1] =
SelenaSinkOP.UPSERT.ordinal();
    }
}
```

Синхронизация данных с Flink CDC 3.0 (с поддержкой изменения схемы)

Фреймворк Flink CDC 3.0 можно использовать для легкого создания потокового конвейера ELT из источников CDC (таких как MySQL и Kafka) в Selena. Конвейер может синхронизировать всю базу данных, объединенные таблицы шардинга и изменения схемы из источников в Selena.

Начиная с v1.2.9, Flink connector для Selena интегрирован в этот фреймворк как Selena Pipeline Connector. Selena Pipeline Connector поддерживает:

- Автоматическое создание баз данных и таблиц
- Синхронизацию изменений схемы
- Полную и инкрементальную синхронизацию данных

Для быстрого старта см. Streaming ELT from MySQL to Selena using Flink CDC 3.0 with Selena Pipeline Connector.

Рекомендуется использовать Selena v1.5.2 и более поздние версии для включения fast_schema_evolution. Это улучшит скорость добавления или удаления столбцов и снизит

использование ресурсов.

3.11.7 Лучшие практики

Загрузка данных в таблицу Primary Key

В этом разделе показано, как загружать данные в таблицу Primary Key Selena для достижения частичных обновлений и условных обновлений. Вы можете увидеть Изменение данных через загрузку для введения этих функций. Эти примеры используют Flink SQL.

Подготовка

Создайте базу данных `test` и создайте таблицу Primary Key `score_board` в Selena.

```
CREATE DATABASE `test`;

CREATE TABLE `test`.`score_board`
(
    `id` int(11) NOT NULL COMMENT "",
    `name` varchar(65533) NULL DEFAULT "" COMMENT "",
    `score` int(11) NOT NULL DEFAULT "0" COMMENT ""
)
ENGINE=OLAP
PRIMARY KEY(`id`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`id`);
```

Частичное обновление

Этот пример покажет, как загружать данные только в столбцы `id` и `name`.

1. Вставьте две строки данных в таблицу Selena `score_board` в клиенте MySQL.

```
mysql> INSERT INTO `score_board` VALUES (1, 'selena', 100), (2,
'flink', 100);

mysql> select * from score_board;
+------+-----------+-------+
| id   | name      | score |
+------+-----------+-------+
|    1 | selena |   100 |
|    2 | flink     |   100 |
+------+-----------+-------+
2 rows in set (0.02 sec)
```

2. Создайте таблицу Flink `score_board` в клиенте Flink SQL.

   - Определите DDL, который включает только столбцы `id` и `name`.
   - Установите опцию `sink.properties.partial_update` в `true`, что сообщает Flink connector о необходимости выполнения частичных обновлений.
   - Если версия Flink connector `<=` 1.2.7, вам также необходимо установить опцию `sink.properties.columns` в `id,name,__op`, чтобы сообщить Flink connector, какие столбцы нужно обновить. Обратите внимание, что вам нужно добавить поле `__op` в конце. Поле `__op` указывает, что загрузка данных является операцией UPSERT или DELETE, и его значения устанавливаются connector автоматически.

```
CREATE TABLE `score_board` (
    `id` INT,
    `name` STRING,
    PRIMARY KEY (id) NOT ENFORCED
) WITH (
    'connector' = 'selena',
    'jdbc-url' = 'jdbc:mysql://127.0.0.1:9030',
    'load-url' = '127.0.0.1:8030',
    'database-name' = 'test',
    'table-name' = 'score_board',
    'username' = 'root',
    'password' = '',
    'sink.properties.partial_update' = 'true',
    -- только для версии Flink connector <= 1.2.7
    'sink.properties.columns' = 'id,name,__op'
);
```

3. Вставьте две строки данных в таблицу Flink. Первичные ключи строк данных такие же, как у строк в таблице Selena, но значения в столбце `name` изменены.

```
INSERT INTO `score_board` VALUES (1, 'selena-update'), (2, 'flink-
update');
```

4. Запросите таблицу Selena в клиенте MySQL.

```
mysql> select * from score_board;
+------+------------------+-------+
| id   | name             | score |
+------+------------------+-------+
|    1 | selena-update |   100 |
|    2 | flink-update     |   100 |
+------+------------------+-------+
2 rows in set (0.02 sec)
```

Вы можете видеть, что изменились только значения для `name`, а значения для `score` не изменились.

Условное обновление

Этот пример покажет, как выполнить условное обновление в соответствии со значением столбца `score`. Обновление для `id` вступает в силу только тогда, когда новое значение для `score` больше или равно старому значению.

1. Вставьте две строки данных в таблицу Selena в клиенте MySQL.

```
mysql> INSERT INTO `score_board` VALUES (1, 'selena', 100), (2,
'flink', 100);

mysql> select * from score_board;
+------+-----------+-------+
| id   | name      | score |
+------+-----------+-------+
|    1 | selena |   100 |
|    2 | flink     |   100 |
+------+-----------+-------+
2 rows in set (0.02 sec)
```

2. Создайте таблицу Flink `score_board` следующими способами:

   - Определите DDL, включающий все столбцы.
   - Установите опцию `sink.properties.merge_condition` в `score`, чтобы сообщить connector использовать столбец `score` в качестве условия.
   - Установите опцию `sink.version` в `V1`, что сообщает connector использовать Stream Load.

```
CREATE TABLE `score_board` (
    `id` INT,
    `name` STRING,
    `score` INT,
    PRIMARY KEY (id) NOT ENFORCED
) WITH (
    'connector' = 'selena',
    'jdbc-url' = 'jdbc:mysql://127.0.0.1:9030',
    'load-url' = '127.0.0.1:8030',
    'database-name' = 'test',
    'table-name' = 'score_board',
    'username' = 'root',
    'password' = '',
    'sink.properties.merge_condition' = 'score',
    'sink.version' = 'V1'
    );
```

3. Вставьте две строки данных в таблицу Flink. Первичные ключи строк данных такие же, как у строк в таблице Selena. Первая строка данных имеет меньшее значение в столбце `score`, а вторая строка данных имеет большее значение в столбце `score`.

```
INSERT INTO `score_board` VALUES (1, 'selena-update', 99), (2, 'flink-
update', 101);
```

4. Запросите таблицу Selena в клиенте MySQL.

```
mysql> select * from score_board;
+------+--------------+-------+
| id   | name         | score |
+------+--------------+-------+
|    1 | selena     |   100 |
|    2 | flink-update |   101 |
+------+--------------+-------+
2 rows in set (0.03 sec)
```

Вы можете видеть, что изменились только значения второй строки данных, а значения первой строки данных не изменились.

Загрузка данных в столбцы типа BITMAP

`BITMAP` часто используется для ускорения count distinct, например, для подсчета UV, см. Использование Bitmap для точного Count Distinct. Здесь мы возьмем подсчет UV в качестве примера, чтобы показать, как загружать данные в столбцы типа `BITMAP`.

1. Создайте таблицу Aggregate Selena в клиенте MySQL.

   В базе данных `test` создайте таблицу Aggregate `page_uv`, где столбец `visit_users` определен как тип `BITMAP` и настроен с агрегатной функцией `BITMAP_UNION`.

```
CREATE TABLE `test`.`page_uv` (
  `page_id` INT NOT NULL COMMENT 'page ID',
  `visit_date` datetime NOT NULL COMMENT 'access time',
  `visit_users` BITMAP BITMAP_UNION NOT NULL COMMENT 'user ID'
) ENGINE=OLAP
AGGREGATE KEY(`page_id`, `visit_date`)
DISTRIBUTED BY HASH(`page_id`);
```

2. Создайте таблицу Flink в клиенте Flink SQL.

   Столбец `visit_user_id` в таблице Flink имеет тип `BIGINT`, и мы хотим загрузить этот столбец в столбец `visit_users` типа `BITMAP` в таблице Selena. Поэтому при определении DDL таблицы Flink обратите внимание, что:

   - Поскольку Flink не поддерживает `BITMAP`, вам необходимо определить столбец `visit_user_id` как тип `BIGINT` для представления столбца `visit_users` типа

`BITMAP` в таблице Selena.

- Вам необходимо установить опцию `sink.properties.columns` в `page_id,visit_date,user_id,visit_users=to_bitmap(visit_user_id)`, что сообщает connector сопоставление столбцов между таблицей Flink и таблицей Selena. Также вам необходимо использовать функцию `to_bitmap` чтобы сообщить connector преобразовать данные типа `BIGINT` в тип `BITMAP`.

```
CREATE TABLE `page_uv` (
    `page_id` INT,
    `visit_date` TIMESTAMP,
    `visit_user_id` BIGINT
) WITH (
    'connector' = 'selena',
    'jdbc-url' = 'jdbc:mysql://127.0.0.1:9030',
    'load-url' = '127.0.0.1:8030',
    'database-name' = 'test',
    'table-name' = 'page_uv',
    'username' = 'root',
    'password' = '',
    'sink.properties.columns' =
'page_id,visit_date,visit_user_id,visit_users=to_bitmap(visit_user_id)'
);
```

3. Загрузите данные в таблицу Flink в клиенте Flink SQL.

```
INSERT INTO `page_uv` VALUES
   (1, CAST('2020-06-23 01:30:30' AS TIMESTAMP), 13),
   (1, CAST('2020-06-23 01:30:30' AS TIMESTAMP), 23),
   (1, CAST('2020-06-23 01:30:30' AS TIMESTAMP), 33),
   (1, CAST('2020-06-23 02:30:30' AS TIMESTAMP), 13),
   (2, CAST('2020-06-23 01:30:30' AS TIMESTAMP), 23);
```

4. Вычислите UV страниц из таблицы Selena в клиенте MySQL.

```
MySQL [test]> SELECT `page_id`, COUNT(DISTINCT `visit_users`) FROM
`page_uv` GROUP BY `page_id`;
+---------+-----------------------------+
| page_id | count(DISTINCT visit_users) |
+---------+-----------------------------+
|       2 |                           1 |
|       1 |                           3 |
+---------+-----------------------------+
2 rows in set (0.05 sec)
```

Загрузка данных в столбцы типа HLL

`HLL` может использоваться для приблизительного count distinct, см. Использование HLL для приблизительного count distinct.

Здесь мы возьмем подсчет UV в качестве примера, чтобы показать, как загружать данные в столбцы типа `HLL`.

1. Создайте таблицу Aggregate Selena

   В базе данных `test` создайте таблицу Aggregate `hll_uv`, где столбец `visit_users` определен как тип `HLL` и настроен с агрегатной функцией `HLL_UNION`.

```
CREATE TABLE `hll_uv` (
  `page_id` INT NOT NULL COMMENT 'page ID',
  `visit_date` datetime NOT NULL COMMENT 'access time',
  `visit_users` HLL HLL_UNION NOT NULL COMMENT 'user ID'
) ENGINE=OLAP
AGGREGATE KEY(`page_id`, `visit_date`)
DISTRIBUTED BY HASH(`page_id`);
```

2. Создайте таблицу Flink в клиенте Flink SQL.

   Столбец `visit_user_id` в таблице Flink имеет тип `BIGINT`, и мы хотим загрузить этот столбец в столбец `visit_users` типа `HLL` в таблице Selena. Поэтому при определении DDL таблицы Flink обратите внимание, что:

   - Поскольку Flink не поддерживает `BITMAP`, вам необходимо определить столбец `visit_user_id` как тип `BIGINT` для представления столбца `visit_users` типа `HLL` в таблице Selena.

- Вам необходимо установить опцию `sink.properties.columns` в `page_id,visit_date,user_id,visit_users=hll_hash(visit_user_id)`, что сообщает connector сопоставление столбцов между таблицей Flink и таблицей Selena. Также вам необходимо использовать функцию `hll_hash` чтобы сообщить connector преобразовать данные типа `BIGINT` в тип `HLL`.

```
CREATE TABLE `hll_uv` (
    `page_id` INT,
    `visit_date` TIMESTAMP,
    `visit_user_id` BIGINT
) WITH (
    'connector' = 'selena',
    'jdbc-url' = 'jdbc:mysql://127.0.0.1:9030',
    'load-url' = '127.0.0.1:8030',
    'database-name' = 'test',
    'table-name' = 'hll_uv',
    'username' = 'root',
    'password' = '',
    'sink.properties.columns' =
'page_id,visit_date,visit_user_id,visit_users=hll_hash(visit_user_id)'
);
```

3. Загрузите данные в таблицу Flink в клиенте Flink SQL.

```
INSERT INTO `hll_uv` VALUES
   (3, CAST('2023-07-24 12:00:00' AS TIMESTAMP), 78),
   (4, CAST('2023-07-24 13:20:10' AS TIMESTAMP), 2),
   (3, CAST('2023-07-24 12:30:00' AS TIMESTAMP), 674);
```

4. Вычислите UV страниц из таблицы Selena в клиенте MySQL.

```
mysql> SELECT `page_id`, COUNT(DISTINCT `visit_users`) FROM `hll_uv`
GROUP BY `page_id`;
**+---------+-----------------------------+
| page_id | count(DISTINCT visit_users) |
+---------+-----------------------------+
|       3 |                           2 |
|       4 |                           1 |
+---------+-----------------------------+
2 rows in set (0.04 sec)
```

3.12 Непрерывная загрузка данных из Apache Pulsar

> **EXPERIMENTAL FEATURE**
>
> Рекомендации по использованию экспериментальных функций

Начиная с версии Selena 2.5, Routine Load поддерживает непрерывную загрузку данных из Apache Pulsar. Pulsar — это распределённая платформа обмена сообщениями и потоковой передачи с открытым исходным кодом, основанная на архитектуре разделения хранения и вычислений. Загрузка данных из Pulsar через Routine Load аналогична загрузке данных из Apache Kafka. В этой теме используются данные в формате CSV в качестве примера для демонстрации загрузки данных из Apache Pulsar через Routine Load.

3.12.1 Поддерживаемые форматы файлов данных

Routine Load поддерживает потребление данных в форматах CSV и JSON из cluster Pulsar.

> ПРИМЕЧАНИЕ
>
> Для данных в формате CSV Selena поддерживает строки в кодировке UTF-8 длиной до 50 байт в качестве разделителей столбцов. Часто используемые разделители столбцов включают запятую (,), табуляцию и вертикальную черту (|).

3.12.2 Концепции, связанные с Pulsar

Топик

Топики в Pulsar — это именованные каналы для передачи сообщений от производителей к потребителям. Топики в Pulsar делятся на партиционированные топики и непартиционированные топики.

- **Партиционированные топики** — это особый тип топика, который обрабатывается несколькими брокерами, что обеспечивает более высокую пропускную способность. Партиционированный топик фактически реализован как N внутренних топиков, где N — количество партиций.
- **Непартиционированные топики** — это обычный тип топика, который обслуживается только одним брокером, что ограничивает максимальную пропускную способность

топика.

Message ID

Message ID сообщения назначается экземплярами BookKeeper сразу после постоянного сохранения сообщения. Message ID указывает конкретную позицию сообщения в реестре и уникален в пределах cluster Pulsar.

Pulsar поддерживает указание потребителями начальной позиции через consumer.*seek*(*messageId*). Но в отличие от смещения потребителя Kafka, которое является длинным целочисленным значением, Message ID состоит из четырёх частей: `ledgerId:entryID:partition-index:batch-index`.

Поэтому вы не можете получить Message ID напрямую из сообщения. В результате в настоящее время **Routine Load не поддерживает указание начальной позиции при загрузке данных из Pulsar, а поддерживает только потребление данных с начала или конца партиции.**

Подписка

Подписка — это именованное правило конфигурации, которое определяет, как сообщения доставляются потребителям. Pulsar также поддерживает одновременную подписку потребителей на несколько топиков. Топик может иметь несколько подписок.

Тип подписки определяется при подключении потребителя к ней, и тип можно изменить, перезапустив всех потребителей с другой конфигурацией. В Pulsar доступны четыре типа подписок:

- `exclusive` (по умолчанию): Только один потребитель может быть подключён к подписке. Только одному клиенту разрешено потреблять сообщения.
- `shared`: Несколько потребителей могут быть подключены к одной подписке. Сообщения доставляются по кругу между потребителями, и любое данное сообщение доставляется только одному потребителю.
- `failover`: Несколько потребителей могут быть подключены к одной подписке. Главный потребитель выбирается для непартиционированного топика или каждой партиции партиционированного топика и получает сообщения. Когда главный потребитель отключается, все (неподтверждённые и последующие) сообщения доставляются следующему потребителю в очереди.

- `key_shared`: Несколько потребителей могут быть подключены к одной подписке. Сообщения распределяются между потребителями, и сообщения с одинаковым ключом или одинаковым ключом упорядочивания доставляются только одному потребителю.

> Примечание:
>
> В настоящее время Routine Load использует тип exclusive.

3.12.3 Создание задания Routine Load

Следующие примеры описывают, как потреблять сообщения в формате CSV в Pulsar и загружать данные в Selena, создавая задание Routine Load. Для получения подробных инструкций и справки см. CREATE ROUTINE LOAD.

```
CREATE ROUTINE LOAD load_test.routine_wiki_edit_1 ON routine_wiki_edit
COLUMNS TERMINATED BY ",",
ROWS TERMINATED BY "\n",
COLUMNS (order_id, pay_dt, customer_name, nationality, temp_gender, price)
WHERE event_time > "2022-01-01 00:00:00",
PROPERTIES
(
    "desired_concurrent_number" = "1",
    "max_batch_interval" = "15000",
    "max_error_number" = "1000"
)
FROM PULSAR
(
    "pulsar_service_url" = "pulsar://localhost:6650",
    "pulsar_topic" = "persistent://tenant/namespace/topic-name",
    "pulsar_subscription" = "load-test",
    "pulsar_partitions" = "load-partition-0,load-partition-1",
    "pulsar_initial_positions" = "POSITION_EARLIEST,POSITION_LATEST",
    "property.auth.token" =
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUJzdWIiOiJqaXV0aWFuY2hlbiJ9.lulGngOC72vE70OW54zc
);
```

При создании Routine Load для потребления данных из Pulsar большинство входных параметров, кроме `data_source_properties`, такие же, как при потреблении данных из Kafka. Для описания параметров, кроме `data_source_properties`, см. CREATE ROUTINE LOAD.

Параметры, связанные с `data_source_properties`, и их описания приведены ниже:

Параметр	Обязательный	Описание
pulsar_service_url	Да	URL, используемый для подключения к cluster Pulsar. Формат: `"pulsar://ip:port"` или `"pulsar://service:port"`. Пример: `"pulsar_service_url" = "pulsar://localhost:6650"`
pulsar_topic	Да	Топик для подписки. Пример: "pulsar_topic" = "persistent://tenant/namespace/topic-name"
pulsar_subscription	Да	Подписка, настроенная для топика. Пример: `"pulsar_subscription" = "my_subscription"`
pulsar_partitions, pulsar_initial_positions	Нет	`pulsar_partitions`: Партиции для подписки в топике. `pulsar_initial_positions`: начальные позиции партиций, указанных в `pulsar_partitions`. Начальные позиции должны соответствовать партициям в `pulsar_partitions`. Допустимые значения: `POSITION_EARLIEST` (значение по умолчанию): Подписка начинается с самого раннего доступного сообщения в партиции. `POSITION_LATEST`: Подписка начинается с самого последнего доступного сообщения в партиции. Примечание: Если `pulsar_partitions` не указан, подписка оформляется на все партиции топика. Если указаны и `pulsar_partitions`, и `property.pulsar_default_initial_position`, значение `pulsar_partitions` переопределяет значение `property.pulsar_default_initial_position`. Если ни `pulsar_partitions`, ни `property.pulsar_default_initial_position` не указаны, подписка начинается с самого последнего доступного сообщения в партиции. Пример: `"pulsar_partitions" = "my-partition-0,my-partition-1,my-partition-2,my-partition-3", "pulsar_initial_positions" = "POSITION_EARLIEST,POSITION_EARLIEST,POSITION_LATEST,POSITION_LATEST"`

Routine Load поддерживает следующие пользовательские параметры для Pulsar.

Параметр	Обязательный	Описание
property.pulsar_default_initial_position	Нет	Начальные позиции по умолчанию при подписке на партиции топика. Параметр действует, когда `pulsar_initial_positions` не указан. Его допустимые значения такие же, как у `pulsar_initial_positions`. Пример: `"property.pulsar_default_initial_position" = "POSITION_EARLIEST"`

Параметр	Обязательный	Описание
property.auth.token	Нет	Если Pulsar включает аутентификацию клиентов с использованием токенов безопасности, вам нужна строка токена для подтверждения вашей личности. Пример: `"property.auth.token" = "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUJzdWIiOiJqaXV0aWFuY2hlbiJ9.lulGngOC72vE70OW54zcbyw7XdKSOxET94WT_hIqD"`

3.12.4 Проверка задания и задачи загрузки

Проверка задания загрузки

Выполните оператор SHOW ROUTINE LOAD, чтобы проверить статус задания загрузки `routine_wiki_edit_1`. Selena возвращает состояние выполнения `State`, статистическую информацию (включая общее количество потреблённых строк и общее количество загруженных строк) `Statistics` и прогресс задания загрузки `progress`.

При проверке задания Routine Load, которое потребляет данные из Pulsar, большинство возвращаемых параметров, кроме `progress`, такие же, как при потреблении данных из Kafka. `progress` относится к отставанию, то есть количеству неподтверждённых сообщений в партиции.

```
MySQL [load_test] > SHOW ROUTINE LOAD for routine_wiki_edit_1 \G
*************************** 1. row ***************************
                  Id: 10142
                Name: routine_wiki_edit_1
          CreateTime: 2022-06-29 14:52:55
           PauseTime: 2022-06-29 17:33:53
             EndTime: NULL
              DbName: default_cluster:test_pulsar
           TableName: test1
               State: PAUSED
      DataSourceType: PULSAR
      CurrentTaskNum: 0
       JobProperties:
{"partitions":"*","rowDelimiter":"'\n'","partial_update":"false","columnToCol
DataSourceProperties: {"serviceUrl":"pulsar://localhost:6650","currentPulsarP
    CustomProperties: {"auth.token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUJzdWIiOiJqa
           Statistic: {"receivedBytes":5480943882,"errorRows":0,"committedTas
            Progress: {"my-partition-0(backlog): 100","my-partition-1(backlog
ReasonOfStateChanged:
        ErrorLogUrls:
            OtherMsg:
1 row in set (0.00 sec)
```

Проверка задачи загрузки

Выполните оператор SHOW ROUTINE LOAD TASK, чтобы проверить задачи загрузки для задания `routine_wiki_edit_1`, такие как количество выполняемых задач, партиции топика Kafka, которые потребляются, и прогресс потребления `DataSourceProperties`, а также соответствующий узел Coordinator BE `BeId`.

```
MySQL [example_db]> SHOW ROUTINE LOAD TASK WHERE JobName =
"routine_wiki_edit_1" \G
```

3.12.5 Изменение задания загрузки

Перед изменением задания загрузки необходимо приостановить его с помощью оператора PAUSE ROUTINE LOAD. Затем можно выполнить ALTER ROUTINE LOAD. После изменения можно выполнить оператор RESUME ROUTINE LOAD, чтобы возобновить его, и проверить его статус с помощью оператора SHOW ROUTINE LOAD.

При использовании Routine Load для потребления данных из Pulsar большинство возвращаемых параметров, кроме `data_source_properties`, такие же, как при потреблении данных из Kafka.

Обратите внимание на следующие моменты:

- Среди параметров, связанных с `data_source_properties`, в настоящее время поддерживается изменение только `pulsar_partitions`, `pulsar_initial_positions` и пользовательских параметров Pulsar `property.pulsar_default_initial_position` и `property.auth.token`. Параметры `pulsar_service_url`, `pulsar_topic` и `pulsar_subscription` изменить нельзя.
- Если вам нужно изменить партицию для потребления и соответствующую начальную позицию, вам нужно убедиться, что вы указали партицию с помощью `pulsar_partitions` при создании задания Routine Load, и можно изменить только начальную позицию `pulsar_initial_positions` указанной партиции.
- Если вы указываете только топик `pulsar_topic` при создании задания Routine Load, но не партиции `pulsar_partitions`, вы можете изменить начальную позицию всех партиций топика через `pulsar_default_initial_position`.

3.13 Изменение данных через загрузку

Таблицы с Primary Key, предоставляемые Selena, позволяют вносить изменения в данные таблиц Selena, выполняя задачи Stream Load, Broker Load или Routine Load. Эти изменения данных включают вставки, обновления и удаления. Однако таблицы с Primary Key не поддерживают изменение данных с использованием Spark Load или INSERT.

Selena также поддерживает частичные обновления и условные обновления.

Вы можете загружать данные в таблицы Selena только как пользователь, имеющий привилегию INSERT на эти таблицы Selena. Если у вас нет привилегии INSERT, следуйте инструкциям в разделе GRANT, чтобы предоставить привилегию INSERT пользователю, которого вы используете для подключения к вашему cluster Selena. Синтаксис: `GRANT INSERT ON TABLE <table_name> IN DATABASE <database_name> TO { ROLE <role_name> | USER <user_identity>}`.

В этом разделе используются данные CSV в качестве примера для описания того, как вносить изменения в данные таблицы Selena через загрузку. Поддерживаемые форматы файлов данных зависят от выбранного вами метода загрузки.

> **ПРИМЕЧАНИЕ**
>
> Для данных CSV вы можете использовать строку UTF-8, такую как запятая (,), табуляция или вертикальная черта (|), длиной не более 50 байт в качестве текстового разделителя.

3.13.1 Реализация

Таблицы с Primary Key, предоставляемые Selena, поддерживают операции UPSERT и DELETE и не различают операции INSERT от операций UPDATE.

При создании задачи загрузки Selena поддерживает добавление поля с именем `__op` в команду или оператор создания задачи. Поле `__op` используется для указания типа операции, которую вы хотите выполнить.

> **ПРИМЕЧАНИЕ**
>
> При создании таблицы вам не нужно добавлять столбец с именем `__op` в эту таблицу.

Метод определения поля `__op` зависит от выбранного вами метода загрузки:

- Если вы выбираете Stream Load, определите поле `__op` с помощью параметра `columns`.
- Если вы выбираете Broker Load, определите поле `__op` с помощью предложения SET.
- Если вы выбираете Routine Load, определите поле `__op` с помощью столбца `COLUMNS`.

Вы можете решить, добавлять ли поле `__op`, в зависимости от изменений данных, которые вы хотите внести. Если вы не добавляете поле `__op`, тип операции по умолчанию — UPSERT. Основные сценарии изменения данных следующие:

- Если файл данных, который вы хотите загрузить, включает только операции UPSERT, вам не нужно добавлять поле `__op`.
- Если файл данных, который вы хотите загрузить, включает только операции DELETE, вы должны добавить поле `__op` и указать тип операции как DELETE.

- Если файл данных, который вы хотите загрузить, включает как операции UPSERT, так и DELETE, вы должны добавить поле `__op` и убедиться, что файл данных содержит столбец со значениями `0` или `1`. Значение `0` указывает на операцию UPSERT, а значение `1` указывает на операцию DELETE.

3.13.2 Примечания по использованию

- Убедитесь, что каждая строка в вашем файле данных имеет одинаковое количество столбцов.

- Столбцы, в которые вносятся изменения данных, должны включать столбец первичного ключа.

3.13.3 Базовые операции

В этом разделе приведены примеры того, как вносить изменения в данные таблицы Selena через загрузку. Для получения подробного синтаксиса и описания параметров см. STREAM LOAD, BROKER LOAD и CREATE ROUTINE LOAD.

UPSERT

Если файл данных, который вы хотите загрузить, включает только операции UPSERT, вам не нужно добавлять поле `__op`.

> **ПРИМЕЧАНИЕ**
>
> Если вы добавляете поле `__op`:
>
> - Вы можете указать тип операции как UPSERT.
>
> - Вы можете оставить поле `__op` пустым, поскольку тип операции по умолчанию — UPSERT.

Примеры данных

1. Подготовьте файл данных.

   a. Создайте CSV-файл с именем `example1.csv` в вашей локальной файловой системе. Файл состоит из трёх столбцов, которые представляют идентификатор пользователя, имя пользователя и оценку пользователя соответственно.

```
101,Lily,100
102,Rose,100
```

b. Опубликуйте данные `example1.csv` в `topic1` вашего cluster Kafka.

2. Подготовьте таблицу Selena.

a. Создайте таблицу с Primary Key с именем `table1` в вашей базе данных Selena `test_db`. Таблица состоит из трёх столбцов: `id`, `name` и `score`, из которых `id` является первичным ключом.

```
CREATE TABLE `table1`
(
    `id` int(11) NOT NULL COMMENT "user ID",
    `name` varchar(65533) NOT NULL COMMENT "user name",
    `score` int(11) NOT NULL COMMENT "user score"
)
ENGINE=OLAP
PRIMARY KEY(`id`)
DISTRIBUTED BY HASH(`id`);
```

> **ПРИМЕЧАНИЕ**
>
> Начиная с версии v1.5.2, Selena может автоматически устанавливать количество bucket (BUCKETS) при создании таблицы или добавлении partition. Вам больше не нужно вручную устанавливать количество bucket. Для получения подробной информации см. установка количества bucket.

b. Вставьте запись в `table1`.

```
INSERT INTO table1 VALUES
    (101, 'Lily',80);
```

Загрузка данных

Запустите задачу загрузки, чтобы обновить запись с `id` равным `101` из `example1.csv` в `table1` и вставить запись с `id` равным `102` из `example1.csv` в `table1`.

- Запустите задачу Stream Load.

- Если вы не хотите включать поле `__op`, выполните следующую команду:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "label:label1" \
    -H "column_separator:," \
    -T example1.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table1/_stream_load
```

- Если вы хотите включить поле `__op`, выполните следующую команду:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "label:label2" \
    -H "column_separator:," \
    -H "columns:__op ='upsert'" \
    -T example1.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table1/_stream_load
```

- Запустите задачу Broker Load.

  - Если вы не хотите включать поле `__op`, выполните следующую команду:

```
LOAD LABEL test_db.label1
(
    data infile("hdfs://<hdfs_host>:<hdfs_port>/example1.csv")
    into table table1
    columns terminated by ","
    format as "csv"
)
WITH BROKER;
```

  - Если вы хотите включить поле `__op`, выполните следующую команду:

```
LOAD LABEL test_db.label2
(
    data infile("hdfs://<hdfs_host>:<hdfs_port>/example1.csv")
    into table table1
    columns terminated by ","
    format as "csv"
    set (__op = 'upsert')
)
WITH BROKER;
```

- Запустите задачу Routine Load.
  - Если вы не хотите включать поле `__op`, выполните следующую команду:

```
CREATE ROUTINE LOAD test_db.table1 ON table1
COLUMNS TERMINATED BY ",",
COLUMNS (id, name, score)
PROPERTIES
(
    "desired_concurrent_number" = "3",
    "max_batch_interval" = "20",
    "max_batch_rows"= "250000",
    "max_error_number" = "1000"
)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker_host>:<kafka_broker_port>",
    "kafka_topic" = "test1",
    "property.kafka_default_offsets" ="OFFSET_BEGINNING"
);
```

  - Если вы хотите включить поле `__op`, выполните следующую команду:

```
CREATE ROUTINE LOAD test_db.table1 ON table1
COLUMNS TERMINATED BY ",",
COLUMNS (id, name, score, __op ='upsert')
PROPERTIES
(
    "desired_concurrent_number" = "3",
    "max_batch_interval" = "20",
    "max_batch_rows"= "250000",
    "max_error_number" = "1000"
)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker_host>:<kafka_broker_port>",
    "kafka_topic" = "test1",
    "property.kafka_default_offsets" ="OFFSET_BEGINNING"
);
```

Запрос данных

После завершения загрузки запросите данные `table1`, чтобы убедиться, что загрузка прошла успешно:

```
SELECT * FROM table1;
+------+------+-------+
| id   | name | score |
+------+------+-------+
|  101 | Lily |   100 |
|  102 | Rose |   100 |
+------+------+-------+
2 rows in set (0.02 sec)
```

Как показано в результате запроса выше, запись с `id` равным `101` из `example1.csv` была обновлена в `table1`, а запись с `id` равным `102` из `example1.csv` была вставлена в `table1`.

DELETE

Если файл данных, который вы хотите загрузить, включает только операции DELETE, вы должны добавить поле `__op` и указать тип операции как DELETE.

Примеры данных

1. Подготовьте файл данных.

   a. Создайте CSV-файл с именем `example2.csv` в вашей локальной файловой системе. Файл состоит из трёх столбцов, которые представляют идентификатор пользователя, имя пользователя и оценку пользователя соответственно.

   ```
   101,Jack,100
   ```

   b. Опубликуйте данные `example2.csv` в `topic2` вашего cluster Kafka.

2. Подготовьте таблицу Selena.

   a. Создайте таблицу с Primary Key с именем `table2` в вашей базе данных Selena `test_db`. Таблица состоит из трёх столбцов: `id`, `name` и `score`, из которых `id` является первичным ключом.

   ```
   CREATE TABLE `table2`
         (
       `id` int(11) NOT NULL COMMENT "user ID",
       `name` varchar(65533) NOT NULL COMMENT "user name",
       `score` int(11) NOT NULL COMMENT "user score"
   )
   ENGINE=OLAP
   PRIMARY KEY(`id`)
   DISTRIBUTED BY HASH(`id`);
   ```

   > **ПРИМЕЧАНИЕ**
   >
   > Начиная с версии v1.5.2, Selena может автоматически устанавливать количество bucket (BUCKETS) при создании таблицы или добавлении partition. Вам больше не нужно вручную устанавливать количество bucket. Для получения подробной информации см. установка количества bucket.

   b. Вставьте две записи в `table2`.

```
INSERT INTO table2 VALUES
(101, 'Jack', 100),
(102, 'Bob', 90);
```

Загрузка данных

Запустите задачу загрузки, чтобы удалить запись с `id` равным `101` из `example2.csv` из `table2`.

- Запустите задачу Stream Load.

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "label:label3" \
    -H "column_separator:," \
    -H "columns:__op='delete'" \
    -T example2.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table2/_stream_load
```

- Запустите задачу Broker Load.

```
LOAD LABEL test_db.label3
(
    data infile("hdfs://<hdfs_host>:<hdfs_port>/example2.csv")
    into table table2
    columns terminated by ","
    format as "csv"
    set (__op = 'delete')
)
WITH BROKER;
```

- Запустите задачу Routine Load.

```
CREATE ROUTINE LOAD test_db.table2 ON table2
COLUMNS(id, name, score, __op = 'delete')
PROPERTIES
(
    "desired_concurrent_number" = "3",
    "max_batch_interval" = "20",
    "max_batch_rows"= "250000",
    "max_error_number" = "1000"
)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker_host>:<kafka_broker_port>",
    "kafka_topic" = "test2",
    "property.kafka_default_offsets" ="OFFSET_BEGINNING"
);
```

Запрос данных

После завершения загрузки запросите данные `table2`, чтобы убедиться, что загрузка прошла успешно:

```
SELECT * FROM table2;
+------+------+-------+
| id   | name | score |
+------+------+-------+
|  102 | Bob  |    90 |
+------+------+-------+
1 row in set (0.00 sec)
```

Как показано в результате запроса выше, запись с `id` равным `101` из `example2.csv` была удалена из `table2`.

UPSERT и DELETE

Если файл данных, который вы хотите загрузить, включает как операции UPSERT, так и DELETE, вы должны добавить поле `__op` и убедиться, что файл данных содержит столбец со значениями `0` или `1`. Значение `0` указывает на операцию UPSERT, а значение `1` указывает на операцию DELETE.

Примеры данных

1. Подготовьте файл данных.

   a. Создайте CSV-файл с именем `example3.csv` в вашей локальной файловой системе. Файл состоит из четырёх столбцов, которые представляют идентификатор пользователя, имя пользователя, оценку пользователя и тип операции соответственно.

```
101,Tom,100,1
102,Sam,70,0
103,Stan,80,0
```

   b. Опубликуйте данные `example3.csv` в `topic3` вашего cluster Kafka.

2. Подготовьте таблицу Selena.

   a. Создайте таблицу с Primary Key с именем `table3` в вашей базе данных Selena `test_db`. Таблица состоит из трёх столбцов: `id`, `name` и `score`, из которых `id` является первичным ключом.

```
CREATE TABLE `table3`
(
    `id` int(11) NOT NULL COMMENT "user ID",
    `name` varchar(65533) NOT NULL COMMENT "user name",
    `score` int(11) NOT NULL COMMENT "user score"
)
ENGINE=OLAP
PRIMARY KEY(`id`)
DISTRIBUTED BY HASH(`id`);
```

> **ПРИМЕЧАНИЕ**
>
> Начиная с версии v1.5.2, Selena может автоматически устанавливать количество bucket (BUCKETS) при создании таблицы или добавлении partition. Вам больше не нужно вручную устанавливать количество bucket. Для получения подробной информации см. установка количества bucket.

   b. Вставьте две записи в `table3`.

```
INSERT INTO table3 VALUES
    (101, 'Tom', 100),
    (102, 'Sam', 90);
```

Загрузка данных

Запустите задачу загрузки, чтобы удалить запись с `id` равным `101` из `example3.csv` из `table3`, обновить запись с `id` равным `102` из `example3.csv` в `table3` и вставить запись с `id` равным `103` из `example3.csv` в `table3`.

- Запустите задачу Stream Load:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "label:label4" \
    -H "column_separator:," \
    -H "columns: id, name, score, temp, __op = temp" \
    -T example3.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table3/_stream_load
```

> **ПРИМЕЧАНИЕ**
>
> В приведённом выше примере четвёртый столбец, представляющий тип операции в `example3.csv`, временно называется `temp`, и поле `__op` сопоставляется со столбцом `temp` с помощью параметра `columns`. Таким образом, Selena может решить, выполнять ли операцию UPSERT или DELETE, в зависимости от того, равно ли значение в четвёртом столбце `example3.csv` `0` или `1`.

- Запустите задачу Broker Load:

```
LOAD LABEL test_db.label4
(
    data infile("hdfs://<hdfs_host>:<hdfs_port>/example1.csv")
    into table table1
    columns terminated by ","
    format as "csv"
    (id, name, score, temp)
    set (__op=temp)
)
WITH BROKER;
```

- Запустите задачу Routine Load:

```
CREATE ROUTINE LOAD test_db.table3 ON table3
COLUMNS(id, name, score, temp, __op = temp)
PROPERTIES
(
    "desired_concurrent_number" = "3",
    "max_batch_interval" = "20",
    "max_batch_rows"= "250000",
    "max_error_number" = "1000"
)
FROM KAFKA
(
    "kafka_broker_list" = "<kafka_broker_host>:<kafka_broker_port>",
    "kafka_topic" = "test3",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

Запрос данных

После завершения загрузки запросите данные `table3`, чтобы убедиться, что загрузка прошла успешно:

```
SELECT * FROM table3;
+------+------+-------+
| id   | name | score |
+------+------+-------+
|  102 | Sam  |    70 |
|  103 | Stan |    80 |
+------+------+-------+
2 rows in set (0.01 sec)
```

Как показано в результате запроса выше, запись с `id` равным `101` из `example3.csv` была удалена из `table3`, запись с `id` равным `102` из `example3.csv` была обновлена в `table3`, а запись с `id` равным `103` из `example3.csv` была вставлена в `table3`.

3.13.4 Частичные обновления

Таблицы с Primary Key также поддерживают частичные обновления и предоставляют два режима частичных обновлений — строчный режим и колоночный режим — для различных сценариев обновления данных. Эти два режима частичных обновлений могут минимизировать накладные расходы на частичные обновления, насколько это возможно, гарантируя при этом производительность запросов и обеспечивая обновления в реальном времени. Строчный режим более подходит для сценариев обновления в реальном времени, включающих множество столбцов и небольшие пакеты. Колоночный режим подходит для сценариев пакетной обработки обновлений, включающих несколько столбцов и большое количество строк.

> **ВНИМАНИЕ**
>
> При выполнении частичного обновления, если обновляемая строка не существует, Selena вставляет новую строку и заполняет значениями по умолчанию поля, которые пусты, поскольку в них не были вставлены обновления данных.

В этом разделе используется CSV в качестве примера для описания того, как выполнять частичные обновления.

Примеры данных

1. Подготовьте файл данных.

   a. Создайте CSV-файл с именем `example4.csv` в вашей локальной файловой системе. Файл состоит из двух столбцов, которые представляют идентификатор

пользователя и имя пользователя соответственно.

```
101,Lily
102,Rose
103,Alice
```

b. Опубликуйте данные `example4.csv` в `topic4` вашего cluster Kafka.

2. Подготовьте таблицу Selena.

a. Создайте таблицу с Primary Key с именем `table4` в вашей базе данных Selena `test_db`. Таблица состоит из трёх столбцов: `id`, `name` и `score`, из которых `id` является первичным ключом.

```
CREATE TABLE `table4`
(
    `id` int(11) NOT NULL COMMENT "user ID",
    `name` varchar(65533) NOT NULL COMMENT "user name",
    `score` int(11) NOT NULL COMMENT "user score"
)
ENGINE=OLAP
PRIMARY KEY(`id`)
DISTRIBUTED BY HASH(`id`);
```

> **ПРИМЕЧАНИЕ**
>
> Начиная с версии v1.5.2, Selena может автоматически устанавливать количество bucket (BUCKETS) при создании таблицы или добавлении partition. Вам больше не нужно вручную устанавливать количество bucket. Для получения подробной информации см. установка количества bucket.

b. Вставьте запись в `table4`.

```
INSERT INTO table4 VALUES
    (101, 'Tom',80);
```

Загрузка данных

Запустите загрузку для обновления данных в двух столбцах `example4.csv` в столбцы `id` и `name` таблицы `table4`.

- Запустите задачу Stream Load:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "label:label7" -H "column_separator:," \
    -H "partial_update:true" \
    -H "columns:id,name" \
    -T example4.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table4/_stream_load
```

> **ПРИМЕЧАНИЕ**
>
> Если вы выбираете Stream Load, вы должны установить параметр `partial_update` в значение `true`, чтобы включить функцию частичного обновления. По умолчанию используются частичные обновления в строчном режиме. Если вам нужно выполнить частичные обновления в колоночном режиме, вам нужно установить `partial_update_mode` в значение `column`. Кроме того, вы должны использовать параметр `columns`, чтобы указать столбцы, которые вы хотите обновить.

- Запустите задачу Broker Load:

```
LOAD LABEL test_db.table4
(
    data infile("hdfs://<hdfs_host>:<hdfs_port>/example4.csv")
    into table table4
    format as "csv"
    (id, name)
)
WITH BROKER
PROPERTIES
(
    "partial_update" = "true"
);
```

> **ПРИМЕЧАНИЕ**
>
> Если вы выбираете Broker Load, вы должны установить параметр `partial_update` в значение `true`, чтобы включить функцию частичного обновления. По умолчанию используются частичные обновления в строчном режиме. Если вам нужно выполнить частичные обновления в колоночном режиме, вам нужно установить `partial_update_mode` в значение `column`. Кроме того, вы должны использовать параметр `column_list`, чтобы указать столбцы, которые вы хотите обновить.

- Запустите задачу Routine Load:

```
CREATE ROUTINE LOAD test_db.table4 on table4
COLUMNS (id, name),
COLUMNS TERMINATED BY ','
PROPERTIES
(
    "partial_update" = "true"
)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker_host>:<kafka_broker_port>",
    "kafka_topic" = "test4",
    "property.kafka_default_offsets" ="OFFSET_BEGINNING"
);
```

> **ПРИМЕЧАНИЕ**
>
> - Если вы выбираете Routine Load, вы должны установить параметр `partial_update` в значение `true`, чтобы включить функцию частичного обновления. Кроме того, вы должны использовать параметр `COLUMNS`, чтобы указать столбцы, которые вы хотите обновить.
> - Routine Load поддерживает только частичные обновления в строчном режиме и не поддерживает частичные обновления в колоночном режиме.

Запрос данных

После завершения загрузки запросите данные `table4`, чтобы убедиться, что загрузка прошла успешно:

```
SELECT * FROM table4;
+------+-------+-------+
| id   | name  | score |
+------+-------+-------+
|  102 | Rose  |     0 |
|  101 | Lily  |    80 |
|  103 | Alice |     0 |
+------+-------+-------+
3 rows in set (0.01 sec)
```

Как показано в результате запроса выше, запись с `id` равным `101` из `example4.csv` была обновлена в `table4`, а записи с `id` равными `102` и `103` из `example4.csv` были вставлены в `table4`.

3.13.5 Условные обновления

Начиная с Selena v1.5.2, таблицы с Primary Key поддерживают условные обновления. Вы можете указать столбец, не являющийся первичным ключом, в качестве условия для определения того, могут ли обновления вступить в силу. Таким образом, обновление от исходной записи к целевой записи вступает в силу только тогда, когда запись исходных данных имеет значение большее или равное значению записи целевых данных в указанном столбце.

Функция условного обновления предназначена для решения проблемы неупорядоченности данных. Если исходные данные неупорядочены, вы можете использовать эту функцию, чтобы гарантировать, что новые данные не будут перезаписаны старыми данными.

> **ВНИМАНИЕ**
>
> - Вы не можете указать разные столбцы в качестве условий обновления для одной и той же партии данных.
> - Операции DELETE не поддерживают условные обновления.
> - В версиях ранее v1.5.2 частичные обновления и условные обновления не могут использоваться одновременно. Начиная с версии v1.5.2, Selena поддерживает использование частичных обновлений вместе с условными обновлениями.

Примеры данных

1. Подготовьте файл данных.

a. Создайте CSV-файл с именем `example5.csv` в вашей локальной файловой системе. Файл состоит из трёх столбцов, которые представляют идентификатор пользователя, версию и оценку пользователя соответственно.

```
101,1,100
102,3,100
```

b. Опубликуйте данные `example5.csv` в `topic5` вашего cluster Kafka.

2. Подготовьте таблицу Selena.

a. Создайте таблицу с Primary Key с именем `table5` в вашей базе данных Selena `test_db`. Таблица состоит из трёх столбцов: `id`, `version` и `score`, из которых `id` является первичным ключом.

```
CREATE TABLE `table5`
(
    `id` int(11) NOT NULL COMMENT "user ID",
    `version` int NOT NULL COMMENT "version",
    `score` int(11) NOT NULL COMMENT "user score"
)
ENGINE=OLAP
PRIMARY KEY(`id`) DISTRIBUTED BY HASH(`id`);
```

> **ПРИМЕЧАНИЕ**
>
> Начиная с версии v1.5.2, Selena может автоматически устанавливать количество bucket (BUCKETS) при создании таблицы или добавлении partition. Вам больше не нужно вручную устанавливать количество bucket. Для получения подробной информации см. установка количества bucket.

b. Вставьте запись в `table5`.

```
INSERT INTO table5 VALUES
    (101, 2, 80),
    (102, 2, 90);
```

Загрузка данных

Запустите загрузку для обновления записей с значениями `id` равными `101` и `102` соответственно из `example5.csv` в `table5` и укажите, что обновления вступают в силу только тогда, когда значение `version` в каждой из двух записей больше или равно их текущим значениям `version`.

- Запустите задачу Stream Load:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "label:label10" \
    -H "column_separator:," \
    -H "merge_condition:version" \
    -T example5.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table5/_stream_load
```

- Запустите задачу Insert Load:

```
INSERT INTO test_db.table5 properties("merge_condition" = "version")
VALUES (101, 2, 70), (102, 3, 100);
```

- Запустите задачу Routine Load:

```
CREATE ROUTINE LOAD test_db.table5 on table5
COLUMNS (id, version, score),
COLUMNS TERMINATED BY ','
PROPERTIES
(
    "merge_condition" = "version"
)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker_host>:<kafka_broker_port>",
    "kafka_topic" = "topic5",
    "property.kafka_default_offsets" ="OFFSET_BEGINNING"
);
```

- Запустите задачу Broker Load:

```
LOAD LABEL test_db.table5
( DATA INFILE ("s3://xxx.csv")
  INTO TABLE table5 COLUMNS TERMINATED BY "," FORMAT AS "CSV"
)
WITH BROKER
PROPERTIES
(
    "merge_condition" = "version"
);
```

Запрос данных

После завершения загрузки запросите данные `table5`, чтобы убедиться, что загрузка прошла успешно:

```
SELECT * FROM table5;
+------+------+-------+
| id   | version | score |
+------+------+-------+
|  101 |       2 |   80 |
|  102 |       3 |  100 |
+------+------+-------+
2 rows in set (0.02 sec)
```

Как показано в результате запроса выше, запись с `id` равным `101` из `example5.csv` не была обновлена в `table5`, а запись с `id` равным `102` из `example5.csv` была вставлена в `table5`.

3.14 Трансформация данных при загрузке

Selena поддерживает трансформацию данных при загрузке.

Эта функция поддерживает Stream Load, Broker Load и Routine Load, но не поддерживает Spark Load.

Вы можете загружать данные в таблицы Selena только как пользователь, имеющий привилегию INSERT на эти таблицы Selena. Если у вас нет привилегии INSERT, следуйте инструкциям в разделе GRANT, чтобы предоставить привилегию INSERT пользователю, которого вы используете для подключения к вашему cluster Selena. Синтаксис: `GRANT`

`INSERT ON TABLE <table_name> IN DATABASE <database_name> TO { ROLE <role_name> | USER <user_identity>}`.

В этом разделе используются данные CSV в качестве примера для описания того, как извлекать и трансформировать данные при загрузке. Поддерживаемые форматы файлов данных различаются в зависимости от выбранного метода загрузки.

> **ПРИМЕЧАНИЕ**
>
> Для данных CSV вы можете использовать строку UTF-8, такую как запятая (,), табуляция или вертикальная черта (|), длина которой не превышает 50 байт, в качестве разделителя текста.

3.14.1 Сценарии

При загрузке файла данных в таблицу Selena данные файла данных могут не полностью сопоставляться с данными таблицы Selena. В этой ситуации вам не нужно извлекать или трансформировать данные перед загрузкой их в таблицу Selena. Selena может помочь вам извлечь и трансформировать данные во время загрузки:

- Пропуск столбцов, которые не нужно загружать.

  Вы можете пропустить столбцы, которые не нужно загружать. Кроме того, если столбцы файла данных расположены в другом порядке, чем столбцы таблицы Selena, вы можете создать сопоставление столбцов между файлом данных и таблицей Selena.

- Фильтрация строк, которые вы не хотите загружать.

  Вы можете указать условия фильтрации, на основе которых Selena отфильтровывает строки, которые вы не хотите загружать.

- Генерация новых столбцов из исходных столбцов.

  Сгенерированные столбцы — это специальные столбцы, которые вычисляются из исходных столбцов файла данных. Вы можете сопоставить сгенерированные столбцы со столбцами таблицы Selena.

- Извлечение значений полей partition из пути к файлу.

Если файл данных сгенерирован из Apache Hive™, вы можете извлечь значения полей partition из пути к файлу.

3.14.2 Примеры данных

1. Создайте файлы данных в вашей локальной файловой системе.

   a. Создайте файл данных с именем `file1.csv`. Файл состоит из четырех столбцов, которые представляют ID пользователя, пол пользователя, дату события и тип события по порядку.

   ```
   354,female,2020-05-20,1
   465,male,2020-05-21,2
   576,female,2020-05-22,1
   687,male,2020-05-23,2
   ```

   b. Создайте файл данных с именем `file2.csv`. Файл состоит только из одного столбца, который представляет дату.

   ```
   2020-05-20
   2020-05-21
   2020-05-22
   2020-05-23
   ```

2. Создайте таблицы в вашей базе данных Selena `test_db`.

   > **ПРИМЕЧАНИЕ**
   >
   > Начиная с версии v1.5.2, Selena может автоматически устанавливать количество buckets (BUCKETS) при создании таблицы или добавлении partition. Вам больше не нужно вручную устанавливать количество buckets. Для получения подробной информации см. установка количества buckets.

   a. Создайте таблицу с именем `table1`, которая состоит из трех столбцов: `event_date`, `event_type` и `user_id`.

```
MySQL [test_db]> CREATE TABLE table1
(
    `event_date` DATE COMMENT "event date",
    `event_type` TINYINT COMMENT "event type",
    `user_id` BIGINT COMMENT "user ID"
)
DISTRIBUTED BY HASH(user_id);
```

b. Создайте таблицу с именем `table2`, которая состоит из четырех столбцов: `date`, `year`, `month` и `day`.

```
MySQL [test_db]> CREATE TABLE table2
(
    `date` DATE COMMENT "date",
    `year` INT COMMENT "year",
    `month` TINYINT COMMENT "month",
    `day` TINYINT COMMENT "day"
)
DISTRIBUTED BY HASH(date);
```

3. Загрузите `file1.csv` и `file2.csv` в путь `/user/selena/data/input/` вашего кластера HDFS, опубликуйте данные `file1.csv` в `topic1` вашего кластера Kafka и опубликуйте данные `file2.csv` в `topic2` вашего кластера Kafka.

3.14.3 Пропуск столбцов, которые не нужно загружать

Файл данных, который вы хотите загрузить в таблицу Selena, может содержать некоторые столбцы, которые не могут быть сопоставлены ни с одним столбцом таблицы Selena. В этой ситуации Selena поддерживает загрузку только тех столбцов, которые могут быть сопоставлены из файла данных со столбцами таблицы Selena.

Эта функция поддерживает загрузку данных из следующих источников данных:

- Локальная файловая система
- HDFS и облачное хранилище

> **ПРИМЕЧАНИЕ**
>
> В этом разделе используется HDFS в качестве примера.

- Kafka

В большинстве случаев столбцы CSV-файла не имеют имен. Для некоторых CSV-файлов первая строка состоит из имен столбцов, но Selena обрабатывает содержимое первой строки как обычные данные, а не как имена столбцов. Поэтому при загрузке CSV-файла вы должны временно назвать столбцы CSV-файла **по порядку** в выражении или команде создания задачи. Эти временно названные столбцы сопоставляются **по имени** со столбцами таблицы Selena. Обратите внимание на следующие моменты о столбцах файла данных:

- Данные столбцов, которые могут быть сопоставлены и временно названы с использованием имен столбцов в таблице Selena, загружаются напрямую.
- Столбцы, которые не могут быть сопоставлены со столбцами таблицы Selena, игнорируются, данные этих столбцов не загружаются.
- Если некоторые столбцы могут быть сопоставлены со столбцами таблицы Selena, но не названы временно в выражении или команде создания задачи, задача загрузки сообщает об ошибках.

В этом разделе используются `file1.csv` и `table1` в качестве примера. Четыре столбца `file1.csv` временно названы как `user_id`, `user_gender`, `event_date` и `event_type` по порядку. Среди временно названных столбцов `file1.csv`, `user_id`, `event_date` и `event_type` могут быть сопоставлены с конкретными столбцами `table1`, тогда как `user_gender` не может быть сопоставлен ни с одним столбцом `table1`. Поэтому `user_id`, `event_date` и `event_type` загружаются в `table1`, но `user_gender` не загружается.

Загрузка данных

Загрузка данных из локальной файловой системы

Если `file1.csv` хранится в вашей локальной файловой системе, выполните следующую команду для создания задачи Stream Load:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "column_separator:," \
    -H "columns: user_id, user_gender, event_date, event_type" \
    -T file1.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table1/_stream_load
```

> **ПРИМЕЧАНИЕ**
>
> Если вы выбираете Stream Load, вы должны использовать параметр `columns` для временного именования столбцов файла данных для создания сопоставления столбцов между файлом данных и таблицей Selena.

Для получения подробного описания синтаксиса и параметров см. STREAM LOAD.

Загрузка данных из кластера HDFS

Если `file1.csv` хранится в вашем кластере HDFS, выполните следующее выражение для создания задачи Broker Load:

```
LOAD LABEL test_db.label1
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/file1.csv")
    INTO TABLE `table1`
    FORMAT AS "csv"
    COLUMNS TERMINATED BY ","
    (user_id, user_gender, event_date, event_type)
)
WITH BROKER;
```

> **ПРИМЕЧАНИЕ**
>
> Если вы выбираете Broker Load, вы должны использовать параметр `column_list` для временного именования столбцов файла данных для создания сопоставления столбцов между файлом данных и таблицей Selena.

Для получения подробного описания синтаксиса и параметров см. BROKER LOAD.

Загрузка данных из кластера Kafka

Если данные `file1.csv` опубликованы в `topic1` вашего кластера Kafka, выполните следующее выражение для создания задачи Routine Load:

```
CREATE ROUTINE LOAD test_db.table101 ON table1
    COLUMNS TERMINATED BY ",",
    COLUMNS(user_id, user_gender, event_date, event_type)
FROM KAFKA
(
    "kafka_broker_list" = "<kafka_broker_host>:<kafka_broker_port>",
    "kafka_topic" = "topic1",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

> **ПРИМЕЧАНИЕ**
>
> Если вы выбираете Routine Load, вы должны использовать параметр `COLUMNS` для временного именования столбцов файла данных для создания сопоставления столбцов между файлом данных и таблицей Selena.

Для получения подробного описания синтаксиса и параметров см. CREATE ROUTINE LOAD.

Запрос данных

После завершения загрузки данных из вашей локальной файловой системы, кластера HDFS или кластера Kafka запросите данные `table1`, чтобы убедиться, что загрузка прошла успешно:

```
MySQL [test_db]> SELECT * FROM table1;
+------------+------------+---------+
| event_date | event_type | user_id |
+------------+------------+---------+
| 2020-05-22 |          1 |     576 |
| 2020-05-20 |          1 |     354 |
| 2020-05-21 |          2 |     465 |
| 2020-05-23 |          2 |     687 |
+------------+------------+---------+
4 rows in set (0.01 sec)
```

3.14.4 Фильтрация строк, которые вы не хотите загружать

При загрузке файла данных в таблицу Selena вы можете не захотеть загружать определенные строки файла данных. В этой ситуации вы можете использовать предложение WHERE для указания строк, которые вы хотите загрузить. Selena отфильтровывает все строки, которые не соответствуют условиям фильтрации, указанным в предложении WHERE.

Эта функция поддерживает загрузку данных из следующих источников данных:

- Локальная файловая система
- HDFS и облачное хранилище

> **ПРИМЕЧАНИЕ**
>
> В этом разделе используется HDFS в качестве примера.

- Kafka

В этом разделе используются `file1.csv` и `table1` в качестве примера. Если вы хотите загрузить только строки, тип события которых `1`, из `file1.csv` в `table1`, вы можете использовать предложение WHERE для указания условия фильтрации `event_type = 1`.

Загрузка данных

Загрузка данных из локальной файловой системы

Если `file1.csv` хранится в вашей локальной файловой системе, выполните следующую команду для создания задачи Stream Load:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "column_separator:," \
    -H "columns: user_id, user_gender, event_date, event_type" \
    -H "where: event_type=1" \
    -T file1.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table1/_stream_load
```

Для получения подробного описания синтаксиса и параметров см. STREAM LOAD.

Загрузка данных из кластера HDFS

Если `file1.csv` хранится в вашем кластере HDFS, выполните следующее выражение для создания задачи Broker Load:

```
LOAD LABEL test_db.label2
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/file1.csv")
    INTO TABLE `table1`
    FORMAT AS "csv"
    COLUMNS TERMINATED BY ","
    (user_id, user_gender, event_date, event_type)
    WHERE event_type = 1
)
WITH BROKER;
```

Для получения подробного описания синтаксиса и параметров см. BROKER LOAD.

Загрузка данных из кластера Kafka

Если данные `file1.csv` опубликованы в `topic1` вашего кластера Kafka, выполните следующее выражение для создания задачи Routine Load:

```
CREATE ROUTINE LOAD test_db.table102 ON table1
COLUMNS TERMINATED BY ",",
COLUMNS (user_id, user_gender, event_date, event_type),
WHERE event_type = 1
FROM KAFKA
(
    "kafka_broker_list" = "<kafka_broker_host>:<kafka_broker_port>",
    "kafka_topic" = "topic1",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

Для получения подробного описания синтаксиса и параметров см. CREATE ROUTINE LOAD.

Запрос данных

После завершения загрузки данных из вашей локальной файловой системы, кластера HDFS или кластера Kafka запросите данные `table1`, чтобы убедиться, что загрузка прошла успешно:

```
MySQL [test_db]> SELECT * FROM table1;
+------------+------------+---------+
| event_date | event_type | user_id |
+------------+------------+---------+
| 2020-05-20 |          1 |     354 |
| 2020-05-22 |          1 |     576 |
+------------+------------+---------+
2 rows in set (0.01 sec)
```

3.14.5 Генерация новых столбцов из исходных столбцов

При загрузке файла данных в таблицу Selena некоторые данные из файла данных могут потребовать преобразований перед тем, как данные могут быть загружены в таблицу Selena. В этой ситуации вы можете использовать функции или выражения в команде или выражении создания задачи для реализации преобразований данных.

Эта функция поддерживает загрузку данных из следующих источников данных:

- Локальная файловая система
- HDFS и облачное хранилище

  > **ПРИМЕЧАНИЕ**
  >
  > В этом разделе используется HDFS в качестве примера.

- Kafka

В этом разделе используются `file2.csv` и `table2` в качестве примера. `file2.csv` состоит только из одного столбца, который представляет дату. Вы можете использовать функции year, month и day для извлечения года, месяца и дня в каждой дате из `file2.csv` и загрузить извлеченные данные в столбцы `year`, `month` и `day` таблицы `table2`.

Загрузка данных

Загрузка данных из локальной файловой системы

Если `file2.csv` хранится в вашей локальной файловой системе, выполните следующую команду для создания задачи Stream Load:

```
curl --location-trusted -u <username>:<password> \
    -H "Expect:100-continue" \
    -H "column_separator:," \
    -H "columns:date,year=year(date),month=month(date),day=day(date)" \
    -T file2.csv -XPUT \
    http://<fe_host>:<fe_http_port>/api/test_db/table2/_stream_load
```

> **ПРИМЕЧАНИЕ**
>
> - В параметре `columns` вы должны сначала временно назвать **все столбцы** файла данных, а затем временно назвать новые столбцы, которые вы хотите сгенерировать из исходных столбцов файла данных. Как показано в предыдущем примере, единственный столбец `file2.csv` временно назван как `date`, а затем вызываются функции `year=year(date)`, `month=month(date)` и `day=day(date)` для генерации трех новых столбцов, которые временно названы как `year`, `month` и `day`.
> - Stream Load не поддерживает `column_name = function(column_name)`, но поддерживает `column_name = function(column_name)`.

Для получения подробного описания синтаксиса и параметров см. STREAM LOAD.

Загрузка данных из кластера HDFS

Если `file2.csv` хранится в вашем кластере HDFS, выполните следующее выражение для создания задачи Broker Load:

```
LOAD LABEL test_db.label3
(
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/file2.csv")
    INTO TABLE `table2`
    FORMAT AS "csv"
    COLUMNS TERMINATED BY ","
    (date)
    SET(year=year(date), month=month(date), day=day(date))
)
WITH BROKER;
```

> **ПРИМЕЧАНИЕ**
>
> Вы должны сначала использовать параметр `column_list` для временного именования **всех столбцов** файла данных, а затем использовать предложение SET для временного именования новых столбцов, которые вы хотите сгенерировать из исходных столбцов файла данных. Как показано в предыдущем примере, единственный столбец `file2.csv` временно назван как `date` в параметре `column_list`, а затем в предложении SET вызываются функции `year=year(date)`, `month=month(date)` и `day=day(date)` для генерации трех новых столбцов, которые временно названы как `year`, `month` и `day`.

Для получения подробного описания синтаксиса и параметров см. BROKER LOAD.

Загрузка данных из кластера Kafka

Если данные `file2.csv` опубликованы в `topic2` вашего кластера Kafka, выполните следующее выражение для создания задачи Routine Load:

```
CREATE ROUTINE LOAD test_db.table201 ON table2
    COLUMNS TERMINATED BY ",",
    COLUMNS(date,year=year(date),month=month(date),day=day(date))
FROM KAFKA
(
    "kafka_broker_list" = "<kafka_broker_host>:<kafka_broker_port>",
    "kafka_topic" = "topic2",
    "property.kafka_default_offsets" = "OFFSET_BEGINNING"
);
```

> **ПРИМЕЧАНИЕ**
>
> В параметре `COLUMNS` вы должны сначала временно назвать **все столбцы** файла данных, а затем временно назвать новые столбцы, которые вы хотите сгенерировать из исходных столбцов файла данных. Как показано в предыдущем примере, единственный столбец `file2.csv` временно назван как `date`, а затем вызываются функции `year=year(date)`, `month=month(date)` и `day=day(date)` для генерации трех новых столбцов, которые временно названы как `year`, `month` и `day`.

Для получения подробного описания синтаксиса и параметров см. CREATE ROUTINE LOAD.

Запрос данных

После завершения загрузки данных из вашей локальной файловой системы, кластера HDFS или кластера Kafka запросите данные `table2`, чтобы убедиться, что загрузка прошла успешно:

```
MySQL [test_db]> SELECT * FROM table2;
+------------+------+-------+------+
| date       | year | month | day  |
+------------+------+-------+------+
| 2020-05-20 | 2020 |  5    | 20   |
| 2020-05-21 | 2020 |  5    | 21   |
| 2020-05-22 | 2020 |  5    | 22   |
| 2020-05-23 | 2020 |  5    | 23   |
+------------+------+-------+------+
4 rows in set (0.01 sec)
```

3.14.6 Извлечение значений полей partition из пути к файлу

Если путь к файлу, который вы указываете, содержит поля partition, вы можете использовать параметр `COLUMNS FROM PATH AS` для указания полей partition, которые вы хотите извлечь из путей к файлам. Поля partition в путях к файлам эквивалентны столбцам в файлах данных. Параметр `COLUMNS FROM PATH AS` поддерживается только при загрузке данных из кластера HDFS.

Например, вы хотите загрузить следующие четыре файла данных, сгенерированных из Hive:

```
/user/selena/data/input/date=2020-05-20/data
1,354
/user/selena/data/input/date=2020-05-21/data
2,465
/user/selena/data/input/date=2020-05-22/data
1,576
/user/selena/data/input/date=2020-05-23/data
2,687
```

Четыре файла данных хранятся в пути `/user/selena/data/input/` вашего кластера HDFS. Каждый из этих файлов данных разбит на разделы по полю partition `date` и состоит из двух столбцов, которые представляют тип события и ID пользователя по порядку.

Загрузка данных из кластера HDFS

Выполните следующее выражение для создания задачи Broker Load, которая позволяет вам извлечь значения поля partition `date` из пути к файлу `/user/selena/data/input/` и использовать подстановочный знак (*) для указания того, что вы хотите загрузить все файлы данных в пути к файлу в `table1`:

```
LOAD LABEL test_db.label4
(
    DATA INFILE("hdfs://<fe_host>:
<fe_http_port>/user/selena/data/input/date=*/*")
    INTO TABLE `table1`
    FORMAT AS "csv"
    COLUMNS TERMINATED BY ","
    (event_type, user_id)
    COLUMNS FROM PATH AS (date)
    SET(event_date = date)
)
WITH BROKER;
```

> **ПРИМЕЧАНИЕ**
>
> В предыдущем примере поле partition `date` в указанном пути к файлу эквивалентно столбцу `event_date` таблицы `table1`. Поэтому вам необходимо использовать предложение SET для сопоставления поля partition `date` со столбцом `event_date`. Если поле partition в указанном пути к файлу имеет то же имя, что и столбец таблицы Selena, вам не нужно использовать предложение SET для создания сопоставления.

Для получения подробного описания синтаксиса и параметров см. BROKER LOAD.

Запрос данных

После завершения загрузки данных из вашего кластера HDFS запросите данные `table1`, чтобы убедиться, что загрузка прошла успешно:

```
MySQL [test_db]> SELECT * FROM table1;
+------------+------------+---------+
| event_date | event_type | user_id |
+------------+------------+---------+
| 2020-05-22 |          1 |     576 |
| 2020-05-20 |          1 |     354 |
| 2020-05-21 |          2 |     465 |
| 2020-05-23 |          2 |     687 |
+------------+------------+---------+
4 rows in set (0.01 sec)
```

3.15 Загрузка данных с помощью инструментов

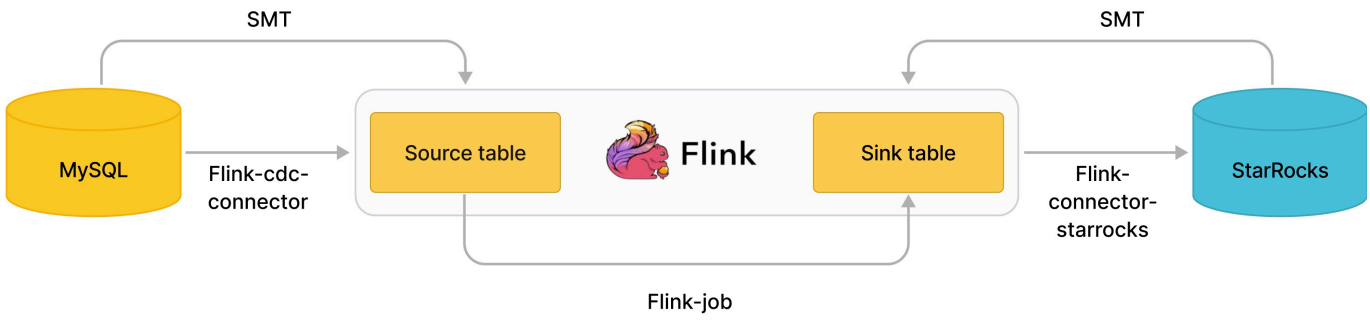
Selena и её партнёры по экосистеме предлагают следующие инструменты, которые помогут вам беспрепятственно интегрировать Selena с внешними базами данных.

3.15.1 SMT

SMT (Selena Migration Tool) — это инструмент миграции данных, предоставляемый Selena, разработанный для оптимизации сложных конвейеров загрузки данных: исходные базы данных (такие как MySQL, Oracle, PostgreSQL) ---> Flink ---> целевые cluster Selena. Его основные функции следующие:

- Упрощение создания таблиц в Selena: генерирует операторы для создания таблиц в Selena на основе информации из внешних баз данных и целевого cluster Selena.
- Упрощение процесса полной или инкрементальной синхронизации данных в конвейере данных: генерирует SQL-операторы, которые можно выполнить в SQL-клиенте Flink для отправки заданий Flink для синхронизации данных.

Следующая блок-схема иллюстрирует процесс загрузки данных из исходной базы данных MySQL через Flink в Selena.



3.15.2 DataX

DataX — это инструмент для офлайн-синхронизации данных, открытый Alibaba. DataX может синхронизировать данные между различными гетерогенными источниками данных, включая реляционные базы данных (MySQL, Oracle и т.д.), HDFS и Hive. DataX предоставляет плагин Selena Writer для синхронизации данных из источников данных, поддерживаемых DataX, в Selena.

3.15.3 CloudCanal

CloudCanal Community Edition — это бесплатная платформа для миграции и синхронизации данных, опубликованная ClouGence Co., Ltd, которая интегрирует миграцию схемы, полную миграцию данных, проверку, исправление и инкрементальную синхронизацию в реальном времени. Вы можете напрямую добавить Selena в качестве источника данных в визуальном интерфейсе CloudCanal и создать задачи для автоматической миграции или синхронизации данных из исходных баз данных (например, MySQL, Oracle, PostgreSQL) в Selena.

3.15.4 Kettle connector

Kettle — это инструмент ETL (Extract, Transform, Load) с визуальным графическим интерфейсом, который позволяет пользователям создавать рабочие процессы обработки данных путём перетаскивания компонентов и настройки параметров. Этот интуитивный метод значительно упрощает процесс обработки и загрузки данных, позволяя пользователям более удобно обрабатывать данные. Кроме того, Kettle предоставляет богатую библиотеку компонентов, позволяя пользователям выбирать подходящие компоненты в соответствии со своими потребностями и выполнять различные сложные задачи обработки данных.

Selena предлагает Kettle Connector для интеграции с Kettle. Объединяя мощные возможности обработки и трансформации данных Kettle с высокопроизводительными

возможностями хранения и аналитики данных Selena, можно достичь более гибких и эффективных рабочих процессов обработки данных.

3.16 Строгий режим

Строгий режим — это необязательное свойство, которое можно настроить для загрузки данных. Он влияет на поведение загрузки и окончательные загруженные данные.

В этой теме рассказывается, что такое строгий режим и как его настроить.

3.16.1 Понимание строгого режима

Во время загрузки данных типы данных исходных столбцов могут не полностью соответствовать типам данных целевых столбцов. В таких случаях Selena выполняет преобразования значений исходных столбцов, имеющих несовместимые типы данных. Преобразования данных могут завершиться неудачей по различным причинам, таким как несоответствие типов данных полей и переполнение длины поля. Значения исходных столбцов, которые не удалось правильно преобразовать, являются неквалифицированными значениями столбцов, а исходные строки, содержащие неквалифицированные значения столбцов, называются "неквалифицированными строками". Строгий режим используется для управления фильтрацией неквалифицированных строк во время загрузки данных.

Строгий режим работает следующим образом:

- Если строгий режим включён, Selena загружает только квалифицированные строки. Неквалифицированные строки отфильтровываются, и возвращаются сведения о неквалифицированных строках.
- Если строгий режим отключён, Selena преобразует неквалифицированные значения столбцов в `NULL` и загружает неквалифицированные строки, содержащие эти значения `NULL`, вместе с квалифицированными строками.

Обратите внимание на следующее:

- В реальных бизнес-сценариях как квалифицированные, так и неквалифицированные строки могут содержать значения `NULL`. Если целевые столбцы не допускают значения `NULL`, Selena сообщает об ошибках и отфильтровывает строки, содержащие значения `NULL`.

- Максимальный процент неквалифицированных строк, которые могут быть отфильтрованы для задания загрузки, контролируется необязательным свойством задания `max_filter_ratio`.

> **ПРИМЕЧАНИЕ**
>
> Свойство `max_filter_ratio` для INSERT поддерживается начиная с версии v1.5.2.

Например, вы хотите загрузить четыре строки со значениями `\N` (`\N` обозначает значение `NULL`), `abc`, `2000` и `1` соответственно в столбце из CSV-файла данных в таблицу Selena, и тип данных целевого столбца таблицы Selena — TINYINT [-128, 127].

- Значение исходного столбца `\N` обрабатывается в `NULL` при преобразовании в TINYINT.

  > **ПРИМЕЧАНИЕ**
  >
  > `\N` всегда обрабатывается в `NULL` при преобразовании независимо от целевого типа данных.

- Значение исходного столбца `abc` обрабатывается в `NULL`, потому что его тип данных не является TINYINT, и преобразование завершается неудачей.

- Значение исходного столбца `2000` обрабатывается в `NULL`, потому что оно выходит за пределы диапазона, поддерживаемого TINYINT, и преобразование завершается неудачей.

- Значение исходного столбца `1` может быть правильно преобразовано в значение типа TINYINT `1`.

Если строгий режим отключён, Selena загружает все четыре строки.

Если строгий режим включён, Selena загружает только строки со значениями `\N` или `1` и отфильтровывает строки со значениями `abc` или `2000`. Отфильтрованные строки учитываются в максимальном проценте строк, которые могут быть отфильтрованы из-за недостаточного качества данных, как указано параметром `max_filter_ratio`.

Окончательные загруженные данные при отключённом строгом режиме

Значение исходного столбца	Значение столбца при преобразовании в TINYINT	Результат загрузки, когда целевой столбец допускает NULL	Результат загрузки, когда целевой столбец не допускает NULL
\N	NULL	Значение `NULL` загружено.	Сообщается об ошибке.
abc	NULL	Значение `NULL` загружено.	Сообщается об ошибке.
2000	NULL	Значение `NULL` загружено.	Сообщается об ошибке.
1	1	Значение `1` загружено.	Значение `1` загружено.

Окончательные загруженные данные при включённом строгом режиме

Значение исходного столбца	Значение столбца при преобразовании в TINYINT	Результат загрузки, когда целевой столбец допускает NULL	Результат загрузки, когда целевой столбец не допускает NULL
\N	NULL	Значение `NULL` загружено.	Сообщается об ошибке.
abc	NULL	Значение `NULL` не допускается и поэтому отфильтровывается.	Сообщается об ошибке.
2000	NULL	Значение `NULL` не допускается и поэтому отфильтровывается.	Сообщается об ошибке.
1	1	Значение `1` загружено.	Значение `1` загружено.

3.16.2 Настройка строгого режима

Вы можете использовать параметр `strict_mode` для установки строгого режима для задания загрузки. Допустимые значения: `true` и `false`. Значение по умолчанию — `false`. Значение `true` включает строгий режим, а значение `false` отключает строгий режим. Обратите внимание, что параметр `strict_mode` поддерживается для INSERT начиная с версии v1.5.2, со значением по умолчанию `true`. Теперь, за исключением Stream Load, для всех других методов загрузки `strict_mode` устанавливается одинаково в разделе PROPERTIES.

Вы также можете использовать сессионную переменную `enable_insert_strict` для установки строгого режима. Допустимые значения: `true` и `false`. Значение по умолчанию — `true`. Значение `true` включает строгий режим, а значение `false` отключает строгий режим.

> **ПРИМЕЧАНИЕ**
>
> Начиная с версии v1.5.2, когда `enable_insert_strict` установлено в `true`, система загружает только квалифицированные строки. Неквалифицированные строки отфильтровываются, и возвращаются сведения о неквалифицированных строках. В версиях ранее v1.5.2, когда `enable_insert_strict` установлено в `true`, задания INSERT завершаются неудачей при наличии неквалифицированных строк.

Примеры:

Stream Load

```
curl --location-trusted -u <username>:<password> \
    -H "strict_mode: {true | false}" \
    -T <file_name> -XPUT \
    http://<fe_host>:
<fe_http_port>/api/<database_name>/<table_name>/_stream_load
```

Подробный синтаксис и параметры Stream Load см. в STREAM LOAD.

Broker Load

```
LOAD LABEL [<database_name>.]<label_name>
(
    DATA INFILE ("<file_path>"[, "<file_path>" ...])
    INTO TABLE <table_name>
)
WITH BROKER
(
    "username" = "<hdfs_username>",
    "password" = "<hdfs_password>"
)
PROPERTIES
(
    "strict_mode" = "{true | false}"
)
```

В приведённом выше фрагменте кода в качестве примера используется HDFS.

Подробный синтаксис и параметры Broker Load см. в BROKER LOAD.

Routine Load

```
CREATE ROUTINE LOAD [<database_name>.]<job_name> ON <table_name>
PROPERTIES
(
    "strict_mode" = "{true | false}"
)
FROM KAFKA
(
    "kafka_broker_list" ="<kafka_broker1_ip>:<kafka_broker1_port>[,
<kafka_broker2_ip>:<kafka_broker2_port>...]",
    "kafka_topic" = "<topic_name>"
)
```

В приведённом выше фрагменте кода в качестве примера используется Apache Kafka. Подробный синтаксис и параметры Routine Load см. в CREATE ROUTINE LOAD.

Spark Load

```
LOAD LABEL [<database_name>.]<label_name>
(
    DATA INFILE ("<file_path>"[, "<file_path>" ...])
    INTO TABLE <table_name>
)
WITH RESOURCE <resource_name>
(
    "spark.executor.memory" = "3g",
    "broker.username" = "<hdfs_username>",
    "broker.password" = "<hdfs_password>"
)
PROPERTIES
(
    "strict_mode" = "{true | false}"
)
```

В приведённом выше фрагменте кода в качестве примера используется HDFS. Подробный синтаксис и параметры Spark Load см. в SPARK LOAD.

INSERT

```
INSERT INTO [<database_name>.]<table_name>
PROPERTIES(
    "strict_mode" = "{true | false}"
)
<query_statement>
```

Подробный синтаксис и параметры INSERT см. в INSERT.

4 Выгрузка данных

4.1 Экспорт данных с помощью EXPORT

В этой теме описывается, как экспортировать данные из указанных таблиц или partitions в вашем Selena cluster в виде CSV файлов данных во внешнюю систему хранения, которой может быть распределенная файловая система HDFS или облачная система хранения, такая как AWS S3.

> **ВАЖНО**
>
> Вы можете экспортировать данные из таблиц Selena только как пользователь, имеющий привилегию EXPORT для этих таблиц Selena. Если у вас нет привилегии EXPORT, следуйте инструкциям, приведенным в GRANT, чтобы предоставить привилегию EXPORT пользователю, которого вы используете для подключения к вашему Selena cluster.

4.1.1 Справочная информация

В версиях 2.4 и более ранних Selena зависела от brokers для установки соединений между вашим Selena cluster и внешней системой хранения при использовании инструкции EXPORT для экспорта данных. Поэтому вам нужно было указать `WITH BROKER "<broker_name>"` для указания broker, который вы хотите использовать в инструкции EXPORT. Это называется "выгрузка на основе broker". Broker — это независимый, stateless сервис, интегрированный с интерфейсом файловой системы, помогающий Selena экспортировать данные во внешнюю систему хранения.

Начиная с версии 2.5, Selena больше не зависит от brokers для установки соединений между вашим Selena cluster и внешней системой хранения при использовании инструкции EXPORT для экспорта данных. Поэтому вам больше не нужно указывать broker в инструкции EXPORT, но вам все еще нужно сохранить ключевое слово `WITH BROKER`. Это называется "выгрузка без broker".

Однако, когда ваши данные хранятся в HDFS, выгрузка без broker может не работать, и вы можете прибегнуть к выгрузке на основе broker:

- Если вы экспортируете данные в несколько HDFS clusters, вам нужно развернуть и настроить независимый broker для каждого из этих HDFS clusters.

- Если вы экспортируете данные в один HDFS cluster и у вас настроено несколько пользователей Kerberos, вам нужно развернуть один независимый broker.

4.1.2 Поддерживаемые системы хранения

- Распределенная файловая система HDFS
- Облачные системы хранения, такие как AWS S3

4.1.3 Меры предосторожности

- Мы рекомендуем, чтобы объем данных, экспортируемых за один раз, не превышал несколько десятков GB. Если вы экспортируете чрезмерно большой объем данных за один раз, экспорт может завершиться неудачей, и стоимость повторной попытки экспорта увеличивается.

- Если исходная таблица Selena содержит большой объем данных, мы рекомендуем экспортировать данные только из нескольких partitions таблицы за раз, пока все данные из таблицы не будут экспортированы.

- Если FE в вашем Selena cluster перезапускаются или выбирается новый leader FE во время выполнения задачи экспорта, задача экспорта завершается неудачей. В этой ситуации вы должны отправить задачу экспорта снова.

- Если FE в вашем Selena cluster перезапускаются или выбирается новый leader FE после завершения задачи экспорта, некоторая информация о задаче, возвращаемая инструкцией SHOW EXPORT, может быть потеряна.

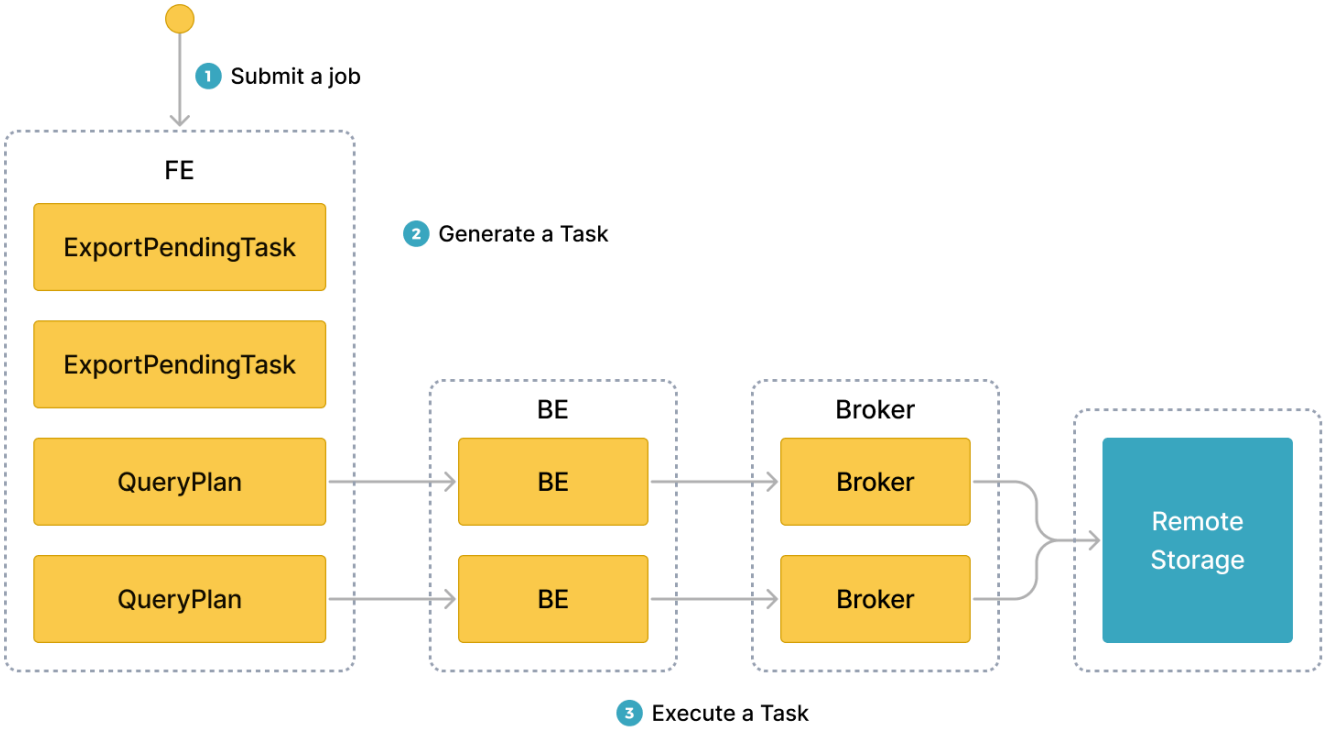
- Selena экспортирует только данные базовых таблиц. Selena не экспортирует данные материализованных представлений, созданных на базовых таблицах.

- Задачи экспорта требуют сканирования данных, что занимает I/O ресурсы и, следовательно, увеличивает задержку запросов.

4.1.4 Рабочий процесс

После отправки задачи экспорта Selena идентифицирует все tablets, участвующие в задаче экспорта. Затем Selena разделяет участвующие tablets на группы и генерирует планы запросов. Планы запросов используются для чтения данных из участвующих tablets и для записи данных в указанный путь целевой системы хранения.

Следующая диаграмма показывает общий рабочий процесс.



Общий рабочий процесс состоит из следующих трех шагов:

1. Пользователь отправляет задачу экспорта на leader FE.
2. Leader FE выдает инструкции `snapshot` всем BE или CN в Selena cluster, чтобы BE или CN могли делать snapshots участвующих tablets для обеспечения согласованности данных, которые нужно экспортировать. Leader FE также генерирует несколько задач экспорта. Каждая задача экспорта — это план запроса, и каждый план запроса используется для обработки части участвующих tablets.
3. Leader FE распределяет задачи экспорта между BE или CN.

4.1.5 Принципы работы

Когда Selena выполняет планы запросов, она сначала создает временную папку с именем `__selena_export_tmp_xxx` в указанном пути целевой системы хранения. В имени временной папки `xxx` представляет ID задачи экспорта. Примером имени временной папки является `__selena_export_tmp_921d8f80-7c9d-11eb-9342-acde48001122`. После успешного выполнения плана запроса Selena генерирует временный файл во временной папке и записывает экспортированные данные в сгенерированный временный файл.

После экспорта всех данных Selena использует инструкцию RENAME для сохранения сгенерированных временных файлов в указанный путь.

4.1.6 Связанные параметры

В этом разделе описываются некоторые параметры, связанные с экспортом, которые вы можете настроить в FE вашего Selena cluster.

- `export_checker_interval_second`: интервал, с которым планируются задачи экспорта. Интервал по умолчанию составляет 5 секунд. После перенастройки этого параметра для FE вам нужно перезапустить FE, чтобы новая настройка параметра вступила в силу.

- `export_running_job_num_limit`: максимальное количество выполняющихся задач экспорта, которое разрешено. Если количество выполняющихся задач экспорта превышает этот лимит, избыточные задачи экспорта переходят в состояние ожидания после выполнения `snapshot`. Максимальное количество по умолчанию равно 5. Вы можете перенастроить этот параметр во время выполнения задач экспорта.

- `export_task_default_timeout_second`: период timeout для задач экспорта. Период timeout по умолчанию составляет 2 часа. Вы можете перенастроить этот параметр во время выполнения задач экспорта.

- `export_max_bytes_per_be_per_task`: максимальный объем данных в сжатом виде, который может быть экспортирован на задачу экспорта с каждого BE или CN. Этот параметр предоставляет политику, на основе которой Selena разделяет задачи экспорта на задачи экспорта, которые могут выполняться одновременно. Максимальный объем по умолчанию составляет 256 MB.

- `export_task_pool_size`: максимальное количество задач экспорта, которые могут выполняться одновременно в thread pool. Максимальное количество по умолчанию равно 5.

4.1.7 Основные операции

Отправка задачи экспорта

Предположим, что ваша база данных Selena `db1` содержит таблицу с именем `tbl1`. Чтобы экспортировать данные колонок `col1` и `col3` из partitions `p1` и `p2` таблицы `tbl1` в

путь `export` вашего HDFS cluster, выполните следующую команду:

```
EXPORT TABLE db1.tbl1
PARTITION (p1,p2)
(col1, col3)
TO "hdfs://HDFS_IP:HDFS_Port/export/lineorder_"
PROPERTIES
(
    "column_separator"=",",
    "load_mem_limit"="2147483648",
    "timeout" = "3600"
)
WITH BROKER
(
    "username" = "user",
    "password" = "passwd"
);
```

Для подробного синтаксиса и описания параметров, а также примеров команд экспорта данных в AWS S3 см. EXPORT.

Получение query ID задачи экспорта

После отправки задачи экспорта вы можете использовать инструкцию SELECT LAST_QUERY_ID() для запроса query ID задачи экспорта. С query ID вы можете просматривать или отменять задачу экспорта.

Для подробного синтаксиса и описания параметров см. last_query_id.

Просмотр статуса задачи экспорта

После отправки задачи экспорта вы можете использовать инструкцию SHOW EXPORT для просмотра статуса задачи экспорта. Пример:

```
SHOW EXPORT WHERE queryid = "edee47f0-abe1-11ec-b9d1-00163e1e238f";
```

> **ПРИМЕЧАНИЕ**
>
> В приведенном выше примере `queryid` — это query ID задачи экспорта.

Возвращается информация, аналогичная следующему выводу:

```
JobId: 14008
State: FINISHED
Progress: 100%
TaskInfo: {"partitions":["*"],"mem limit":2147483648,"column
separator":",","line delimiter":"\n","tablet num":1,"broker":"hdfs","coord
num":1,"db":"default_cluster:db1","tbl":"tbl3",columns:["col1", "col3"]}
Path: oss://bj-test/export/
CreateTime: 2019-06-25 17:08:24
StartTime: 2019-06-25 17:08:28
FinishTime: 2019-06-25 17:08:34
Timeout: 3600
ErrorMsg: N/A
```

Для подробного синтаксиса и описания параметров см. SHOW EXPORT.

Отмена задачи экспорта

Вы можете использовать инструкцию CANCEL EXPORT для отмены задачи экспорта, которую вы отправили. Пример:

```
CANCEL EXPORT WHERE queryid = "921d8f80-7c9d-11eb-9342-acde48001122";
```

> **ПРИМЕЧАНИЕ**
>
> В приведенном выше примере `queryid` — это query ID задачи экспорта.

Для подробного синтаксиса и описания параметров см. CANCEL EXPORT.

4.1.8 Лучшие практики

Разделение плана запроса

Количество планов запросов, на которые разделяется задача экспорта, варьируется в зависимости от количества tablets, участвующих в задаче экспорта, и от максимального объема данных, который может быть обработан на план запроса. Задачи экспорта повторяются как планы запросов. Если объем данных, обрабатываемый планом запроса, превышает максимально допустимый объем, план запроса сталкивается с ошибками, такими как колебания в удаленном хранилище. В результате стоимость повторной попытки плана запроса увеличивается. Максимальный объем данных, который может быть обработан на план запроса каждым BE или CN, указывается параметром

`export_max_bytes_per_be_per_task`, который по умолчанию равен 256 MB. В плане запроса каждому BE или CN выделяется как минимум один tablet, и он может экспортировать объем данных, не превышающий лимит, указанный параметром `export_max_bytes_per_be_per_task`.

Несколько планов запросов задачи экспорта выполняются одновременно. Вы можете использовать параметр FE `export_task_pool_size` для указания максимального количества задач экспорта, которым разрешено одновременное выполнение в thread pool. По умолчанию этот параметр равен `5`.

В обычных случаях каждый план запроса задачи экспорта состоит только из двух частей: сканирование и экспорт. Логика выполнения вычислений, требуемых планами запросов, не потребляет много памяти. Поэтому лимит памяти по умолчанию в 2 GB может удовлетворить большинство ваших бизнес-требований. Однако в определенных обстоятельствах, таких как когда план запроса требует сканирования многих tablets на BE или CN, или tablet имеет много версий, емкости памяти в 2 GB может быть недостаточно. В этих обстоятельствах вам нужно использовать параметр `load_mem_limit` для указания более высокого лимита емкости памяти, такого как 4 GB или 8 GB.

4.2 Чтение данных из Selena с помощью Spark connector

Selena предоставляет собственный connector под названием Selena Connector for Apache Spark™ (сокращенно Spark connector), который помогает читать данные из таблицы Selena с помощью Spark. Вы можете использовать Spark для сложной обработки и машинного обучения на данных, прочитанных из Selena.

Spark connector поддерживает три метода чтения: Spark SQL, Spark DataFrame и Spark RDD.

Вы можете использовать Spark SQL для создания временного представления на таблице Selena, а затем напрямую читать данные из таблицы Selena, используя это временное представление.

Вы также можете сопоставить таблицу Selena с Spark DataFrame или Spark RDD, а затем читать данные из Spark DataFrame или Spark RDD. Мы рекомендуем использовать Spark DataFrame.

> **NOTICE**

> Только пользователи с привилегией SELECT на таблицу Selena могут читать данные из этой таблицы. Вы можете следовать инструкциям, предоставленным в GRANT, чтобы предоставить привилегию пользователю.

4.2.1 Примечания по использованию

- Вы можете фильтровать данные в Selena перед чтением, тем самым уменьшая объем передаваемых данных.
- Если накладные расходы на чтение данных существенны, вы можете использовать соответствующий дизайн таблицы и условия фильтрации, чтобы предотвратить чтение Spark чрезмерного объема данных за раз. Таким образом, вы можете снизить нагрузку на ввод-вывод диска и сетевое подключение, тем самым обеспечивая правильное выполнение обычных запросов.

4.2.2 Требования к версиям

Spark connector	Spark	Selena	Java	Scala
1.1.2	3.2, 3.3, 3.4, 3.5	2.5 и новее	8	2.12
1.1.1	3.2, 3.3, 3.4	2.5 и новее	8	2.12
1.1.0	3.2, 3.3, 3.4	2.5 и новее	8	2.12
1.0.0	3.x	1.18 и новее	8	2.12
1.0.0	2.x	1.18 и новее	8	2.11

> **NOTICE**
>
> - Пожалуйста, см. Обновление Spark connector для изменений в поведении между различными версиями connector.
> - Connector не предоставляет MySQL JDBC driver начиная с версии 1.1.1, и вам необходимо импортировать driver в classpath Spark вручную. Вы можете найти driver на Maven Central.
> - В версии 1.0.0 Spark connector поддерживает только чтение данных из Selena. Начиная с версии 1.1.0, Spark connector поддерживает как чтение данных из Selena, так и запись данных в Selena.
> - Версия 1.0.0 отличается от версии 1.1.0 по параметрам и сопоставлению типов данных. См. Обновление Spark connector.
> - В общих случаях новые функции не будут добавляться в версию 1.0.0. Мы рекомендуем обновить ваш Spark connector при первой возможности.

4.2.3 Получение Spark connector

Используйте один из следующих методов для получения пакета **.jar** Spark connector, который соответствует вашим бизнес-потребностям:

- Загрузить скомпилированный пакет.
- Использовать Maven для добавления зависимостей, требуемых Spark connector. (Этот метод поддерживается только для Spark connector 1.1.0 и новее.)
- Вручную скомпилировать пакет.

Spark connector 1.1.0 и новее

Пакеты **.jar** Spark connector названы в следующем формате:

`selena-spark-connector-${spark_version}_${scala_version}-${connector_version}.jar`

Например, если вы хотите использовать Spark connector 1.1.0 со Spark 3.2 и Scala 2.12, вы можете выбрать `selena-spark-connector-3.2_2.12-1.1.0.jar`.

> **NOTICE**
>
> В обычных случаях последняя версия Spark connector может использоваться с тремя последними версиями Spark.

Загрузка скомпилированного пакета

Вы можете получить пакеты **.jar** Spark connector различных версий на Maven Central Repository.

Добавление зависимостей Maven

Настройте зависимости, требуемые Spark connector, следующим образом:

> **NOTICE**
>
> Вы должны заменить `spark_version`, `scala_version` и `connector_version` на версию Spark, версию Scala и версию Spark connector, которые вы используете.

```
<dependency>
  <groupId>com.selena</groupId>
  <artifactId>selena-spark-connector-${spark_version}_${scala_version}
</artifactId>
  <version>${connector_version}</version>
</dependency>
```

Например, если вы хотите использовать Spark connector 1.1.0 со Spark 3.2 и Scala 2.12, настройте зависимости следующим образом:

```
<dependency>
  <groupId>com.selena</groupId>
  <artifactId>selena-spark-connector-3.2_2.12</artifactId>
  <version>1.1.0</version>
</dependency>
```

Ручная компиляция пакета

1. Загрузите код Spark connector.

2. Используйте следующую команду для компиляции Spark connector:

   > **NOTICE**
   >
   > Вы должны заменить `spark_version` на версию Spark, которую вы используете.

   ```
   sh build.sh <spark_version>
   ```

   Например, если вы хотите использовать Spark connector со Spark 3.2, скомпилируйте Spark connector следующим образом:

   ```
   sh build.sh 3.2
   ```

3. Перейдите по пути `target/`, где создается пакет **.jar** Spark connector, такой как `selena-spark-connector-3.2_2.12-1.1.0-SNAPSHOT.jar`, после компиляции.

   > **NOTICE**

> Если вы используете версию Spark connector, которая не выпущена официально, имя созданного пакета **.jar** Spark connector содержит `SNAPSHOT` в качестве суффикса.

Spark connector 1.0.0

Загрузка скомпилированного пакета

- Spark 2.x
- Spark 3.x

Ручная компиляция пакета

1. Загрузите код Spark connector.

   > **NOTICE**
   >
   > Вы должны переключиться на `spark-1.0`.

2. Выполните одно из следующих действий для компиляции Spark connector:

   - Если вы используете Spark 2.x, выполните следующую команду, которая компилирует Spark connector для Spark 2.3.4 по умолчанию:

     ```
     sh build.sh 2
     ```

   - Если вы используете Spark 3.x, выполните следующую команду, которая компилирует Spark connector для Spark 3.1.2 по умолчанию:

     ```
     sh build.sh 3
     ```

3. Перейдите по пути `output/`, где создается файл `selena-spark2_2.11-1.0.0.jar` после компиляции. Затем скопируйте файл в classpath Spark:

   - Если ваш кластер Spark работает в режиме `Local`, поместите файл в путь `jars/`.
   - Если ваш кластер Spark работает в режиме `Yarn`, поместите файл в пакет для предварительного развертывания.

Вы можете использовать Spark connector для чтения данных из Selena только после того, как поместите файл в указанное место.

4.2.4 Параметры

В этом разделе описаны параметры, которые вам необходимо настроить при использовании Spark connector для чтения данных из Selena.

Общие параметры

Следующие параметры применяются ко всем трем методам чтения: Spark SQL, Spark DataFrame и Spark RDD.

Параметр	Значение по умолчанию	Описание
selena.fenodes	None	HTTP URL FE в вашем кластере Selena. Формат `<fe_host>:<fe_http_port>`. Вы можете указать несколько URL, которые должны быть разделены запятой (,).
selena.table.identifier	None	Имя таблицы Selena. Формат: `<database_name>.<table_name>`.
selena.request.retries	3	Максимальное количество повторных попыток отправки запроса на чтение в Selena.
selena.request.connect.timeout.ms	30000	Максимальное время, после которого запрос на чтение, отправленный в Selena, истекает.
selena.request.read.timeout.ms	30000	Максимальное время, после которого чтение для запроса, отправленного в Selena, истекает.
selena.request.query.timeout.s	3600	Максимальное время, после которого запрос данных из Selena истекает. Период тайм-аута по умолчанию - 1 час. `-1` означает, что период тайм-аута не указан.
selena.request.tablet.size	Integer.MAX_VALUE	Количество tablet Selena, сгруппированных в каждый partition Spark RDD. Меньшее значение этого параметра означает, что будет создано большее количество partition Spark RDD. Большее количество partition Spark RDD означает более высокий параллелизм в Spark, но большее давление на Selena.
selena.batch.size	4096	Максимальное количество строк, которые могут быть прочитаны из BE за раз. Увеличение значения этого параметра может уменьшить количество соединений,

Параметр	Значение по умолчанию	Описание
		установленных между Spark и Selena, тем самым смягчая дополнительные временные издержки, вызванные задержкой сети.
selena.exec.mem.limit	2147483648	Максимальный объем памяти, разрешенный для одного запроса. Единица: байты. Ограничение памяти по умолчанию - 2 ГБ.
selena.deserialize.arrow.async	false	Указывает, поддерживается ли асинхронное преобразование формата памяти Arrow в RowBatches, необходимые для итерации Spark connector.
selena.deserialize.queue.size	64	Размер внутренней очереди, которая содержит задачи для асинхронного преобразования формата памяти Arrow в RowBatches. Этот параметр действителен, когда `selena.deserialize.arrow.async` установлен в `true`.
selena.filter.query	None	Условие, на основе которого вы хотите фильтровать данные в Selena. Вы можете указать несколько условий фильтрации, которые должны быть объединены с помощью `and`. Selena фильтрует данные из таблицы Selena на основе указанных условий фильтрации перед тем, как данные будут прочитаны Spark.
selena.timezone	Часовой пояс JVM по умолчанию	Поддерживается с версии 1.1.1. Часовой пояс, используемый для преобразования Selena `DATETIME` в Spark `TimestampType`. По умолчанию используется часовой пояс JVM, возвращаемый `ZoneId#systemDefault()`. Формат может быть именем часового пояса, таким как `Asia/Shanghai`, или смещением зоны, таким как `+08:00`.

Параметры для Spark SQL и Spark DataFrame

Следующие параметры применяются только к методам чтения Spark SQL и Spark DataFrame.

Параметр	Значение по умолчанию	Описание
selena.fe.http.url	None	HTTP IP-адрес FE. Этот параметр поддерживается начиная с Spark connector 1.1.0. Этот параметр эквивалентен `selena.fenodes`. Вам нужно настроить только один из них. В Spark connector 1.1.0 и новее мы рекомендуем использовать `selena.fe.http.url`, потому что `selena.fenodes` может быть устаревшим.
selena.fe.jdbc.url	None	Адрес, используемый для подключения к MySQL серверу FE. Формат: `jdbc:mysql://<fe_host>:<fe_query_port>`. **NOTICE** В Spark connector 1.1.0 и новее этот параметр обязателен.

Параметр	Значение по умолчанию	Описание
user	None	Имя пользователя учетной записи кластера Selena. Пользователю нужна привилегия SELECT на таблицу Selena.
selena.user	None	Имя пользователя учетной записи кластера Selena. Этот параметр поддерживается начиная с Spark connector 1.1.0. Этот параметр эквивалентен `user`. Вам нужно настроить только один из них. В Spark connector 1.1.0 и новее мы рекомендуем использовать `selena.user`, потому что `user` может быть устаревшим.
password	None	Пароль учетной записи кластера Selena.
selena.password	None	Пароль учетной записи кластера Selena. Этот параметр поддерживается начиная с Spark connector 1.1.0. Этот параметр эквивалентен `password`. Вам нужно настроить только один из них. В Spark connector 1.1.0 и новее мы рекомендуем использовать `selena.password`, потому что `password` может быть устаревшим.
selena.filter.query.in.max.count	100	Максимальное количество значений, поддерживаемых выражением IN во время проталкивания предиката. Если количество значений, указанных в выражении IN, превышает этот лимит, условия фильтрации, указанные в выражении IN, обрабатываются в Spark.

Параметры для Spark RDD

Следующие параметры применяются только к методу чтения Spark RDD.

Параметр	Значение по умолчанию	Описание
selena.request.auth.user	None	Имя пользователя учетной записи кластера Selena.
selena.request.auth.password	None	Пароль учетной записи кластера Selena.
selena.read.field	None	Столбец таблицы Selena, из которого вы хотите читать данные. Вы можете указать несколько столбцов, которые должны быть разделены запятой (,).

4.2.5 Сопоставление типов данных между Selena и Spark

Spark connector 1.1.0 и новее

Тип данных Selena	Тип данных Spark
BOOLEAN	DataTypes.BooleanType
TINYINT	DataTypes.ByteType
SMALLINT	DataTypes.ShortType

Тип данных Selena	Тип данных Spark
INT	DataTypes.IntegerType
BIGINT	DataTypes.LongType
LARGEINT	DataTypes.StringType
FLOAT	DataTypes.FloatType
DOUBLE	DataTypes.DoubleType
DECIMAL	DecimalType
CHAR	DataTypes.StringType
VARCHAR	DataTypes.StringType
STRING	DataTypes.StringType
DATE	DataTypes.DateType
DATETIME	DataTypes.TimestampType
JSON	DataTypes.StringType **NOTE:** Это сопоставление типов данных поддерживается с Spark connector v1.1.2 и требует версии Selena не менее 2.5.13, 3.0.3, 3.1.0 или новее.
ARRAY	Неподдерживаемый тип данных
HLL	Неподдерживаемый тип данных
BITMAP	Неподдерживаемый тип данных

Spark connector 1.0.0

Тип данных Selena	Тип данных Spark
BOOLEAN	DataTypes.BooleanType
TINYINT	DataTypes.ByteType
SMALLINT	DataTypes.ShortType
INT	DataTypes.IntegerType
BIGINT	DataTypes.LongType
LARGEINT	DataTypes.StringType
FLOAT	DataTypes.FloatType
DOUBLE	DataTypes.DoubleType
DECIMAL	DecimalType
CHAR	DataTypes.StringType
VARCHAR	DataTypes.StringType
DATE	DataTypes.StringType
DATETIME	DataTypes.StringType

Тип данных Selena	Тип данных Spark
ARRAY	Неподдерживаемый тип данных
HLL	Неподдерживаемый тип данных
BITMAP	Неподдерживаемый тип данных

Логика обработки базового движка хранения, используемого Selena, не может покрыть ожидаемый диапазон времени при прямом использовании типов данных DATE и DATETIME. Поэтому Spark connector сопоставляет типы данных DATE и DATETIME из Selena с типом данных STRING из Spark и генерирует читаемые текстовые строки, соответствующие данным даты и времени, прочитанным из Selena.

4.2.6 Обновление Spark connector

Обновление с версии 1.0.0 до версии 1.1.0

- Начиная с 1.1.1, Spark connector не предоставляет `mysql-connector-java`, который является официальным JDBC драйвером для MySQL, из-за ограничений лицензии GPL, используемой `mysql-connector-java`. Однако Spark connector все еще нуждается в `mysql-connector-java` для подключения к Selena для получения метаданных таблицы, поэтому вам нужно добавить драйвер в classpath Spark вручную. Вы можете найти драйвер на сайте MySQL или Maven Central.

- В версии 1.1.0 Spark connector использует JDBC для доступа к Selena для получения более детальной информации о таблице. Поэтому вы должны настроить `selena.fe.jdbc.url`.

- В версии 1.1.0 некоторые параметры переименованы. Оба старых и новых параметра сохранены на данный момент. Для каждой пары эквивалентных параметров вам нужно настроить только один из них, но мы рекомендуем использовать новый, потому что старый может быть устаревшим.

  - `selena.fenodes` переименован в `selena.fe.http.url`.
  - `user` переименован в `selena.user`.
  - `password` переименован в `selena.password`.
- В версии 1.1.0 сопоставления некоторых типов данных скорректированы на основе Spark 3.x:

  - `DATE` в Selena сопоставлен с `DataTypes.DateType` (изначально `DataTypes.StringType`) в Spark.

- `DATETIME` в Selena сопоставлен с `DataTypes.TimestampType` (изначально `DataTypes.StringType`) в Spark.

4.2.7 Примеры

Следующие примеры предполагают, что вы создали базу данных с именем `test` в вашем кластере Selena и у вас есть разрешения пользователя `root`. Настройки параметров в примерах основаны на Spark Connector 1.1.0.

Настройка сети

Убедитесь, что машина, на которой расположен Spark, может получить доступ к узлам FE кластера Selena через `http_port` (по умолчанию: `8030`) и `query_port` (по умолчанию: `9030`), а также к узлам BE через `be_port` (по умолчанию: `9060`).

Пример данных

Выполните следующие действия для подготовки примера таблицы:

1. Перейдите в базу данных `test` и создайте таблицу с именем `score_board`.

```
MySQL [test]> CREATE TABLE `score_board`
(
    `id` int(11) NOT NULL COMMENT "",
    `name` varchar(65533) NULL DEFAULT "" COMMENT "",
    `score` int(11) NOT NULL DEFAULT "0" COMMENT ""
)
ENGINE=OLAP
PRIMARY KEY(`id`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`id`)
PROPERTIES (
    "replication_num" = "3"
);
```

2. Вставьте данные в таблицу `score_board`.

```
MySQL [test]> INSERT INTO score_board
VALUES
    (1, 'Bob', 21),
    (2, 'Stan', 21),
    (3, 'Sam', 22),
    (4, 'Tony', 22),
    (5, 'Alice', 22),
    (6, 'Lucy', 23),
    (7, 'Polly', 23),
    (8, 'Tom', 23),
    (9, 'Rose', 24),
    (10, 'Jerry', 24),
    (11, 'Jason', 24),
    (12, 'Lily', 25),
    (13, 'Stephen', 25),
    (14, 'David', 25),
    (15, 'Eddie', 26),
    (16, 'Kate', 27),
    (17, 'Cathy', 27),
    (18, 'Judy', 27),
    (19, 'Julia', 28),
    (20, 'Robert', 28),
    (21, 'Jack', 29);
```

3. Запросите таблицу `score_board`.

```
MySQL [test]> SELECT * FROM score_board;
+------+---------+-------+
| id   | name    | score |
+------+---------+-------+
|    1 | Bob     |    21 |
|    2 | Stan    |    21 |
|    3 | Sam     |    22 |
|    4 | Tony    |    22 |
|    5 | Alice   |    22 |
|    6 | Lucy    |    23 |
|    7 | Polly   |    23 |
|    8 | Tom     |    23 |
|    9 | Rose    |    24 |
|   10 | Jerry   |    24 |
|   11 | Jason   |    24 |
|   12 | Lily    |    25 |
|   13 | Stephen |    25 |
|   14 | David   |    25 |
|   15 | Eddie   |    26 |
|   16 | Kate    |    27 |
|   17 | Cathy   |    27 |
|   18 | Judy    |    27 |
|   19 | Julia   |    28 |
|   20 | Robert  |    28 |
|   21 | Jack    |    29 |
+------+---------+-------+
21 rows in set (0.01 sec)
```

Чтение данных с помощью Spark SQL

1. Выполните следующую команду в директории Spark для запуска Spark SQL:

```
sh spark-sql
```

2. Выполните следующую команду для создания временного представления с именем `spark_selena` на таблице `score_board`, которая принадлежит базе данных `test`:

```
spark-sql> CREATE TEMPORARY VIEW spark_selena
           USING selena
           OPTIONS
           (
               "selena.table.identifier" = "test.score_board",
               "selena.fe.http.url" = "<fe_host>:<fe_http_port>",
               "selena.fe.jdbc.url" = "jdbc:mysql://<fe_host>:
<fe_query_port>",
               "selena.user" = "root",
               "selena.password" = ""
           );
```

3. Выполните следующую команду для чтения данных из временного представления:

```
spark-sql> SELECT * FROM spark_selena;
```

Spark возвращает следующие данные:

```
1         Bob         21
2         Stan         21
3         Sam         22
4         Tony         22
5         Alice         22
6         Lucy         23
7         Polly         23
8         Tom         23
9         Rose         24
10         Jerry         24
11         Jason         24
12         Lily         25
13         Stephen         25
14         David         25
15         Eddie         26
16         Kate         27
17         Cathy         27
18         Judy         27
19         Julia         28
20         Robert         28
21         Jack         29
Time taken: 1.883 seconds, Fetched 21 row(s)
22/08/09 15:29:36 INFO thriftserver.SparkSQLCLIDriver: Time taken:
1.883 seconds, Fetched 21 row(s)
```

Чтение данных с помощью Spark DataFrame

1. Выполните следующую команду в директории Spark для запуска Spark Shell:

```
sh spark-shell
```

2. Выполните следующую команду для создания DataFrame с именем `selenaSparkDF` на таблице `score_board`, которая принадлежит базе данных `test`:

```
scala> val selenaSparkDF = spark.read.format("selena")
           .option("selena.table.identifier", s"test.score_board")
           .option("selena.fe.http.url", s"<fe_host>:<fe_http_port>")
           .option("selena.fe.jdbc.url", s"jdbc:mysql://<fe_host>:
<fe_query_port>")
           .option("selena.user", s"root")
           .option("selena.password", s"")
           .load()
```

3. Прочитайте данные из DataFrame. Например, если вы хотите прочитать первые 10 строк, выполните следующую команду:

```
scala> selenaSparkDF.show(10)
```

Spark возвращает следующие данные:

```
+---+-----+-----+
| id| name|score|
+---+-----+-----+
|  1|  Bob|   21|
|  2| Stan|   21|
|  3|  Sam|   22|
|  4| Tony|   22|
|  5|Alice|   22|
|  6| Lucy|   23|
|  7|Polly|   23|
|  8|  Tom|   23|
|  9| Rose|   24|
| 10|Jerry|   24|
+---+-----+-----+
only showing top 10 rows
```

> **NOTE**
>
> По умолчанию, если вы не указываете количество строк, которые хотите прочитать, Spark возвращает первые 20 строк.

Чтение данных с помощью Spark RDD

1. Выполните следующую команду в директории Spark для запуска Spark Shell:

```
sh spark-shell
```

2. Выполните следующую команду для создания RDD с именем `selenaSparkRDD` на таблице `score_board`, которая принадлежит базе данных `test`.

```
scala> import com.selena.connector.spark._
scala> val selenaSparkRDD = sc.selenaRDD
           (
           tableIdentifier = Some("test.score_board"),
           cfg = Some(Map(
               "selena.fenodes" -> "<fe_host>:<fe_http_port>",
               "selena.request.auth.user" -> "root",
               "selena.request.auth.password" -> ""
           ))
           )
```

3. Прочитайте данные из RDD. Например, если вы хотите прочитать первые 10 элементов, выполните следующую команду:

```
scala> selenaSparkRDD.take(10)
```

Spark возвращает следующие данные:

```
res0: Array[AnyRef] = Array([1, Bob, 21], [2, Stan, 21], [3, Sam, 22],
[4, Tony, 22], [5, Alice, 22], [6, Lucy, 23], [7, Polly, 23], [8, Tom,
23], [9, Rose, 24], [10, Jerry, 24])
```

Чтобы прочитать весь RDD, выполните следующую команду:

```
scala> selenaSparkRDD.collect()
```

Spark возвращает следующие данные:

```
res1: Array[AnyRef] = Array([1, Bob, 21], [2, Stan, 21], [3, Sam, 22],
[4, Tony, 22], [5, Alice, 22], [6, Lucy, 23], [7, Polly, 23], [8, Tom,
23], [9, Rose, 24], [10, Jerry, 24], [11, Jason, 24], [12, Lily, 25],
[13, Stephen, 25], [14, David, 25], [15, Eddie, 26], [16, Kate, 27],
[17, Cathy, 27], [18, Judy, 27], [19, Julia, 28], [20, Robert, 28],
[21, Jack, 29])
```

4.2.8 Лучшие практики

При чтении данных из Selena с помощью Spark connector вы можете использовать параметр `selena.filter.query` для указания условий фильтрации, на основе которых Spark обрезает partition, bucket и префиксные индексы, чтобы уменьшить стоимость извлечения данных. В этом разделе Spark DataFrame используется в качестве примера, чтобы показать, как это достигается.

Настройка среды

Компонент	Версия
Spark	Spark 2.4.4 и Scala 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_302)
Selena	2.2.0
Spark connector	selena-spark2_2.11-1.0.0.jar

Пример данных

Выполните следующие действия для подготовки примера таблицы:

1. Перейдите в базу данных `test` и создайте таблицу с именем `mytable`.

```
MySQL [test]> CREATE TABLE `mytable`
(
    `k` int(11) NULL COMMENT "bucket",
    `b` int(11) NULL COMMENT "",
    `dt` datetime NULL COMMENT "",
    `v` int(11) NULL COMMENT ""
)
ENGINE=OLAP
DUPLICATE KEY(`k`,`b`, `dt`)
COMMENT "OLAP"
PARTITION BY RANGE(`dt`)
(
    PARTITION p202201 VALUES [('2022-01-01 00:00:00'), ('2022-02-01
00:00:00')),
    PARTITION p202202 VALUES [('2022-02-01 00:00:00'), ('2022-03-01
00:00:00')),
    PARTITION p202203 VALUES [('2022-03-01 00:00:00'), ('2022-04-01
00:00:00'))
)
DISTRIBUTED BY HASH(`k`)
PROPERTIES (
    "replication_num" = "3"
);
```

2. Вставьте данные в `mytable`.

```
MySQL [test]> INSERT INTO mytable
VALUES
     (1, 11, '2022-01-02 08:00:00', 111),
     (2, 22, '2022-02-02 08:00:00', 222),
     (3, 33, '2022-03-02 08:00:00', 333);
```

3. Запросите таблицу `mytable`.

```
MySQL [test]> select * from mytable;
+------+------+---------------------+------+
| k    | b    | dt                  | v    |
+------+------+---------------------+------+
|    1 |   11 | 2022-01-02 08:00:00 |  111 |
|    2 |   22 | 2022-02-02 08:00:00 |  222 |
|    3 |   33 | 2022-03-02 08:00:00 |  333 |
+------+------+---------------------+------+
3 rows in set (0.01 sec)
```

Полное сканирование таблицы

1. Выполните следующую команду в директории Spark для создания DataFrame с именем `df` на таблице `mytable`, которая принадлежит базе данных `test`:

```
scala>  val df = spark.read.format("selena")
        .option("selena.table.identifier", s"test.mytable")
        .option("selena.fenodes", s"<fe_host>:<fe_http_port>")
        .option("user", s"root")
        .option("password", s"")
        .load()
```

2. Просмотрите файл журнала FE **fe.log** вашего кластера Selena и найдите SQL-оператор, выполненный для чтения данных. Пример:

```
2022-08-09 18:57:38,091 INFO (nioEventLoopGroup-3-10|196)
[TableQueryPlanAction.executeWithoutPassword():126] receive SQL
statement [select `k`,`b`,`dt`,`v` from `test`.`mytable`] from external
service [ user ['root'@'%']] for database [test] table [mytable]
```

3. В базе данных `test` используйте EXPLAIN для получения плана выполнения оператора SELECT `k`,`b`,`dt`,`v` from `test`.`mytable`:

```
MySQL [test]> EXPLAIN select `k`,`b`,`dt`,`v` from `test`.`mytable`;
+-----------------------------------------------------------------------
-+
| Explain String
|
+-----------------------------------------------------------------------
-+
| PLAN FRAGMENT 0
|
|  OUTPUT EXPRS:1: k | 2: b | 3: dt | 4: v
|
|   PARTITION: UNPARTITIONED
|
|
|
|   RESULT SINK
|
|
|
|   1:EXCHANGE
|
|
|
| PLAN FRAGMENT 1
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 01
|
|     UNPARTITIONED
|
|
|
|   0:OlapScanNode
|
|      TABLE: mytable
|
|      PREAGGREGATION: ON
|
```

```
|      partitions=3/3
|
|      rollup: mytable
|
|      tabletRatio=9/9
|
|      tabletList=41297,41299,41301,41303,41305,41307,41309,41311,41313
|
|      cardinality=3
|
|      avgRowSize=4.0
|
|      numNodes=0
|
+-----------------------------------------------------------------------
-+
26 rows in set (0.00 sec)
```

В этом примере обрезка не выполняется. Поэтому Spark сканирует все три partition (как указано `partitions=3/3`), которые содержат данные, и сканирует все 9 tablet (как указано `tabletRatio=9/9`) в этих трех partition.

Обрезка partition

1. Выполните следующую команду, в которой вы используете параметр `selena.filter.query` для указания условия фильтрации `dt='2022-01-02 08:00:00` для обрезки partition, в директории Spark для создания DataFrame с именем `df` на таблице `mytable`, которая принадлежит базе данных `test`:

```
scala> val df = spark.read.format("selena")
       .option("selena.table.identifier", s"test.mytable")
       .option("selena.fenodes", s"<fe_host>:<fe_http_port>")
       .option("user", s"root")
       .option("password", s"")
       .option("selena.filter.query", "dt='2022-01-02 08:00:00'")
       .load()
```

2. Просмотрите файл журнала FE **fe.log** вашего кластера Selena и найдите SQL-оператор, выполненный для чтения данных. Пример:

```
2022-08-09 19:02:31,253 INFO (nioEventLoopGroup-3-14|204)
[TableQueryPlanAction.executeWithoutPassword():126] receive SQL
statement [select `k`,`b`,`dt`,`v` from `test`.`mytable` where
dt='2022-01-02 08:00:00'] from external service [ user ['root'@'%']]
for database [test] table [mytable]
```

3. В базе данных `test` используйте EXPLAIN для получения плана выполнения оператора SELECT `k`,`b`,`dt`,`v` from `test`.`mytable` where dt='2022-01-02 08:00:00':

```
MySQL [test]> EXPLAIN select `k`,`b`,`dt`,`v` from `test`.`mytable`
where dt='2022-01-02 08:00:00';
+-----------------------------------------------+
| Explain String                                |
+-----------------------------------------------+
| PLAN FRAGMENT 0                               |
|  OUTPUT EXPRS:1: k | 2: b | 3: dt | 4: v      |
|   PARTITION: UNPARTITIONED                    |
|                                               |
|   RESULT SINK                                 |
|                                               |
|   1:EXCHANGE                                  |
|                                               |
| PLAN FRAGMENT 1                               |
|  OUTPUT EXPRS:                                |
|   PARTITION: RANDOM                           |
|                                               |
|   STREAM DATA SINK                            |
|     EXCHANGE ID: 01                           |
|     UNPARTITIONED                             |
|                                               |
|   0:OlapScanNode                              |
|      TABLE: mytable                           |
|      PREAGGREGATION: ON                       |
|      PREDICATES: 3: dt = '2022-01-02 08:00:00' |
|      partitions=1/3                           |
|      rollup: mytable                          |
|      tabletRatio=3/3                          |
|      tabletList=41297,41299,41301             |
|      cardinality=1                            |
|      avgRowSize=20.0                          |
|      numNodes=0                               |
+-----------------------------------------------+
27 rows in set (0.01 sec)
```

В этом примере выполняется только обрезка partition, в то время как обрезка bucket не выполняется. Поэтому Spark сканирует один из трех partition (как указано `partitions=1/3`) и все tablet (как указано `tabletRatio=3/3`) в этом partition.

Обрезка bucket

1. Выполните следующую команду, в которой вы используете параметр `selena.filter.query` для указания условия фильтрации `k=1` для обрезки bucket, в

директории Spark для создания DataFrame с именем `df` на таблице `mytable`, которая принадлежит базе данных `test`:

```
scala> val df = spark.read.format("selena")
       .option("selena.table.identifier", s"test.mytable")
       .option("selena.fenodes", s"<fe_host>:<fe_http_port>")
       .option("user", s"root")
       .option("password", s"")
       .option("selena.filter.query", "k=1")
       .load()
```

2. Просмотрите файл журнала FE **fe.log** вашего кластера Selena и найдите SQL-оператор, выполненный для чтения данных. Пример:

```
2022-08-09 19:04:44,479 INFO (nioEventLoopGroup-3-16|208)
[TableQueryPlanAction.executeWithoutPassword():126] receive SQL
statement [select `k`,`b`,`dt`,`v` from `test`.`mytable` where k=1]
from external service [ user ['root'@'%']] for database [test] table
[mytable]
```

3. В базе данных `test` используйте EXPLAIN для получения плана выполнения оператора SELECT `k`,`b`,`dt`,`v` from `test`.`mytable` where k=1:

```
MySQL [test]> EXPLAIN select `k`,`b`,`dt`,`v` from `test`.`mytable`
where k=1;
+-----------------------------------------+
| Explain String                          |
+-----------------------------------------+
| PLAN FRAGMENT 0                         |
|  OUTPUT EXPRS:1: k | 2: b | 3: dt | 4: v |
|   PARTITION: UNPARTITIONED              |
|                                         |
|   RESULT SINK                           |
|                                         |
|   1:EXCHANGE                            |
|                                         |
| PLAN FRAGMENT 1                         |
|  OUTPUT EXPRS:                          |
|   PARTITION: RANDOM                     |
|                                         |
|   STREAM DATA SINK                      |
|     EXCHANGE ID: 01                     |
|     UNPARTITIONED                       |
|                                         |
|   0:OlapScanNode                        |
|      TABLE: mytable                     |
|      PREAGGREGATION: ON                 |
|      PREDICATES: 1: k = 1               |
|      partitions=3/3                     |
|      rollup: mytable                    |
|      tabletRatio=3/9                    |
|      tabletList=41299,41305,41311       |
|      cardinality=1                      |
|      avgRowSize=20.0                    |
|      numNodes=0                         |
+-----------------------------------------+
27 rows in set (0.01 sec)
```

В этом примере выполняется только обрезка bucket, в то время как обрезка partition не выполняется. Поэтому Spark сканирует все три partition (как указано `partitions=3/3`), которые содержат данные, и сканирует все три tablet (как указано `tabletRatio=3/9`) для получения значений Hash, которые соответствуют условию фильтрации `k = 1` в этих трех partition.

Обрезка partition и bucket

1. Выполните следующую команду, в которой вы используете параметр `selena.filter.query` для указания двух условий фильтрации `k=7` и `dt='2022-01-02 08:00:00'` для обрезки bucket и partition, в директории Spark для создания DataFrame с именем `df` на таблице `mytable` в базе данных `test`:

```
scala> val df = spark.read.format("selena")
       .option("selena.table.identifier", s"test.mytable")
       .option("selena.fenodes", s"<fe_host>:<fe_http_port>")
       .option("user", s"")
       .option("password", s"")
       .option("selena.filter.query", "k=7 and dt='2022-01-02
08:00:00'")
       .load()
```

2. Просмотрите файл журнала FE **fe.log** вашего кластера Selena и найдите SQL-оператор, выполненный для чтения данных. Пример:

```
2022-08-09 19:06:34,939 INFO (nioEventLoopGroup-3-18|212)
[TableQueryPlanAction.executeWithoutPassword():126] receive SQL
statement [select `k`,`b`,`dt`,`v` from `test`.`mytable` where k=7 and
dt='2022-01-02 08:00:00'] from external service [ user ['root'@'%']]
for database [test] t
able [mytable]
```

3. В базе данных `test` используйте EXPLAIN для получения плана выполнения оператора SELECT `k`,`b`,`dt`,`v` from `test`.`mytable` where k=7 and dt='2022-01-02 08:00:00':

```
MySQL [test]> EXPLAIN select `k`,`b`,`dt`,`v` from `test`.`mytable`
where k=7 and dt='2022-01-02 08:00:00';
+-----------------------------------------------------------+
| Explain String                                            |
+-----------------------------------------------------------+
| PLAN FRAGMENT 0                                           |
|  OUTPUT EXPRS:1: k | 2: b | 3: dt | 4: v                 |
|   PARTITION: RANDOM                                       |
|                                                           |
|   RESULT SINK                                             |
|                                                           |
|   0:OlapScanNode                                          |
|      TABLE: mytable                                       |
|      PREAGGREGATION: ON                                   |
|      PREDICATES: 1: k = 7, 3: dt = '2022-01-02 08:00:00' |
|      partitions=1/3                                       |
|      rollup: mytable                                      |
|      tabletRatio=1/3                                      |
|      tabletList=41301                                     |
|      cardinality=1                                        |
|      avgRowSize=20.0                                      |
|      numNodes=0                                           |
+-----------------------------------------------------------+
17 rows in set (0.00 sec)
```

В этом примере выполняется как обрезка partition, так и обрезка bucket. Поэтому Spark сканирует только один из трех partition (как указано `partitions=1/3`) и только один tablet (как указано `tabletRatio=1/3`) в этом partition.

Фильтрация по префиксному индексу

1. Вставьте больше записей данных в partition таблицы `mytable`, которая принадлежит базе данных `test`:

```
MySQL [test]> INSERT INTO mytable
VALUES
    (1, 11, "2022-01-02 08:00:00", 111),
    (3, 33, "2022-01-02 08:00:00", 333),
    (3, 33, "2022-01-02 08:00:00", 333),
    (3, 33, "2022-01-02 08:00:00", 333);
```

2. Запросите таблицу `mytable`:

```
MySQL [test]> SELECT * FROM mytable;
+------+------+---------------------+------+
| k    | b    | dt                  | v    |
+------+------+---------------------+------+
|    1 |   11 | 2022-01-02 08:00:00 |  111 |
|    3 |   33 | 2022-01-02 08:00:00 |  333 |
|    3 |   33 | 2022-01-02 08:00:00 |  333 |
|    3 |   33 | 2022-01-02 08:00:00 |  333 |
+------+------+---------------------+------+
4 rows in set (0.01 sec)
```

3. Выполните следующую команду, в которой вы используете параметр `selena.filter.query` для указания условия фильтрации `k=1` для фильтрации по префиксному индексу, в директории Spark для создания DataFrame с именем `df` на таблице `mytable`, которая принадлежит базе данных `test`:

```
scala> val df = spark.read.format("selena")
       .option("selena.table.identifier", s"test.mytable")
       .option("selena.fenodes", s"<fe_host>:<fe_http_port>")
       .option("user", s"root")
       .option("password", s"")
       .option("selena.filter.query", "k=1")
       .load()
```

4. В базе данных `test` установите `is_report_success` в `true`, чтобы включить отчет профиля:

```
MySQL [test]> SET is_report_success = true;
Query OK, 0 rows affected (0.00 sec)
```

5. Используйте браузер для открытия страницы `http://<fe_host>:<http_http_port>/query` и просмотрите профиль оператора SELECT * FROM mytable where k=1. Пример:

```
OLAP_SCAN (plan_node_id=0):
  CommonMetrics:
     - CloseTime: 1.255ms
     - OperatorTotalTime: 1.404ms
     - PeakMemoryUsage: 0.00
     - PullChunkNum: 8
     - PullRowNum: 2
       - __MAX_OF_PullRowNum: 2
       - __MIN_OF_PullRowNum: 0
     - PullTotalTime: 148.60us
     - PushChunkNum: 0
     - PushRowNum: 0
     - PushTotalTime: 0ns
     - SetFinishedTime: 136ns
     - SetFinishingTime: 129ns
  UniqueMetrics:
     - Predicates: 1: k = 1
     - Rollup: mytable
     - Table: mytable
     - BytesRead: 88.00 B
       - __MAX_OF_BytesRead: 88.00 B
       - __MIN_OF_BytesRead: 0.00
     - CachedPagesNum: 0
     - CompressedBytesRead: 844.00 B
       - __MAX_OF_CompressedBytesRead: 844.00 B
       - __MIN_OF_CompressedBytesRead: 0.00
     - CreateSegmentIter: 18.582us
     - IOTime: 4.425us
     - LateMaterialize: 17.385us
     - PushdownPredicates: 3
     - RawRowsRead: 2
       - __MAX_OF_RawRowsRead: 2
       - __MIN_OF_RawRowsRead: 0
     - ReadPagesNum: 12
       - __MAX_OF_ReadPagesNum: 12
       - __MIN_OF_ReadPagesNum: 0
     - RowsRead: 2
       - __MAX_OF_RowsRead: 2
       - __MIN_OF_RowsRead: 0
     - ScanTime: 154.367us
     - SegmentInit: 95.903us
       - BitmapIndexFilter: 0ns
       - BitmapIndexFilterRows: 0
       - BloomFilterFilterRows: 0
       - ShortKeyFilterRows: 3
```

```
          - __MAX_OF_ShortKeyFilterRows: 3
          - __MIN_OF_ShortKeyFilterRows: 0
        - ZoneMapIndexFilterRows: 0
      - SegmentRead: 2.559us
        - BlockFetch: 2.187us
        - BlockFetchCount: 2
          - __MAX_OF_BlockFetchCount: 2
          - __MIN_OF_BlockFetchCount: 0
        - BlockSeek: 7.789us
        - BlockSeekCount: 2
          - __MAX_OF_BlockSeekCount: 2
          - __MIN_OF_BlockSeekCount: 0
        - ChunkCopy: 25ns
        - DecompressT: 0ns
        - DelVecFilterRows: 0
        - IndexLoad: 0ns
        - PredFilter: 353ns
        - PredFilterRows: 0
        - RowsetsReadCount: 7
        - SegmentsReadCount: 3
          - __MAX_OF_SegmentsReadCount: 2
          - __MIN_OF_SegmentsReadCount: 0
        - TotalColumnsDataPageCount: 8
          - __MAX_OF_TotalColumnsDataPageCount: 8
          - __MIN_OF_TotalColumnsDataPageCount: 0
      - UncompressedBytesRead: 508.00 B
        - __MAX_OF_UncompressedBytesRead: 508.00 B
        - __MIN_OF_UncompressedBytesRead: 0.00
```

В этом примере условие фильтрации `k = 1` может попасть в префиксный индекс. Поэтому Spark может отфильтровать три строки (как указано `ShortKeyFilterRows: 3`).

4.3 Чтение данных из Selena с помощью Flink connector

Selena предоставляет собственный connector под названием Selena Connector for Apache Flink® (сокращенно Flink connector), который помогает вам читать данные массово из Selena cluster с помощью Flink.

Flink connector поддерживает два метода чтения: Flink SQL и Flink DataStream. Рекомендуется использовать Flink SQL.

> **ПРИМЕЧАНИЕ**

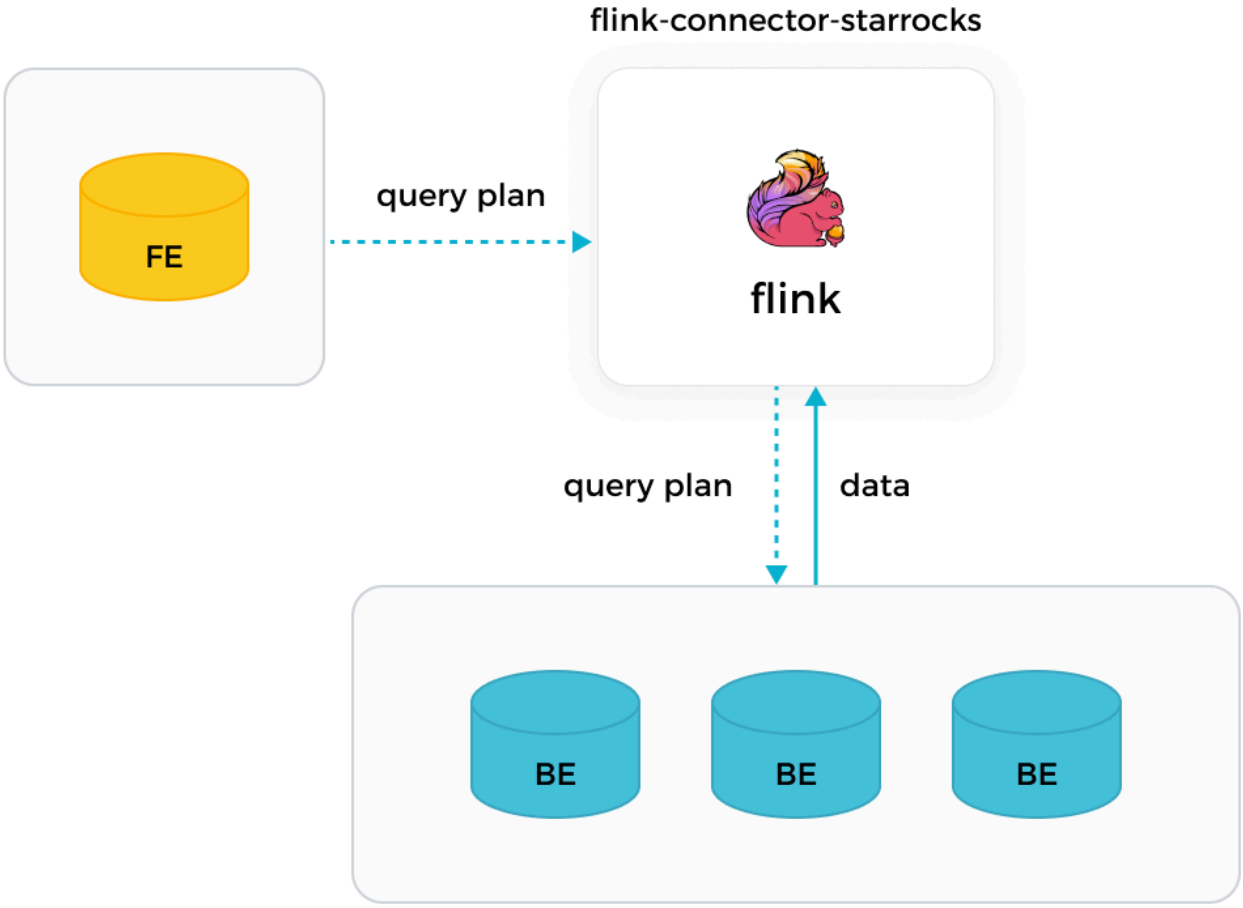
> Flink connector также поддерживает запись данных, прочитанных Flink, в другой Selena cluster или систему хранения. См. Непрерывная загрузка данных из Apache Flink®.

4.3.1 Справочная информация

В отличие от JDBC connector, предоставляемого Flink, Flink connector от Selena поддерживает параллельное чтение данных из нескольких BE вашего Selena cluster, значительно ускоряя задачи чтения. Следующее сравнение показывает разницу в реализации между двумя connectors.

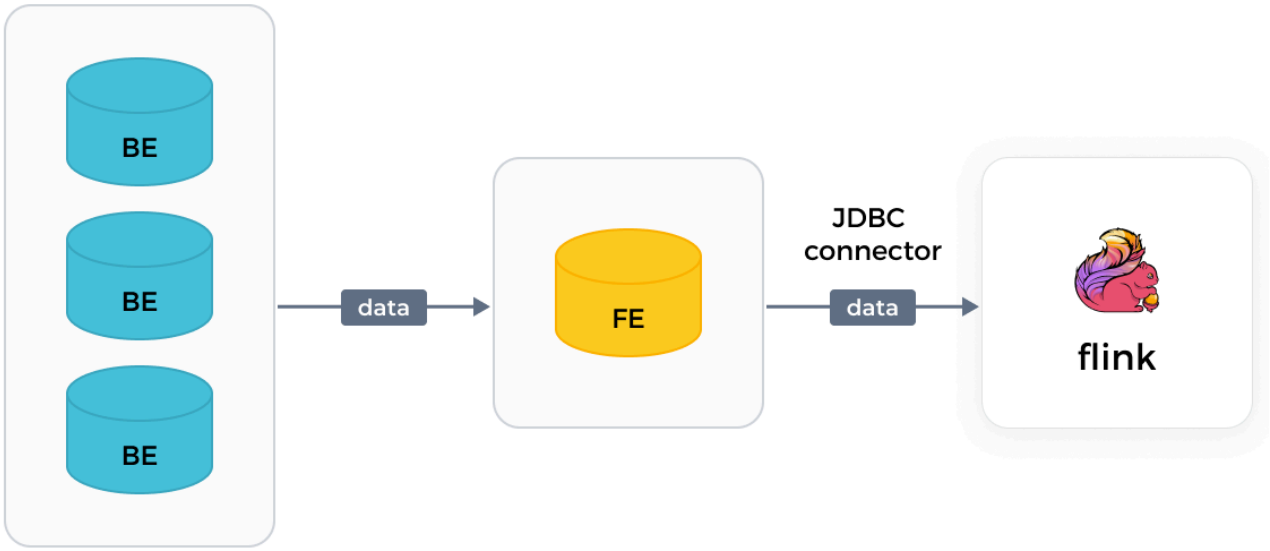
- Flink connector от Selena

  С Flink connector от Selena, Flink может сначала получить план запроса от ответственного FE, затем распределить полученный план запроса в качестве параметров всем задействованным BE и, наконец, получить данные, возвращаемые BE.



- JDBC connector от Flink

  С JDBC connector от Flink, Flink может читать данные только из отдельных FE, по одному за раз. Чтение данных происходит медленно.



4.3.2 Требования к версиям

Connector	Flink	Selena	Java	Scala
1.2.10	1.15,1.16,1.17,1.18,1.19	2.1 и новее	8	2.11,2.12
1.2.9	1.15,1.16,1.17,1.18	2.1 и новее	8	2.11,2.12
1.2.8	1.13,1.14,1.15,1.16,1.17	2.1 и новее	8	2.11,2.12
1.2.7	1.11,1.12,1.13,1.14,1.15	2.1 и новее	8	2.11,2.12

4.3.3 Предварительные требования

Flink должен быть развернут. Если Flink не развернут, следуйте этим шагам для его развертывания:

1. Установите Java 8 или Java 11 в вашей операционной системе, чтобы обеспечить правильную работу Flink. Вы можете использовать следующую команду для проверки версии вашей установки Java:

   ```
   java -version
   ```

   Например, если возвращается следующая информация, Java 8 установлена:

```
openjdk version "1.8.0_322"
OpenJDK Runtime Environment (Temurin)(build 1.8.0_322-b06)
OpenJDK 64-Bit Server VM (Temurin)(build 25.322-b06, mixed mode)
```

2. Загрузите и разархивируйте пакет Flink по вашему выбору.

> **ПРИМЕЧАНИЕ**
>
> Мы рекомендуем использовать Flink версии 1.14 или новее. Минимальная поддерживаемая версия Flink — 1.11.

```
# Загрузите пакет Flink.
wget https://dlcdn.apache.org/flink/flink-1.14.5/flink-1.14.5-bin-scala_2.11.tgz
# Разархивируйте пакет Flink.
tar -xzf flink-1.14.5-bin-scala_2.11.tgz
# Перейдите в директорию Flink.
cd flink-1.14.5
```

3. Запустите ваш Flink cluster.

```
# Запустите ваш Flink cluster.
./bin/start-cluster.sh

# Когда отображается следующая информация, ваш Flink cluster успешно запущен:
Starting cluster.
Starting standalonesession daemon on host.
Starting taskexecutor daemon on host.
```

Вы также можете развернуть Flink, следуя инструкциям, приведенным в документации Flink.

4.3.4 Прежде чем начать

Развертывание Flink connector

Следуйте этим шагам для развертывания Flink connector:

1. Выберите и загрузите JAR-пакет flink-connector-selena, соответствующий версии Flink, которую вы используете. Если требуется отладка кода, скомпилируйте пакет Flink connector в соответствии с вашими бизнес-требованиями.

> **ВАЖНО**
>
> Мы рекомендуем загрузить пакет Flink connector версии 1.2.x или новее, и чья соответствующая версия Flink имеет те же первые две цифры, что и используемая вами версия Flink. Например, если вы используете Flink версии 1.14.x, вы можете загрузить `flink-connector-selena-1.2.4_flink-1.14_x.yy.jar`.

2. Поместите загруженный или скомпилированный пакет Flink connector в директорию `lib` Flink.

3. Перезапустите ваш Flink cluster.

Конфигурация сети

Убедитесь, что машина, на которой находится Flink, может получить доступ к узлам FE Selena cluster через `http_port` (по умолчанию: `8030`) и `query_port` (по умолчанию: `9030`), а также к узлам BE через `be_port` (по умолчанию: `9060`).

4.3.5 Параметры

Общие параметры

Следующие параметры применяются к методам чтения как Flink SQL, так и Flink DataStream.

Параметр	Обязательный	Тип данных	Описание
connector	Да	STRING	Тип connector, который вы хотите использовать для чтения данных. Установите значение `selena`.
scan-url	Да	STRING	Адрес, используемый для подключения FE с веб-сервера. Формат: `<fe_host>:<fe_http_port>`. Порт по умолчанию — `8030`. Вы можете указать несколько адресов, которые должны быть разделены запятой (,). Пример: `192.168.xxx.xxx:8030,192.168.xxx.xxx:8030`.
jdbc-url	Да	STRING	Адрес, используемый для подключения клиента MySQL к FE. Формат: `jdbc:mysql://<fe_host>:<fe_query_port>`. Номер порта по умолчанию — `9030`.

Параметр	Обязательный	Тип данных	Описание
username	Да	STRING	Имя пользователя вашей учетной записи Selena cluster. Учетная запись должна иметь разрешения на чтение таблицы Selena, которую вы хотите прочитать. См. Привилегии пользователя.
password	Да	STRING	Пароль вашей учетной записи Selena cluster.
database-name	Да	STRING	Имя базы данных Selena, к которой принадлежит таблица Selena, которую вы хотите прочитать.
table-name	Да	STRING	Имя таблицы Selena, которую вы хотите прочитать.
scan.connect.timeout-ms	Нет	STRING	Максимальное время, после которого соединение от Flink connector к вашему Selena cluster истекает. Единица: миллисекунды. Значение по умолчанию: `1000`. Если время, затраченное на установление соединения, превышает этот лимит, задача чтения завершается неудачей.
scan.params.keep-alive-min	Нет	STRING	Максимальное время, в течение которого задача чтения остается активной. Время keep-alive проверяется на регулярной основе с использованием механизма опроса. Единица: минуты. Значение по умолчанию: `10`. Мы рекомендуем установить этот параметр на значение, большее или равное `5`.
scan.params.query-timeout-s	Нет	STRING	Максимальное время, после которого задача чтения истекает. Продолжительность timeout проверяется во время выполнения задачи. Единица: секунды. Значение по умолчанию: `600`. Если результат чтения не возвращается после истечения времени, задача чтения останавливается.
scan.params.mem-limit-byte	Нет	STRING	Максимальный объем памяти, разрешенный для каждого запроса на каждом BE. Единица: байты. Значение по умолчанию: `1073741824`, что равно 1 GB.
scan.max-retries	Нет	STRING	Максимальное количество раз, которое задача чтения может быть повторена при сбоях. Значение по умолчанию: `1`. Если количество повторных попыток задачи чтения превышает этот лимит, задача чтения возвращает ошибки.

Параметры для Flink DataStream

Следующие параметры применяются только к методу чтения Flink DataStream.

Параметр	Обязательный	Тип данных	Описание
scan.columns	Нет	STRING	Колонка, которую вы хотите прочитать. Вы можете указать несколько колонок, которые должны быть разделены запятой (,).
scan.filter	Нет	STRING	Условие фильтрации, на основе которого вы хотите фильтровать данные.

Предположим, что в Flink вы создаете таблицу, состоящую из трех колонок: `c1`, `c2`, `c3`. Чтобы прочитать строки, значения в колонке `c1` которых равны `100`, вы можете указать два условия фильтрации `"scan.columns, "c1"` и `"scan.filter, "c1 = 100"`.

4.3.6 Соответствие типов данных между Selena и Flink

Следующее соответствие типов данных действительно только для чтения данных Flink из Selena. Для соответствия типов данных, используемого при записи данных Flink в Selena, см. Непрерывная загрузка данных из Apache Flink®.

Selena	Flink
NULL	NULL
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
LARGEINT	STRING
FLOAT	FLOAT
DOUBLE	DOUBLE
DATE	DATE
DATETIME	TIMESTAMP
DECIMAL	DECIMAL
DECIMALV2	DECIMAL
DECIMAL32	DECIMAL
DECIMAL64	DECIMAL
DECIMAL128	DECIMAL
CHAR	CHAR
VARCHAR	STRING
JSON	STRING **ПРИМЕЧАНИЕ:** **Поддерживается с версии 1.2.10**
ARRAY	ARRAY **ПРИМЕЧАНИЕ:** **Поддерживается с версии 1.2.10, требуется Selena v1.5.2/v1.5.2 или новее.**
STRUCT	ROW **ПРИМЕЧАНИЕ:** **Поддерживается с версии 1.2.10, требуется Selena v1.5.2/v1.5.2 или новее.**

Selena	Flink
MAP	MAP **ПРИМЕЧАНИЕ:** **Поддерживается с версии 1.2.10, требуется Selena v1.5.2/v1.5.2 или новее.**

4.3.7 Примеры

Следующие примеры предполагают, что вы создали базу данных с именем `test` в вашем Selena cluster и у вас есть права пользователя `root`.

> **ПРИМЕЧАНИЕ**
>
> Если задача чтения завершается неудачей, вы должны пересоздать её.

Пример данных

1. Перейдите в базу данных `test` и создайте таблицу с именем `score_board`.

```
MySQL [test]> CREATE TABLE `score_board`
(
    `id` int(11) NOT NULL COMMENT "",
    `name` varchar(65533) NULL DEFAULT "" COMMENT "",
    `score` int(11) NOT NULL DEFAULT "0" COMMENT ""
)
ENGINE=OLAP
PRIMARY KEY(`id`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`id`)
PROPERTIES
(
    "replication_num" = "3"
);
```

2. Вставьте данные в таблицу `score_board`.

```
MySQL [test]> INSERT INTO score_board
VALUES
    (1, 'Bob', 21),
    (2, 'Stan', 21),
    (3, 'Sam', 22),
    (4, 'Tony', 22),
    (5, 'Alice', 22),
    (6, 'Lucy', 23),
    (7, 'Polly', 23),
    (8, 'Tom', 23),
    (9, 'Rose', 24),
    (10, 'Jerry', 24),
    (11, 'Jason', 24),
    (12, 'Lily', 25),
    (13, 'Stephen', 25),
    (14, 'David', 25),
    (15, 'Eddie', 26),
    (16, 'Kate', 27),
    (17, 'Cathy', 27),
    (18, 'Judy', 27),
    (19, 'Julia', 28),
    (20, 'Robert', 28),
    (21, 'Jack', 29);
```

3. Выполните запрос к таблице `score_board`.

```
MySQL [test]> SELECT * FROM score_board;
+------+---------+-------+
| id   | name    | score |
+------+---------+-------+
|    1 | Bob     |    21 |
|    2 | Stan    |    21 |
|    3 | Sam     |    22 |
|    4 | Tony    |    22 |
|    5 | Alice   |    22 |
|    6 | Lucy    |    23 |
|    7 | Polly   |    23 |
|    8 | Tom     |    23 |
|    9 | Rose    |    24 |
|   10 | Jerry   |    24 |
|   11 | Jason   |    24 |
|   12 | Lily    |    25 |
|   13 | Stephen |    25 |
|   14 | David   |    25 |
|   15 | Eddie   |    26 |
|   16 | Kate    |    27 |
|   17 | Cathy   |    27 |
|   18 | Judy    |    27 |
|   19 | Julia   |    28 |
|   20 | Robert  |    28 |
|   21 | Jack    |    29 |
+------+---------+-------+
21 rows in set (0.00 sec)
```

Чтение данных с помощью Flink SQL

1. В вашем Flink cluster создайте таблицу с именем `flink_test` на основе схемы исходной таблицы Selena (которая в этом примере `score_board`). В команде создания таблицы вы должны настроить свойства задачи чтения, включая информацию о Flink connector, исходной базе данных Selena и исходной таблице Selena.

```sql
CREATE TABLE flink_test
(
    `id` INT,
    `name` STRING,
    `score` INT
)
WITH
(
    'connector'='selena',
    'scan-url'='192.168.xxx.xxx:8030',
    'jdbc-url'='jdbc:mysql://192.168.xxx.xxx:9030',
    'username'='xxxxxx',
    'password'='xxxxxx',
    'database-name'='test',
    'table-name'='score_board'
);
```

2. Используйте SELECT для чтения данных из Selena.

```sql
SELECT id, name FROM flink_test WHERE score > 20;
```

При чтении данных с помощью Flink SQL обратите внимание на следующие моменты:

- Вы можете использовать только SQL-инструкции вида `SELECT ... FROM <table_name> WHERE ...` для чтения данных из Selena. Из всех агрегатных функций поддерживается только `count`.
- Поддерживается проталкивание предикатов (predicate pushdown). Например, если ваш запрос содержит условие фильтрации `char_1 <> 'A' and int_1 = -126`, условие фильтрации будет протолкнуто к Flink connector и преобразовано в инструкцию, которая может быть выполнена Selena, перед выполнением запроса. Вам не нужно выполнять дополнительные настройки.
- Инструкция LIMIT не поддерживается.
- Selena не поддерживает механизм checkpointing. В результате согласованность данных не может быть гарантирована при сбое задачи чтения.
- Порядок полей в созданной таблице должен быть таким же, как в таблице Selena.

Чтение данных с помощью Flink DataStream

1. Добавьте следующие зависимости в файл `pom.xml`:

```
<dependency>
    <groupId>com.selena</groupId>
    <artifactId>flink-connector-selena</artifactId>
    <!-- для Apache Flink® 1.15 -->
    <version>x.x.x_flink-1.15</version>
    <!-- для Apache Flink® 1.14 -->
    <version>x.x.x_flink-1.14_2.11</version>
    <version>x.x.x_flink-1.14_2.12</version>
    <!-- для Apache Flink® 1.13 -->
    <version>x.x.x_flink-1.13_2.11</version>
    <version>x.x.x_flink-1.13_2.12</version>
    <!-- для Apache Flink® 1.12 -->
    <version>x.x.x_flink-1.12_2.11</version>
    <version>x.x.x_flink-1.12_2.12</version>
    <!-- для Apache Flink® 1.11 -->
    <version>x.x.x_flink-1.11_2.11</version>
    <version>x.x.x_flink-1.11_2.12</version>
</dependency>
```

Вы должны заменить `x.x.x` в приведенном выше примере кода на последнюю версию Flink connector, которую вы используете. См. Информация о версии.

2. Вызовите Flink connector для чтения данных из Selena:

```
import com.selena.connector.flink.SelenaSource;
import com.selena.connector.flink.table.source.SelenaSourceOptions;
import
org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.table.api.DataTypes;
import org.apache.flink.table.api.TableSchema;

public class SelenaSourceApp {
        public static void main(String[] args) throws Exception {
            SelenaSourceOptions options = SelenaSourceOptions.builder()
                    .withProperty("scan-url", "192.168.xxx.xxx:8030")
                    .withProperty("jdbc-url",
"jdbc:mysql://192.168.xxx.xxx:9030")
                    .withProperty("username", "root")
                    .withProperty("password", "")
                    .withProperty("table-name", "score_board")
                    .withProperty("database-name", "test")
                    .build();
            TableSchema tableSchema = TableSchema.builder()
                    .field("id", DataTypes.INT())
                    .field("name", DataTypes.STRING())
                    .field("score", DataTypes.INT())
                    .build();
            StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
            env.addSource(SelenaSource.source(tableSchema,
options)).setParallelism(5).print();
            env.execute("Selena flink source");
        }

    }
```

4.3.8 Часто задаваемые вопросы

Я получил ошибку "Failed to get next from be" при экспорте данных с помощью Flink Connector. Что мне делать?

Вы можете установить конфигурацию BE `scan_context_gc_interval_min` (По умолчанию: 5, Единица: Минуты) на большее значение, чтобы увеличить временной интервал, с которым очищается Scan Context.

4.3.9 Что дальше

После того как Flink успешно прочитает данные из Selena, вы можете использовать Flink WebUI для мониторинга задачи чтения. Например, вы можете просмотреть метрику `totalScannedRows` на странице **Metrics** WebUI, чтобы получить количество успешно прочитанных строк. Вы также можете использовать Flink SQL для выполнения вычислений, таких как joins, на прочитанных данных.

4.4 Взаимодействие с Selena через Arrow Flight SQL

Начиная с версии v1.5.2, Selena поддерживает подключения через протокол Apache Arrow Flight SQL.

4.4.1 Обзор

С протоколом Arrow Flight SQL вы можете выполнять обычные операторы DDL, DML, DQL и использовать код Python или Java для чтения крупномасштабных данных через драйвер Arrow Flight SQL ADBC или JDBC.

Это решение устанавливает полностью колоночный конвейер передачи данных от колоночного движка выполнения Selena к клиенту, устраняя частые преобразования строк в столбцы и накладные расходы на сериализацию, обычно наблюдаемые в традиционных интерфейсах JDBC и ODBC. Это позволяет Selena передавать данные с нулевым копированием, низкой задержкой и высокой пропускной способностью.

Сценарии использования

Интеграция Arrow Flight SQL делает Selena особенно подходящей для:

- Рабочих процессов data science, где инструменты, такие как Pandas и Apache Arrow, ожидают колоночные данные.
- Аналитики озер данных, требующей высокой пропускной способности и низкой задержки доступа к массивным наборам данных.
- Машинного обучения, где критически важны быстрая итерация и скорость обработки.
- Платформ аналитики в реальном времени, которые должны доставлять данные с минимальными задержками.

С Arrow Flight SQL вы можете извлечь выгоду из:

- Сквозной передачи колоночных данных, устраняющей дорогостоящие преобразования между колоночными и строковыми форматами.
- Перемещения данных с нулевым копированием, снижающего накладные расходы на CPU и память.
- Низкой задержки и чрезвычайно высокой пропускной способности, ускоряющей анализ и отзывчивость.

Технический подход

Традиционно Selena организует результаты запросов в колоночной структуре Block внутри системы. Однако при использовании JDBC, ODBC или MySQL протоколов данные должны быть:

1. Сериализованы в байты на основе строк на сервере.
2. Переданы по сети.
3. Десериализованы обратно в целевую структуру (часто требуя повторного преобразования в колоночные форматы).

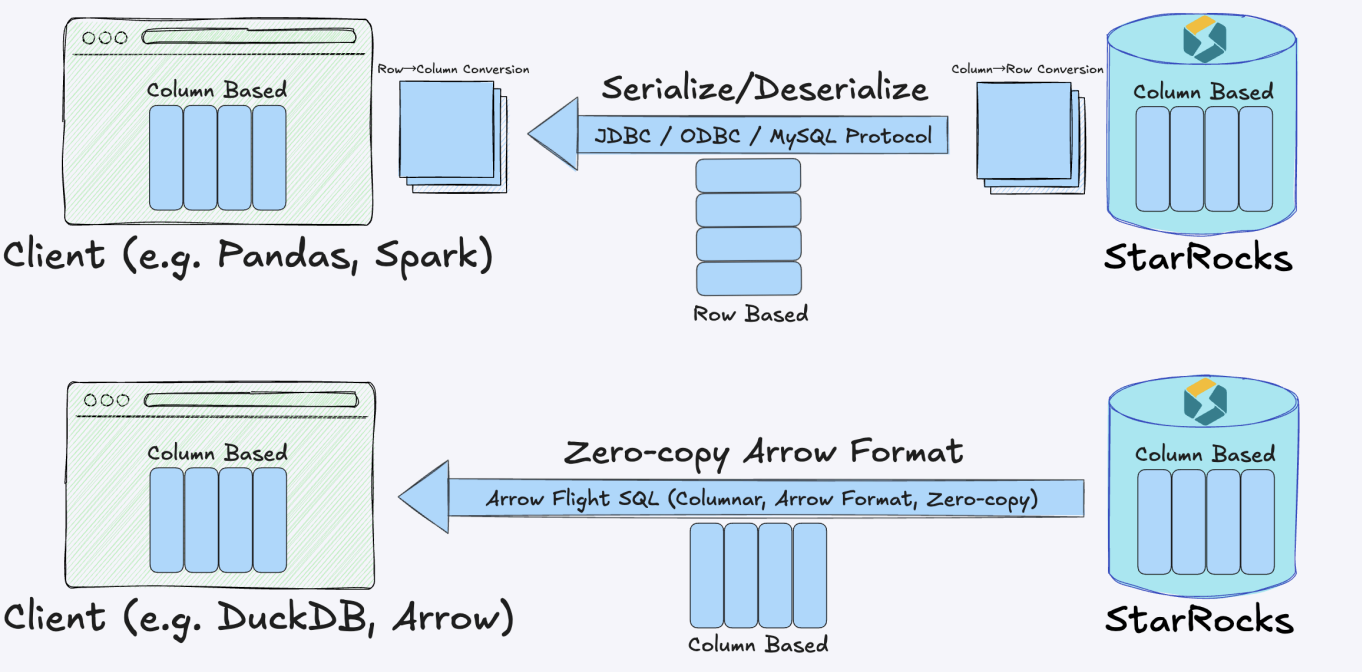
Этот трехэтапный процесс создает:

- Высокие накладные расходы на сериализацию/десериализацию.
- Сложные преобразования данных.
- Задержку, которая растет с объемом данных.

Интеграция с Arrow Flight SQL решает эти проблемы путем:

- Сохранения колоночного формата от начала до конца, от движка выполнения Selena непосредственно к клиенту.
- Использования колоночного представления Apache Arrow в памяти, которое оптимизировано для аналитических рабочих нагрузок.
- Использования протокола Arrow Flight для высокоскоростной передачи, обеспечивающей эффективную потоковую передачу без промежуточных преобразований.

© ООО «Платформа Данных Селена», 2026



Эта конструкция обеспечивает настоящую передачу с нулевым копированием, которая одновременно быстрее и более эффективна по ресурсам, чем традиционные методы.

Кроме того, Selena предлагает универсальный JDBC драйвер для Arrow Flight SQL, поэтому приложения могут использовать этот высокопроизводительный путь передачи, не жертвуя совместимостью с JDBC или взаимодействием с другими системами, поддерживающими Arrow Flight.

Для развертываний, где узлы BE недоступны напрямую для клиентов (например, в частных сетях или кластерах Kubernetes), Selena предоставляет функцию прокси Arrow Flight. При включении FE может выступать в качестве прокси для маршрутизации данных Arrow от узлов BE к клиентам, сохраняя преимущества колоночной передачи при учете ограничений сетевой топологии. Этот режим прокси вносит небольшие накладные расходы на производительность, но обеспечивает доступ к Arrow Flight SQL в средах, где прямое подключение к BE недоступно.

Сравнение производительности

Комплексные тесты демонстрируют значительные улучшения скорости извлечения данных. Для различных типов данных (целые числа, числа с плавающей точкой, строки, логические значения и смешанные столбцы) Arrow Flight SQL постоянно превосходил традиционные интерфейсы PyMySQL и Pandas `read_sql`. Ключевые результаты включают:

- Для чтения 10 миллионов строк целых чисел время выполнения снизилось с ~35 секунд до 0,4 секунды (~85× быстрее).
- Для таблицы со смешанными столбцами улучшение производительности достигло 160× ускорения.
- Даже в менее сложных запросах (например, одиночные строковые столбцы) прирост производительности превысил 12×.

В среднем Arrow Flight SQL достиг:

- В 20-160 раз более быстрого времени передачи, в зависимости от сложности запроса и типа данных.
- Явного снижения использования CPU и памяти благодаря устранению избыточных этапов сериализации.

Эти прироста производительности напрямую транслируются в более быстрые дашборды, более отзывчивые рабочие процессы data science и возможность анализировать гораздо более крупные наборы данных в реальном времени.

4.4.2 Использование

Следуйте этим шагам для подключения и взаимодействия с Selena, используя Python ADBC Driver через протокол Arrow Flight SQL. См. Приложение для полного примера кода.

> **ПРИМЕЧАНИЕ**
>
> Python 3.9 или новее является предварительным требованием.

Шаг 1. Установка библиотек

Используйте `pip` для установки `adbc_driver_manager` и `adbc_driver_flightsql` из PyPI:

```
pip install adbc_driver_manager
pip install adbc_driver_flightsql
```

Импортируйте следующие модули или библиотеки в ваш код:

- Обязательные библиотеки:

```
import adbc_driver_manager
import adbc_driver_flightsql.dbapi as flight_sql
```

- Опциональные модули для лучшей удобства использования и отладки:

```
import pandas as pd       # Опционально: для лучшего отображения
результатов с использованием DataFrame
import traceback          # Опционально: для детального отслеживания ошибок
во время выполнения SQL
import time               # Опционально: для измерения времени выполнения
SQL
```

Шаг 2. Подключение к Selena

ПРИМЕЧАНИЕ

- Если вы хотите запустить сервис FE с помощью командной строки, вы можете использовать один из следующих способов:
  - Указать переменную окружения `JAVA_TOOL_OPTIONS`.

    ```
    export JAVA_TOOL_OPTIONS="--add-opens=java.base/java.nio=org.apache.arrow.memory.core,ALL-UNNAMED"
    ```

  - Указать параметр конфигурации FE `JAVA_OPTS` в **fe.conf**. Таким образом, вы можете добавить другие значения `JAVA_OPTS`.

    ```
    JAVA_OPTS="--add-opens=java.base/java.nio=org.apache.arrow.memory.core,ALL-UNNAMED ..."
    ```

- Если вы хотите запустить сервис в IntelliJ IDEA, вы должны добавить следующую опцию в `Build and run` в `Run/Debug Configurations`:

  ```
  --add-opens=java.base/java.nio=org.apache.arrow.memory.core,ALL-UNNAMED
  ```

Настройка Selena

Перед подключением к Selena через Arrow Flight SQL необходимо сначала настроить узлы FE и BE, чтобы обеспечить включение сервиса Arrow Flight SQL и прослушивание на указанных портах.

В файле конфигурации FE **fe.conf** и файле конфигурации BE **be.conf** установите `arrow_flight_port` на доступный порт. После изменения файлов конфигурации перезапустите сервисы FE и BE, чтобы изменения вступили в силу.

> **ПРИМЕЧАНИЕ**
>
> Вы должны установить разные `arrow_flight_port` для FE и BE.

Пример:

```
// fe.conf
arrow_flight_port = 9408
// be.conf
arrow_flight_port = 9419
```

Настройка Arrow Flight Proxy (Опционально)

Если ваши узлы BE недоступны напрямую из клиентских приложений (например, при развертывании в частных сетях или средах Kubernetes), вы можете включить функцию прокси Arrow Flight на FE для маршрутизации данных от узлов BE через FE.

Функция прокси управляется двумя глобальными переменными:

- `arrow_flight_proxy_enabled`: Управляет включением режима прокси. По умолчанию `true`. При включении возникают небольшие накладные расходы на производительность.
- `arrow_flight_proxy`: Указывает имя хоста прокси. Если пусто (по умолчанию), текущий узел FE выступает в качестве прокси. Вы можете установить это значение на конкретное имя хоста, если используете другую конечную точку прокси.

Чтобы настроить эти переменные глобально для всех сессий:

```
-- Включить или отключить режим прокси (включен по умолчанию)
SET GLOBAL arrow_flight_proxy_enabled = true;

-- Установить конкретное имя хоста прокси (опционально, по умолчанию
текущий FE)
SET GLOBAL arrow_flight_proxy = 'your-proxy-hostname:Port';
```

> **ПРИМЕЧАНИЕ**
>
> - Функция прокси включена по умолчанию, что может привести к снижению пропускной способности на 8-10% по сравнению с прямыми подключениями к BE. Если ваши клиенты имеют прямой сетевой доступ к узлам BE, вы можете отключить прокси для достижения оптимальной производительности: `SET GLOBAL arrow_flight_proxy_enabled = false;`
> - Когда `arrow_flight_proxy` пуст, тикеты будут автоматически маршрутизироваться через узел FE, к которому клиент изначально подключился.

Установка подключения

На стороне клиента создайте клиент Arrow Flight SQL, используя следующую информацию:

- Адрес хоста Selena FE
- Порт, который Arrow Flight использует для прослушивания на Selena FE
- Имя пользователя и пароль пользователя Selena с необходимыми привилегиями

Пример:

```
FE_HOST = "127.0.0.1"
FE_PORT = 9408

conn = flight_sql.connect(
    uri=f"grpc://{FE_HOST}:{FE_PORT}",
    db_kwargs={
        adbc_driver_manager.DatabaseOptions.USERNAME.value: "root",
        adbc_driver_manager.DatabaseOptions.PASSWORD.value: "",
    }
)
cursor = conn.cursor()
```

После установления соединения вы можете взаимодействовать с Selena, выполняя SQL операторы через возвращенный Cursor.

Шаг 3. (Опционально) Предварительное определение утилитных функций

Эти функции используются для форматирования вывода, стандартизации формата и упрощения отладки. Вы можете опционально определить их в своем коде для тестирования.

```
#
================================================================================
# Утилитные функции для лучшего форматирования вывода и выполнения SQL
#
================================================================================

# Вывести заголовок раздела
def print_header(title: str):
    """
    Вывести заголовок раздела для лучшей читаемости.
    """
    print("\n" + "=" * 80)
    print(f"🔴 {title}")
    print("=" * 80)

# Вывести выполняемый SQL оператор
def print_sql(sql: str):
    """
    Вывести SQL оператор перед выполнением.
    """
    print(f"\n🔵 SQL:\n{sql.strip()}")

# Вывести результат DataFrame
def print_result(df: pd.DataFrame):
    """
    Вывести результат DataFrame в читаемом формате.
    """
    if df.empty:
        print("\n🟢 Result: (no rows returned)\n")
    else:
        print("\n🟢 Result:\n")
        print(df.to_string(index=False))

# Вывести трассировку ошибки
def print_error(e: Exception):
    """
    Вывести трассировку ошибки, если выполнение SQL не удалось.
    """
    print("\n⚠ Error occurred:")
    traceback.print_exc()

# Выполнить SQL оператор и вывести результат
def execute(sql: str):
    """
    Выполнить SQL оператор и вывести результат и время выполнения.
```

```
    """
    print_sql(sql)
    try:
        start = time.time()  # Опционально: время начала для измерения
времени выполнения
        cursor.execute(sql)
        result = cursor.fetchallarrow()  # Arrow Table
        df = result.to_pandas()  # Опционально: преобразовать в DataFrame
для лучшего отображения
        print_result(df)
        print(f"\n⏱  Execution time: {time.time() - start:.3f} seconds")
    except Exception as e:
        print_error(e)
```

Шаг 4. Взаимодействие с Selena

Этот раздел проведет вас через некоторые базовые операции, такие как создание таблицы, загрузка данных, проверка метаданных таблицы, установка переменных и выполнение запросов.

> **ПРИМЕЧАНИЕ**
>
> Примеры вывода, перечисленные ниже, реализованы на основе опциональных модулей и утилитных функций, описанных на предыдущих шагах.

1. Создайте базу данных и таблицу, куда будут загружены данные, и проверьте схему таблицы.

```
# Шаг 1: Удалить и создать базу данных
print_header("Step 1: Drop and Create Database")
execute("DROP DATABASE IF EXISTS sr_arrow_flight_sql FORCE;")
execute("SHOW DATABASES;")
execute("CREATE DATABASE sr_arrow_flight_sql;")
execute("SHOW DATABASES;")
execute("USE sr_arrow_flight_sql;")

# Шаг 2: Создать таблицу
print_header("Step 2: Create Table")
execute("""
CREATE TABLE sr_arrow_flight_sql_test
(
    k0 INT,
    k1 DOUBLE,
    k2 VARCHAR(32) NULL DEFAULT "" COMMENT "",
    k3 DECIMAL(27,9) DEFAULT "0",
    k4 BIGINT NULL DEFAULT '10',
    k5 DATE
)
DISTRIBUTED BY HASH(k5) BUCKETS 5
PROPERTIES("replication_num" = "1");
""")
execute("SHOW CREATE TABLE sr_arrow_flight_sql_test;")
```

Пример вывода:

```
================================================================================
🔘 Step 1: Drop and Create Database
================================================================================

🔘 SQL:
DROP DATABASE IF EXISTS sr_arrow_flight_sql FORCE;
/Users/selena/test/venv/lib/python3.9/site-
packages/adbc_driver_manager/dbapi.py:307: Warning: Cannot disable autocom
conn will not be DB-API 2.0 compliant
  warnings.warn(

🔘 Result:

StatusResult
           0

⏱️  Execution time: 0.025 seconds

🔘 SQL:
SHOW DATABASES;

🔘 Result:

          Database
      _statistics_
              hits
information_schema
               sys

⏱️  Execution time: 0.014 seconds

🔘 SQL:
CREATE DATABASE sr_arrow_flight_sql;

🔘 Result:

StatusResult
           0

⏱️  Execution time: 0.012 seconds

🔘 SQL:
SHOW DATABASES;

🔘 Result:
```

```
           Database
       _statistics_
               hits
 information_schema
sr_arrow_flight_sql
                sys

⏱  Execution time: 0.005 seconds

🔵 SQL:
USE sr_arrow_flight_sql;

🟢 Result:

StatusResult
           0

⏱  Execution time: 0.006 seconds

================================================================================
🟢 Step 2: Create Table
================================================================================

🔵 SQL:
CREATE TABLE sr_arrow_flight_sql_test
(
    k0 INT,
    k1 DOUBLE,
    k2 VARCHAR(32) NULL DEFAULT "" COMMENT "",
    k3 DECIMAL(27,9) DEFAULT "0",
    k4 BIGINT NULL DEFAULT '10',
    k5 DATE
)
DISTRIBUTED BY HASH(k5) BUCKETS 5
PROPERTIES("replication_num" = "1");

🟢 Result:

StatusResult
           0

⏱  Execution time: 0.021 seconds

🔵 SQL:
SHOW CREATE TABLE sr_arrow_flight_sql_test;
```

```
🔘 Result:

                    Table
Create Table
sr_arrow_flight_sql_test CREATE TABLE `sr_arrow_flight_sql_test` (\n  `k0
int(11) NULL COMMENT "",\n  `k1` double NULL COMMENT "",\n  `k2` varchar(
NULL DEFAULT "" COMMENT "",\n  `k3` decimal(27, 9) NULL DEFAULT "0" COMME
"",\n  `k4` bigint(20) NULL DEFAULT "10" COMMENT "",\n  `k5` date NULL CO
""\n) ENGINE=OLAP \nDUPLICATE KEY(`k0`)\nDISTRIBUTED BY HASH(`k5`) BUCKET
\nPROPERTIES (\n"compression" = "LZ4",\n"fast_schema_evolution" =
"true",\n"replicated_storage" = "true",\n"replication_num" = "1"\n);

⏱  Execution time: 0.005 seconds
```

2. Вставьте данные, выполните несколько запросов и установите переменные.

```
# Шаг 3: Вставить данные
print_header("Step 3: Insert Data")
execute("""
INSERT INTO sr_arrow_flight_sql_test VALUES
    (0, 0.1, "ID", 0.0001, 1111111111, '2025-04-21'),
    (1, 0.20, "ID_1", 1.00000001, 0, '2025-04-21'),
    (2, 3.4, "ID_1", 3.1, 123456, '2025-04-22'),
    (3, 4, "ID", 4, 4, '2025-04-22'),
    (4, 122345.54321, "ID", 122345.54321, 5, '2025-04-22');
""")

# Шаг 4: Запросить данные
print_header("Step 4: Query Data")
execute("SELECT * FROM sr_arrow_flight_sql_test ORDER BY k0;")

# Шаг 5: Переменные сессии
print_header("Step 5: Session Variables")
execute("SHOW VARIABLES LIKE '%query_mem_limit%';")
execute("SET query_mem_limit = 2147483648;")
execute("SHOW VARIABLES LIKE '%query_mem_limit%';")
execute("SHOW VARIABLES LIKE '%arrow_flight_proxy%';")
execute("SET arrow_flight_proxy_enabled = true;")
execute("SET arrow_flight_proxy = 'fe-proxy.example.com';")
execute("SHOW VARIABLES LIKE '%arrow_flight_proxy%';")

# Шаг 6: Агрегационный запрос
print_header("Step 6: Aggregation Query")
execute("""
SELECT k5, SUM(k1) AS total_k1, COUNT(1) AS row_count, AVG(k3) AS
avg_k3
FROM sr_arrow_flight_sql_test
GROUP BY k5
ORDER BY k5;
""")
```

Пример вывода:

```
================================================================================
🔘 Step 3: Insert Data
================================================================================

🔘 SQL:
INSERT INTO sr_arrow_flight_sql_test VALUES
    (0, 0.1, "ID", 0.0001, 1111111111, '2025-04-21'),
    (1, 0.20, "ID_1", 1.00000001, 0, '2025-04-21'),
    (2, 3.4, "ID_1", 3.1, 123456, '2025-04-22'),
    (3, 4, "ID", 4, 4, '2025-04-22'),
    (4, 122345.54321, "ID", 122345.54321, 5, '2025-04-22');

🔘 Result:

StatusResult
           0

⏱  Execution time: 0.149 seconds

================================================================================
🔘 Step 4: Query Data
================================================================================

🔘 SQL:
SELECT * FROM sr_arrow_flight_sql_test ORDER BY k0;

🔘 Result:

0      0.10000   ID      0.000100000 1111111111 2025-04-21
1      0.20000 ID_1      1.000000010          0 2025-04-21
2      3.40000 ID_1      3.100000000     123456 2025-04-22
3      4.00000   ID      4.000000000          4 2025-04-22
4 122345.54321   ID 122345.543210000          5 2025-04-22

⏱  Execution time: 0.019 seconds

================================================================================
🔘 Step 5: Session Variables
================================================================================

🔘 SQL:
SHOW VARIABLES LIKE '%query_mem_limit%';

🔘 Result:
```

```
  Variable_name Value
query_mem_limit     0

⏱  Execution time: 0.005 seconds

🔘 SQL:
SET query_mem_limit = 2147483648;

🔘 Result:

StatusResult
           0

⏱  Execution time: 0.007 seconds

🔘 SQL:
SHOW VARIABLES LIKE '%query_mem_limit%';

🔘 Result:

  Variable_name        Value
  query_mem_limit 2147483648

⏱  Execution time: 0.005 seconds

🔘 SQL:
SHOW VARIABLES LIKE '%arrow_flight_proxy%';

🔘 Result:

  Variable_name Value
  arrow_flight_proxy
  arrow_flight_proxy_enabled  true

⏱  Execution time: 0.006 seconds

🔘 SQL:
SET arrow_flight_proxy_enabled = true;

🔘 Result:

StatusResult
           0

⏱  Execution time: 0.008 seconds
```

```
🔘 SQL:
SET arrow_flight_proxy = 'fe-proxy.example.com';

🔘 Result:

StatusResult
           0

⏱️  Execution time: 0.007 seconds

🔘 SQL:
SHOW VARIABLES LIKE '%arrow_flight_proxy%';

🔘 Result:

      Variable_name    Value
      arrow_flight_proxy   fe-proxy.example.com
      arrow_flight_proxy_enabled   true

 ⏱️  Execution time: 0.006 seconds

================================================================================
🔘 Step 6: Aggregation Query
================================================================================

🔘 SQL:
SELECT k5, SUM(k1) AS total_k1, COUNT(1) AS row_count, AVG(k3) AS avg_k3
FROM sr_arrow_flight_sql_test
GROUP BY k5
ORDER BY k5;

🔘 Result:

2025-04-21      0.30000 2     0.500050005000
2025-04-22 122352.94321 3 40784.214403333333

⏱️  Execution time: 0.014 second
```

Шаг 5. Закрытие соединения

Включите следующий раздел в ваш код для закрытия соединения.

```
# Шаг 7: Закрыть
print_header("Step 7: Close Connection")
cursor.close()
conn.close()
print("▫ Test completed successfully.")
```

Пример вывода:

```
================================================================================
🔘 Step 7: Close Connection
================================================================================
▫ Test completed successfully.

Process finished with exit code 0
```

4.4.3 Примеры использования крупномасштабной передачи данных

Python

После подключения к Selena (с поддержкой Arrow Flight SQL) через ADBC Driver в Python вы можете использовать различные API ADBC для загрузки набора данных Clickbench из Selena в Python.

Пример кода:

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import adbc_driver_manager
import adbc_driver_flightsql.dbapi as flight_sql
from datetime import datetime

# ----------------------------------------
# Настройки подключения Selena Flight SQL
# ----------------------------------------
# Замените URI и учетные данные по мере необходимости
my_uri = "grpc://127.0.0.1:9408"  # Порт по умолчанию Flight SQL для Selena
my_db_kwargs = {
    adbc_driver_manager.DatabaseOptions.USERNAME.value: "root",
    adbc_driver_manager.DatabaseOptions.PASSWORD.value: "",
}

# ----------------------------------------
# SQL запрос (ClickBench: таблица hits)
# ----------------------------------------
# Замените на актуальную таблицу и набор данных по мере необходимости
sql = "SELECT * FROM clickbench.hits LIMIT 1000000;"  # Прочитать 1 миллион
строк

# ----------------------------------------
# Метод 1: fetchallarrow + to_pandas
# ----------------------------------------
def test_fetchallarrow():
    conn = flight_sql.connect(uri=my_uri, db_kwargs=my_db_kwargs)
    cursor = conn.cursor()
    start = datetime.now()
    cursor.execute(sql)
    arrow_table = cursor.fetchallarrow()
    df = arrow_table.to_pandas()
    duration = datetime.now() - start

    print("\n[Method 1] fetchallarrow + to_pandas")
    print(f"Time taken: {duration}, Arrow table size: {arrow_table.nbytes /
1024 / 1024:.2f} MB, Rows: {len(df)}")
    print(df.info(memory_usage='deep'))

# ----------------------------------------
# Метод 2: fetch_df (рекомендуется)
# ----------------------------------------
def test_fetch_df():
```

```
    conn = flight_sql.connect(uri=my_uri, db_kwargs=my_db_kwargs)
    cursor = conn.cursor()
    start = datetime.now()
    cursor.execute(sql)
    df = cursor.fetch_df()
    duration = datetime.now() - start

    print("\n[Method 2] fetch_df (recommended)")
    print(f"Time taken: {duration}, Rows: {len(df)}")
    print(df.info(memory_usage='deep'))

# ----------------------------------------
# Метод 3: adbc_execute_partitions (для параллельного чтения)
# ----------------------------------------
def test_execute_partitions():
    conn = flight_sql.connect(uri=my_uri, db_kwargs=my_db_kwargs)
    cursor = conn.cursor()
    start = datetime.now()
    partitions, schema = cursor.adbc_execute_partitions(sql)

    # Прочитать первый partition (для демонстрации)
    cursor.adbc_read_partition(partitions[0])
    arrow_table = cursor.fetchallarrow()
    df = arrow_table.to_pandas()
    duration = datetime.now() - start

    print("\n[Method 3] adbc_execute_partitions (parallel read)")
    print(f"Time taken: {duration}, Partitions: {len(partitions)}, Rows:
{len(df)}")
    print(df.info(memory_usage='deep'))

# ----------------------------------------
# Запустить все тесты
# ----------------------------------------
if __name__ == "__main__":
    test_fetchallarrow()
    test_fetch_df()
    test_execute_partitions()
```

Результаты показывают, что загрузка 1 миллиона строк набора данных Clickbench (105 столбцов, 780 МБ) из Selena заняла всего 3 секунды.

```
[Method 1] fetchallarrow + to_pandas
Time taken: 0:00:03.219575, Arrow table size: 717.42 MB, Rows: 1000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Columns: 105 entries, CounterID to CLID
dtypes: int16(48), int32(19), int64(6), object(32)
memory usage: 2.4 GB

[Method 2] fetch_df (recommended)
Time taken: 0:00:02.358840, Rows: 1000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Columns: 105 entries, CounterID to CLID
dtypes: int16(48), int32(19), int64(6), object(32)
memory usage: 2.4 GB

[Method 3] adbc_execute_partitions (parallel read)
Time taken: 0:00:02.231144, Partitions: 1, Rows: 1000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Columns: 105 entries, CounterID to CLID
dtypes: int16(48), int32(19), int64(6), object(32)
memory usage: 2.4 GB
```

Arrow Flight SQL JDBC Driver

Протокол Arrow Flight SQL предоставляет JDBC драйвер с открытым исходным кодом, совместимый со стандартным интерфейсом JDBC. Вы можете легко интегрировать его в различные инструменты BI (такие как Tableau, Power BI, DBeaver и т.д.) для доступа к базе данных Selena, точно так же, как с традиционным JDBC драйвером. Значительным преимуществом этого драйвера является его поддержка высокоскоростной передачи данных на основе Apache Arrow, что значительно повышает эффективность запросов и передачи данных. Использование почти идентично традиционному MySQL JDBC драйверу. Вам нужно только заменить `jdbc:mysql` на `jdbc:arrow-flight-sql` в URL подключения для бесшовного переключения. Результаты запроса по-прежнему возвращаются в стандартном формате `ResultSet`, обеспечивая совместимость с существующей логикой обработки JDBC.

ПРИМЕЧАНИЕ

Обратите внимание, что если вы используете Java 9 или новее, вы должны добавить `--add-opens=java.base/java.nio=ALL-UNNAMED` в ваш Java код, чтобы открыть внутреннюю структуру JDK. В противном случае вы можете столкнуться с определенными ошибками.

- Если вы хотите запустить сервис FE с помощью командной строки, вы можете использовать один из следующих способов:
  - Указать переменную окружения `JAVA_TOOL_OPTIONS`.

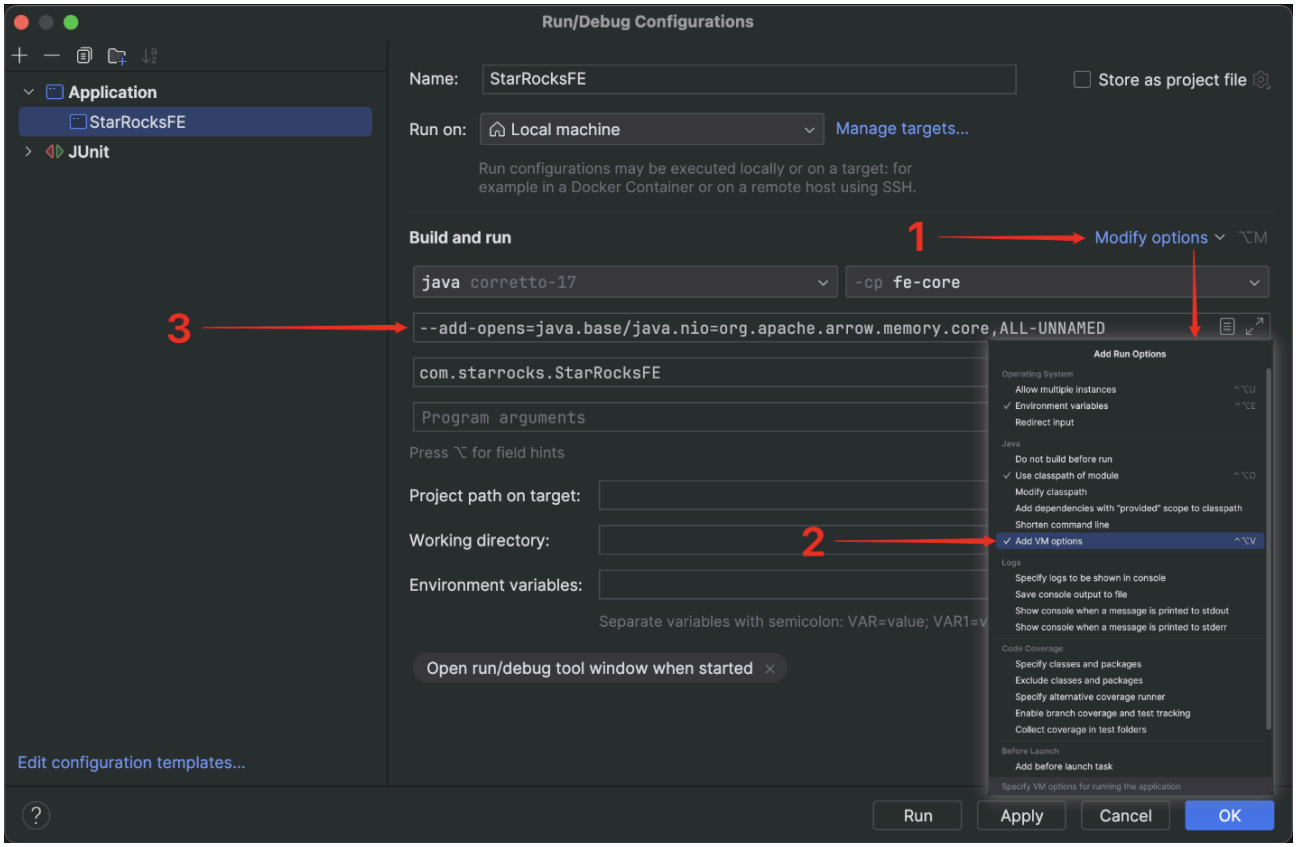
    ```
    export JAVA_TOOL_OPTIONS="--add-opens=java.base/java.nio=org.apache.arrow.memory.core,ALL-UNNAMED"
    ```

  - Указать параметр конфигурации FE `JAVA_OPTS` в **fe.conf**. Таким образом, вы можете добавить другие значения `JAVA_OPTS`.

    ```
    JAVA_OPTS="--add-opens=java.base/java.nio=org.apache.arrow.memory.core,ALL-UNNAMED ..."
    ```

- Если вы хотите отлаживать в IntelliJ IDEA, вы должны добавить следующую опцию в `Build and run` в `Run/Debug Configurations`:

  ```
  --add-opens=java.base/java.nio=org.apache.arrow.memory.core,ALL-UNNAMED
  ```



Нажмите здесь, чтобы просмотреть зависимости POM

Пример кода:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class ArrowFlightSqlIntegrationTest {

    private static final String JDBC_URL = "jdbc:arrow-flight-
sql://127.0.0.1:9408"
            + "?useEncryption=false"
            + "&useServerPrepStmts=false"
            + "&useSSL=false"
            + "&useArrowFlightSql=true";

    private static final String USER = "root";
    private static final String PASSWORD = "";

    private static int testCaseNum = 1;

    public static void main(String[] args) {
        try {
            // Загрузить Arrow Flight SQL JDBC driver

Class.forName("org.apache.arrow.driver.jdbc.ArrowFlightJdbcDriver");

            try (Connection conn = DriverManager.getConnection(JDBC_URL,
USER, PASSWORD);
                    Statement stmt = conn.createStatement()) {

                testUpdate(stmt, "DROP DATABASE IF EXISTS
sr_arrow_flight_sql FORCE;");
                testQuery(stmt, "SHOW PROCESSLIST;");
                testUpdate(stmt, "CREATE DATABASE sr_arrow_flight_sql;");
                testQuery(stmt, "SHOW DATABASES;");
                testUpdate(stmt, "USE sr_arrow_flight_sql;");
                testUpdate(stmt, "CREATE TABLE sr_table_test (id INT, name
STRING) ENGINE=OLAP PRIMARY KEY (id) " +
                        "DISTRIBUTED BY HASH(id) BUCKETS 1 " +
                        "PROPERTIES ('replication_num' = '1');");
                testUpdate(stmt, "INSERT INTO sr_table_test VALUES (1,
'Alice'), (2, 'Bob');");
                testQuery(stmt, "SELECT * FROM
sr_arrow_flight_sql.sr_table_test;");
            }
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }

    }

    /**
     * Выполняет запрос и выводит результат в консоль.
     */
    private static void testQuery(Statement stmt, String sql) throws
Exception {
        System.out.println("Test Case: " + testCaseNum);
        System.out.println("▶ Executing query: " + sql);
        ResultSet rs = stmt.executeQuery(sql);
        try {
            System.out.println("Result:");
            int columnCount = rs.getMetaData().getColumnCount();
            while (rs.next()) {
                for (int i = 1; i <= columnCount; i++) {
                    System.out.print(rs.getString(i) + "\t");
                }
                System.out.println();
            }
        } finally {
            rs.close();
        }
        testCaseNum++;
        System.out.println();
    }

    /**
     * Выполняет обновление (DDL или DML) и выводит результат в консоль.
     */
    private static void testUpdate(Statement stmt, String sql) throws
Exception {
        System.out.println("Test Case: " + testCaseNum);
        System.out.println("▶ Executing update: " + sql);
        stmt.executeUpdate(sql);
        System.out.println("Result: ▫ Success");
        testCaseNum++;
        System.out.println();
    }
}
```

Результаты выполнения:

```
Test Case: 1
▶ Executing update: DROP DATABASE IF EXISTS sr_arrow_flight_sql FORCE;
Result: ▫ Success

Test Case: 2
▶ Executing query: SHOW PROCESSLIST;
Result:
192.168.124.48_9010_1751449846872       16777217        root
Query   2025-07-02 18:46:49     0       OK      SHOW PROCESSLIST;
false   default_warehouse

Test Case: 3
▶ Executing update: CREATE DATABASE sr_arrow_flight_sql;
Result: ▫ Success

Test Case: 4
▶ Executing query: SHOW DATABASES;
Result:
_statistics_
information_schema
sr_arrow_flight_sql
sys

Test Case: 5
▶ Executing update: USE sr_arrow_flight_sql;
Result: ▫ Success

Test Case: 6
▶ Executing update: CREATE TABLE sr_table_test (id INT, name STRING)
ENGINE=OLAP PRIMARY KEY (id) DISTRIBUTED BY HASH(id) BUCKETS 1 PROPERTIES
('replication_num' = '1');
Result: ▫ Success

Test Case: 7
▶ Executing update: INSERT INTO sr_table_test VALUES (1, 'Alice'), (2,
'Bob');
Result: ▫ Success

Test Case: 8
▶ Executing query: SELECT * FROM sr_arrow_flight_sql.sr_table_test;
Result:
1       Alice
2       Bob
```

Java ADBC Driver

Протокол Arrow Flight SQL предоставляет JDBC драйвер с открытым исходным кодом, совместимый со стандартным интерфейсом JDBC. Вы можете легко интегрировать его в различные инструменты BI (такие как Tableau, Power BI, DBeaver и т.д.) для доступа к базе данных Selena, точно так же, как с традиционным JDBC драйвером. Значительным преимуществом этого драйвера является его поддержка высокоскоростной передачи данных на основе Apache Arrow, что значительно повышает эффективность запросов и передачи данных. Использование почти идентично традиционному MySQL JDBC драйверу.

ПРИМЕЧАНИЕ

- Если вы хотите запустить сервис FE с помощью командной строки, вы можете использовать один из следующих способов:
  - Указать переменную окружения `JAVA_TOOL_OPTIONS`.

    ```
    export JAVA_TOOL_OPTIONS="--add-opens=java.base/java.nio=org.apache.arrow.memory.core,ALL-UNNAMED"
    ```

  - Указать параметр конфигурации FE `JAVA_OPTS` в **fe.conf**. Таким образом, вы можете добавить другие значения `JAVA_OPTS`.

    ```
    JAVA_OPTS="--add-opens=java.base/java.nio=org.apache.arrow.memory.core,ALL-UNNAMED ..."
    ```

- Если вы хотите отлаживать в IntelliJ IDEA, вы должны добавить следующую опцию в `Build and run` в `Run/Debug Configurations`:

  ```
  --add-opens=java.base/java.nio=org.apache.arrow.memory.core,ALL-UNNAMED
  ```

Зависимости POM

Аналогично Python, вы также можете напрямую создать клиент ADBC в Java для чтения данных из Selena.

В этом процессе вам сначала нужно получить FlightInfo, а затем подключиться к каждому Endpoint для получения данных.

Пример кода:

```
public static void main(String[] args) throws Exception {
    try (BufferAllocator allocator = new RootAllocator()) {
        FlightSqlDriver driver = new FlightSqlDriver(allocator);

        Map<String, Object> parameters = new HashMap<>();
        String host = "localhost";
        int port = 9408;
        String uri = Location.forGrpcInsecure(host,
port).getUri().toString();

        AdbcDriver.PARAM_URI.set(parameters, uri);
        AdbcDriver.PARAM_USERNAME.set(parameters, "root");
        AdbcDriver.PARAM_PASSWORD.set(parameters, "");

        try (AdbcDatabase database = driver.open(parameters);
                AdbcConnection connection = database.connect();
                AdbcStatement statement = connection.createStatement()) {

            statement.setSqlQuery("SHOW DATABASES;");

            try (AdbcStatement.QueryResult result =
statement.executeQuery();
                    ArrowReader reader = result.getReader()) {

                int batchCount = 0;
                while (reader.loadNextBatch()) {
                    batchCount++;
                    VectorSchemaRoot root = reader.getVectorSchemaRoot();
                    System.out.println("Batch " + batchCount + ":");
                    System.out.println(root.contentToTSVString());
                }

                System.out.println("Total batches: " + batchCount);
            }
        }
    }
}
```

Рекомендации

- Среди трех методов подключения Java Arrow Flight SQL, упомянутых выше:
    - Если последующий анализ данных опирается на форматы данных на основе строк, рекомендуется использовать `jdbc:arrow-flight-sql`, который

возвращает данные в формате JDBC ResultSet.
  - Если анализ может напрямую обрабатывать формат Arrow или другие колоночные форматы данных, вы можете использовать Flight AdbcDriver или Flight JdbcDriver. Эти варианты возвращают данные в формате Arrow напрямую, избегая преобразования строк в столбцы и используя функции Arrow для ускорения разбора данных.
- Независимо от того, разбираете ли вы JDBC ResultSet или данные в формате Arrow, время разбора часто превышает время, затраченное на чтение самих данных. Если вы обнаружите, что Arrow Flight SQL не обеспечивает ожидаемое улучшение производительности по сравнению с `jdbc:mysql://`, рассмотрите возможность исследования того, занимает ли разбор данных слишком много времени.

- Для всех методов подключения чтение данных с JDK 17 обычно быстрее, чем с JDK 1.8.

- При чтении крупномасштабных наборов данных Arrow Flight SQL обычно потребляет меньше памяти по сравнению с `jdbc:mysql://`. Поэтому, если вы испытываете ограничения памяти, также стоит попробовать Arrow Flight SQL.

- В дополнение к трем методам подключения выше, вы также можете использовать нативный FlightClient для подключения к Arrow Flight Server, обеспечивая более гибкое параллельное чтение из нескольких endpoints. Java Flight AdbcDriver построен поверх FlightClient и предоставляет более простой интерфейс по сравнению с прямым использованием FlightClient.

Spark

В настоящее время официальный проект Arrow Flight не планирует поддерживать Spark или Flink. В будущем поддержка будет постепенно добавляться, чтобы позволить selena-spark-connector получать доступ к Selena через Arrow Flight SQL, с ожидаемым улучшением производительности чтения в несколько раз.

При доступе к Selena с помощью Spark, в дополнение к традиционным методам JDBC или Java клиента, вы также можете использовать компонент Spark-Flight-Connector с открытым исходным кодом для прямого чтения из и записи в Selena Flight SQL Server в качестве Spark DataSource. Этот подход, основанный на протоколе Apache Arrow Flight, предлагает следующие значительные преимущества:

- **Высокопроизводительная передача данных** Spark-Flight-Connector использует Apache Arrow в качестве формата передачи данных, обеспечивая обмен данными с нулевым копированием и высокой эффективностью. Преобразование между внутренним форматом данных `Block` Selena и Arrow очень эффективно, достигая улучшения производительности до 10 раз по сравнению с традиционными методами `CSV` или `JDBC`, и значительно снижая накладные расходы на передачу данных.
- **Нативная поддержка сложных типов данных** Формат данных Arrow нативно поддерживает сложные типы (такие как `Map`, `Array`, `Struct` и т.д.), обеспечивая лучшую адаптацию к сложным моделям данных Selena по сравнению с традиционными методами JDBC, и улучшая выразительность и совместимость данных.
- **Поддержка чтения, записи и потоковой записи** Компонент поддерживает Spark в качестве клиента Flight SQL для эффективных операций чтения и записи, включая операторы DML `insert`, `merge`, `update` и `delete`, и даже поддерживает потоковую запись, что делает его подходящим для сценариев обработки данных в реальном времени.
- **Поддержка проталкивания предикатов и обрезки столбцов** При чтении данных Spark-Flight-Connector поддерживает проталкивание предикатов и обрезку столбцов, позволяя фильтровать данные и выбирать столбцы на стороне Selena, значительно уменьшая объем передаваемых данных и улучшая производительность запросов.
- **Поддержка проталкивания агрегации и параллельного чтения** Агрегационные операции (такие как `sum`, `count`, `max`, `min` и т.д.) могут быть протолкнуты в Selena для выполнения, уменьшая вычислительную нагрузку на Spark. Также поддерживается параллельное чтение на основе разделения, улучшая эффективность чтения в сценариях с большими данными.
- **Лучше для сценариев больших данных** По сравнению с традиционными методами JDBC, протокол Flight SQL лучше подходит для крупномасштабных сценариев доступа с высокой параллельностью, позволяя Selena полностью использовать свои высокопроизводительные аналитические возможности.

4.4.4 Приложение

Следующий полный пример кода из учебника по использованию.

```
#
==============================================================================
# Тестовый скрипт Selena Arrow Flight SQL
#
==============================================================================
# pip install adbc_driver_manager adbc_driver_flightsql pandas
#
==============================================================================

#
==============================================================================
# Необходимые основные модули для подключения к Selena через Arrow Flight SQL
#
==============================================================================
import adbc_driver_manager
import adbc_driver_flightsql.dbapi as flight_sql

#
==============================================================================
# Опциональные модули для лучшей удобства использования и отладки
#
==============================================================================
import pandas as pd       # Опционально: для лучшего отображения результатов
с использованием DataFrame
import traceback          # Опционально: для детального отслеживания ошибок
во время выполнения SQL
import time               # Опционально: для измерения времени выполнения
SQL

#
==============================================================================
# Конфигурация Selena Flight SQL
#
==============================================================================
FE_HOST = "127.0.0.1"
FE_PORT = 9408

#
==============================================================================
# Подключение к Selena
#
==============================================================================
conn = flight_sql.connect(
    uri=f"grpc://{FE_HOST}:{FE_PORT}",
    db_kwargs={
```

```
        adbc_driver_manager.DatabaseOptions.USERNAME.value: "root",
        adbc_driver_manager.DatabaseOptions.PASSWORD.value: "",
    }
)

cursor = conn.cursor()

# ================================================================================
# Утилитные функции для лучшего форматирования вывода и выполнения SQL
# ================================================================================

def print_header(title: str):
    """
    Вывести заголовок раздела для лучшей читаемости.
    """
    print("\n" + "=" * 80)
    print(f"🔴 {title}")
    print("=" * 80)


def print_sql(sql: str):
    """
    Вывести SQL оператор перед выполнением.
    """
    print(f"\n🟢 SQL:\n{sql.strip()}")


def print_result(df: pd.DataFrame):
    """
    Вывести результат DataFrame в читаемом формате.
    """
    if df.empty:
        print("\n🔴 Result: (no rows returned)\n")
    else:
        print("\n🔴 Result:\n")
        print(df.to_string(index=False))


def print_error(e: Exception):
    """
    Вывести трассировку ошибки, если выполнение SQL не удалось.
    """
    print("\n⚠ Error occurred:")
    traceback.print_exc()
```

```
def execute(sql: str):
    """
    Выполнить SQL оператор и вывести результат и время выполнения.
    """
    print_sql(sql)
    try:
        start = time.time()  # Время начала для измерения времени выполнения
        cursor.execute(sql)
        result = cursor.fetchallarrow()  # Arrow Table
        df = result.to_pandas()          # Преобразовать в DataFrame для
лучшего отображения
        print_result(df)
        print(f"\n⏱  Execution time: {time.time() - start:.3f} seconds")
    except Exception as e:
        print_error(e)

#
==============================================================================
# Шаг 1: Удалить и создать базу данных
#
==============================================================================
print_header("Step 1: Drop and Create Database")
execute("DROP DATABASE IF EXISTS sr_arrow_flight_sql FORCE;")
execute("SHOW DATABASES;")
execute("CREATE DATABASE sr_arrow_flight_sql;")
execute("SHOW DATABASES;")
execute("USE sr_arrow_flight_sql;")

#
==============================================================================
# Шаг 2: Создать таблицу
#
==============================================================================
print_header("Step 2: Create Table")
execute("""
CREATE TABLE sr_arrow_flight_sql_test
(
    k0 INT,
    k1 DOUBLE,
    k2 VARCHAR(32) NULL DEFAULT "" COMMENT "",
    k3 DECIMAL(27,9) DEFAULT "0",
    k4 BIGINT NULL DEFAULT '10',
    k5 DATE
)
```

```
DISTRIBUTED BY HASH(k5) BUCKETS 5
PROPERTIES("replication_num" = "1");
""")

execute("SHOW CREATE TABLE sr_arrow_flight_sql_test;")

#
==============================================================================
# Шаг 3: Вставить данные
#
==============================================================================
print_header("Step 3: Insert Data")
execute("""
INSERT INTO sr_arrow_flight_sql_test VALUES
    (0, 0.1, "ID", 0.0001, 1111111111, '2025-04-21'),
    (1, 0.20, "ID_1", 1.00000001, 0, '2025-04-21'),
    (2, 3.4, "ID_1", 3.1, 123456, '2025-04-22'),
    (3, 4, "ID", 4, 4, '2025-04-22'),
    (4, 122345.54321, "ID", 122345.54321, 5, '2025-04-22');
""")

#
==============================================================================
# Шаг 4: Запросить данные
#
==============================================================================
print_header("Step 4: Query Data")
execute("SELECT * FROM sr_arrow_flight_sql_test ORDER BY k0;")

#
==============================================================================
# Шаг 5: Переменные сессии
#
==============================================================================
print_header("Step 5: Session Variables")
execute("SHOW VARIABLES LIKE '%query_mem_limit%';")
execute("SET query_mem_limit = 2147483648;")
execute("SHOW VARIABLES LIKE '%query_mem_limit%';")
execute("SHOW VARIABLES LIKE '%arrow_flight_proxy%';")
execute("SET arrow_flight_proxy_enabled = true;")
execute("SET arrow_flight_proxy = 'fe-proxy.example.com';")
execute("SHOW VARIABLES LIKE '%arrow_flight_proxy%';")

#
==============================================================================
# Шаг 6: Агрегационный запрос
```

```
# ==============================================================================
print_header("Step 6: Aggregation Query")
execute("""
SELECT k5, SUM(k1) AS total_k1, COUNT(1) AS row_count, AVG(k3) AS avg_k3
FROM sr_arrow_flight_sql_test
GROUP BY k5
ORDER BY k5;
""")

# ==============================================================================
# Шаг 7: Закрыть соединение
# ==============================================================================
print_header("Step 7: Close Connection")
cursor.close()
conn.close()
print("▫ Test completed successfully.")
```

4.5 Выгрузка данных с помощью INSERT INTO FILES

В этом разделе описывается, как выгружать данные из Selena в удалённое хранилище с помощью INSERT INTO FILES.

Начиная с версии v1.5.2, Selena поддерживает использование табличной функции FILES() для определения записываемого файла в удалённом хранилище. Затем вы можете объединить FILES() с командой INSERT для выгрузки данных из Selena в ваше удалённое хранилище.

По сравнению с другими методами экспорта данных, поддерживаемыми Selena, выгрузка данных с помощью INSERT INTO FILES обеспечивает более унифицированный и простой в использовании интерфейс. Вы можете выгружать данные непосредственно в удалённое хранилище, используя тот же синтаксис, который вы использовали для загрузки данных из него. Более того, этот метод поддерживает хранение файлов данных в разных путях хранилища путём извлечения значений указанного столбца, что позволяет управлять экспортированными данными в партиционированной структуре.

> **ПРИМЕЧАНИЕ**

> Обратите внимание, что выгрузка данных с помощью INSERT INTO FILES не поддерживает прямой экспорт данных в локальные файловые системы. Однако вы можете экспортировать данные в локальные файлы с помощью NFS. См. Выгрузка в локальные файлы с помощью NFS.

4.5.1 Подготовка

В следующем примере создаётся база данных `unload` и таблица `sales_records` в качестве объектов данных, которые могут быть использованы в последующем руководстве. Вы можете использовать свои собственные данные.

```
CREATE DATABASE unload;
USE unload;
CREATE TABLE sales_records(
    record_id     BIGINT,
    seller        STRING,
    store_id      INT,
    sales_time    DATETIME,
    sales_amt     DOUBLE
)
DUPLICATE KEY(record_id)
PARTITION BY date_trunc('day', sales_time)
DISTRIBUTED BY HASH(record_id);

INSERT INTO sales_records
VALUES
    (220313001,"Amy",1,"2022-03-13 12:00:00",8573.25),
    (220314002,"Bob",2,"2022-03-14 12:00:00",6948.99),
    (220314003,"Amy",1,"2022-03-14 12:00:00",4319.01),
    (220315004,"Carl",3,"2022-03-15 12:00:00",8734.26),
    (220316005,"Carl",3,"2022-03-16 12:00:00",4212.69),
    (220317006,"Bob",2,"2022-03-17 12:00:00",9515.88);
```

Таблица `sales_records` содержит ID транзакции `record_id`, продавца `seller`, ID магазина `store_id`, время `sales_time` и сумму продажи `sales_amt` для каждой транзакции. Она партиционирована на ежедневной основе в соответствии с `sales_time`.

Вам также необходимо подготовить систему удалённого хранилища с доступом на запись. Следующие примеры экспортируют данные в следующие удалённые хранилища:

- Кластер HDFS с включённым простым методом аутентификации.

- Bucket AWS S3, использующий учётные данные IAM User.

Подробнее о системах удалённого хранилища и методах учётных данных, поддерживаемых FILES(), см. Справочник SQL - FILES().

4.5.2 Выгрузка данных

INSERT INTO FILES поддерживает выгрузку данных в один файл или несколько файлов. Вы можете дополнительно партиционировать эти файлы данных, указав для них отдельные пути хранения.

При выгрузке данных с помощью INSERT INTO FILES вы должны вручную установить алгоритм сжатия, используя свойство `compression`. Для получения дополнительной информации об алгоритмах сжатия данных, поддерживаемых FILES, см. unload_data_param.

Выгрузка данных в несколько файлов

По умолчанию INSERT INTO FILES выгружает данные в несколько файлов данных, каждый размером 1 ГБ. Вы можете настроить размер файла с помощью свойства `target_max_file_size` (единица измерения: байты).

Следующий пример выгружает все строки данных в `sales_records` в виде нескольких файлов Parquet с префиксом `data1`. Размер каждого файла составляет 1 КБ.

> **ПРИМЕЧАНИЕ**
>
> Здесь установка `target_max_file_size` в 1 КБ используется для демонстрации выгрузки в несколько файлов с небольшим набором данных. В производственной среде настоятельно рекомендуется устанавливать это значение в диапазоне от сотен МБ до нескольких ГБ.

- **В S3**:

```
INSERT INTO
FILES(
    "path" = "s3://mybucket/unload/data1",
    "format" = "parquet",
    "compression" = "uncompressed",
    "target_max_file_size" = "1024", -- 1KB
    "aws.s3.access_key" = "xxxxxxxxxx",
    "aws.s3.secret_key" = "yyyyyyyyyy",
    "aws.s3.region" = "us-west-2"
)
SELECT * FROM sales_records;
```

- **В HDFS**:

```
INSERT INTO
FILES(
    "path" = "hdfs://xxx.xx.xxx.xx:9000/unload/data1",
    "format" = "parquet",
    "compression" = "uncompressed",
    "target_max_file_size" = "1024", -- 1KB
    "hadoop.security.authentication" = "simple",
    "username" = "xxxxx",
    "password" = "xxxxx"
)
SELECT * FROM sales_records;
```

Выгрузка данных в несколько файлов по разным путям

Вы также можете партиционировать файлы данных в разных путях хранения, извлекая значения указанного столбца с помощью свойства `partition_by`.

Следующий пример выгружает все строки данных в `sales_records` в виде нескольких файлов Parquet по пути **/unload/partitioned/**. Эти файлы хранятся в разных подпапках, различаемых по значениям в столбце `sales_time`.

- **В S3**:

```
INSERT INTO
FILES(
    "path" = "s3://mybucket/unload/partitioned/",
    "format" = "parquet",
    "compression" = "lz4",
    "partition_by" = "sales_time",
    "aws.s3.access_key" = "xxxxxxxxxx",
    "aws.s3.secret_key" = "yyyyyyyyyy",
    "aws.s3.region" = "us-west-2"
)
SELECT * FROM sales_records;
```

- **В HDFS**:

```
INSERT INTO
FILES(
    "path" = "hdfs://xxx.xx.xxx.xx:9000/unload/partitioned/",
    "format" = "parquet",
    "compression" = "lz4",
    "partition_by" = "sales_time",
    "hadoop.security.authentication" = "simple",
    "username" = "xxxxx",
    "password" = "xxxxx"
)
SELECT * FROM sales_records;
```

Выгрузка данных в один файл

Чтобы выгрузить данные в один файл данных, необходимо указать свойство `single` как `true`.

Следующий пример выгружает все строки данных в `sales_records` в виде одного файла Parquet с префиксом `data2`.

- **В S3**:

```
INSERT INTO
FILES(
    "path" = "s3://mybucket/unload/data2",
    "format" = "parquet",
    "compression" = "lz4",
    "single" = "true",
    "aws.s3.access_key" = "xxxxxxxxxx",
    "aws.s3.secret_key" = "yyyyyyyyyy",
    "aws.s3.region" = "us-west-2"
)
SELECT * FROM sales_records;
```

- **В HDFS**:

```
INSERT INTO
FILES(
    "path" = "hdfs://xxx.xx.xxx.xx:9000/unload/data2",
    "format" = "parquet",
    "compression" = "lz4",
    "single" = "true",
    "hadoop.security.authentication" = "simple",
    "username" = "xxxxx",
    "password" = "xxxxx"
)
SELECT * FROM sales_records;
```

Выгрузка в MinIO

Параметры, используемые для MinIO, отличаются от тех, которые используются для AWS S3.

Пример:

```
INSERT INTO
FILES(
    "path" = "s3://huditest/unload/data3",
    "format" = "parquet",
    "compression" = "zstd",
    "single" = "true",
    "aws.s3.access_key" = "xxxxxxxxxx",
    "aws.s3.secret_key" = "yyyyyyyyyy",
    "aws.s3.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.enable_ssl" = "false",
    "aws.s3.enable_path_style_access" = "true",
    "aws.s3.endpoint" = "http://minio:9000"
)
SELECT * FROM sales_records;
```

Выгрузка в локальные файлы с помощью NFS

Для доступа к файлам в NFS через протокол `file://` необходимо смонтировать NAS-устройство как NFS в одной и той же директории на каждом узле BE или CN.

Пример:

```
-- Выгрузка данных в CSV файлы.
INSERT INTO FILES(
  'path' = 'file:///home/ubuntu/csvfile/',
  'format' = 'csv',
  'csv.column_separator' = ',',
  'csv.row_delimitor' = '\n'
)
SELECT * FROM sales_records;

-- Выгрузка данных в Parquet файлы с версией Parquet 1.0.
INSERT INTO FILES(
  'path' = 'file:///home/ubuntu/parquetfile/',
  'format' = 'parquet',
  'parquet.version' = '1.0'
)
SELECT * FROM sales_records;
```

4.5.3 См. также

- Для получения дополнительных инструкций по использованию INSERT см. Справочник SQL - INSERT.
- Для получения дополнительных инструкций по использованию FILES() см. Справочник SQL - FILES().

4.6 Поддержка функций: Загрузка и выгрузка данных

4.7 Поддержка функций: Загрузка и выгрузка данных

В этом документе описаны функции различных методов загрузки и выгрузки данных, поддерживаемых Selena.

4.7.1 Формат файла

Форматы файлов для загрузки

	Источник данных	Формат файла						
		CSV	JSON [3]	Parquet	ORC	Avro	ProtoBuf	Thrift
Stream Load	Локальные файловые системы, приложения, коннекторы	Да	Да	Планируется	Планируется	Планируется		
INSERT from FILES	HDFS, S3, OSS, Azure, GCS, NFS(NAS) [5]	Да (v1.5.2+)	Планируется	Да (v1.5.2+)	Да (v1.5.2+)	Да (v1.5.2+)	Планируется	
Broker Load		Да	Да (v1.5.2+)	Да	Да	Планируется		
Routine Load	Kafka	Да	Да	Планируется	Планируется	Да (v1.5.2+) [1]	Планируется	Планируется
Spark Load		Да	Планируется	Да	Да	Планируется		
Коннекторы	Flink, Spark	Да	Да	Планируется	Планируется	Планируется		
Kafka Connector [2]	Kafka	Да (v1.5.2+)		Планируется	Планируется	Да (v1.5.2+)		Планируется
PIPE [4]	Соответствует INSERT from FILES							

> **ПРИМЕЧАНИЕ**
>
> [1], [2]: Требуется Schema Registry.
>
> [3]: JSON поддерживает различные форматы CDC. Подробнее о форматах JSON CDC, поддерживаемых Selena, см. в разделе Форматы JSON CDC.
>
> [4]: В настоящее время для загрузки с помощью PIPE поддерживается только INSERT from FILES.
>
> [5]: Вам нужно примонтировать устройство NAS как NFS в одну и ту же директорию каждого узла BE или CN для доступа к файлам в NFS через протокол `file://`.

Форматы JSON CDC

	Stream Load	Routine Load	Broker Load	INSERT from FILES	Kafka Connector [1]
Debezium	Планируется	Планируется	Планируется	Планируется	Да (v1.5.2+)
Canal	Планируется				
Maxwell					

> **ПРИМЕЧАНИЕ**
>
> [1]: При загрузке данных в формате Debezium CDC в таблицы Primary Key в Selena необходимо настроить параметр `transforms`.

Форматы файлов для выгрузки

	Цель		Формат файла			
	Формат таблицы	Удалённое хранилище	CSV	JSON	Parquet	ORC
INSERT INTO FILES	Н/Д	HDFS, S3, OSS, Azure, GCS, NFS(NAS) [3]	Да (v1.5.2+)	Планируется	Да (v1.5.2+)	Да (v1.5.2+)
INSERT INTO Catalog	Hive	HDFS, S3, OSS, Azure, GCS	Да (v1.5.2+)	Планируется	Да (v1.5.2+)	Да (v1.5.2+)
	Iceberg	HDFS, S3, OSS, Azure, GCS	Планируется	Планируется	Да (v1.5.2+)	Планируется
	Hudi/Delta		Планируется			
EXPORT	Н/Д	HDFS, S3, OSS, Azure, GCS	Да [1]	Планируется	Планируется	Планируется

PIPE	Планируется [2]

> **ПРИМЕЧАНИЕ**
>
> [1]: Поддерживается настройка процесса Broker.
>
> [2]: В настоящее время выгрузка данных с помощью PIPE не поддерживается.
>
> [3]: Вам нужно примонтировать устройство NAS как NFS в одну и ту же директорию каждого узла BE или CN для доступа к файлам в NFS через протокол `file://`.

4.7.2 Параметры, связанные с форматом файла

Параметры формата файла для загрузки

<table>
<tr><th rowspan="2">Формат файла</th><th rowspan="2">Параметр</th><th colspan="5">Метод загрузки</th></tr>
<tr><th>Stream Load</th><th>INSERT from FILES</th><th>Broker Load</th><th>Routine Load</th><th>Spark Load</th></tr>
<tr><td rowspan="6">CSV</td><td>column_separator</td><td>Да</td><td rowspan="6">Да (v1.5.2+)</td><td colspan="3">Да [1]</td></tr>
<tr><td>row_delimiter</td><td>Да</td><td>Да [2] (v1.5.2+)</td><td>Да [3] (v1.5.2+)</td><td>Планируется</td></tr>
<tr><td>enclose</td><td rowspan="3">Да (v1.5.2+)</td><td rowspan="3">Да (v1.5.2+)</td><td>Да (v1.5.2+)</td><td rowspan="3">Планируется</td></tr>
<tr><td>escape</td><td></td></tr>
<tr><td>skip_header</td><td>Планируется</td></tr>
<tr><td>trim_space</td><td></td><td>Да (v1.5.2+)</td></tr>
<tr><td rowspan="4">JSON</td><td>jsonpaths</td><td rowspan="4">Да</td><td rowspan="4">Планируется</td><td rowspan="4">Да (v1.5.2+)</td><td rowspan="3">Да</td><td rowspan="4">Планируется</td></tr>
<tr><td>strip_outer_array</td></tr>
<tr><td>json_root</td></tr>
<tr><td>ignore_json_size</td><td>Планируется</td></tr>
</table>

> **ПРИМЕЧАНИЕ**
>
> [1]: Соответствующий параметр — `COLUMNS TERMINATED BY`.
>
> [2]: Соответствующий параметр — `ROWS TERMINATED BY`.
>
> [3]: Соответствующий параметр — `ROWS TERMINATED BY`.

Параметры формата файла для выгрузки

Формат файла	Параметр	Метод выгрузки	
		INSERT INTO FILES	EXPORT
CSV	column_separator	Да (v1.5.2+)	Да
	line_delimiter [1]		

> **ПРИМЕЧАНИЕ**
>
> [1]: Соответствующий параметр при загрузке данных — `row_delimiter`.

4.7.3 Форматы сжатия

Форматы сжатия для загрузки

Формат файла	Формат сжатия	Метод загрузки				
		Stream Load	Broker Load	INSERT from FILES	Routine Load	Spark Load
CSV	• deflate • bzip2 • gzip • lz4_frame • zstd	Да [1]	Да [2]	Планируется	Планируется	Планируется
JSON		Да (v1.5.2+) [3]	Планируется	Н/Д	Планируется	Н/Д
Parquet	• gzip • lz4 • snappy • zlib • zstd	Н/Д	Да [4]		Планируется	Да [4]
ORC						

> **ПРИМЕЧАНИЕ**
>
> [1]: В настоящее время только при загрузке файлов CSV с помощью Stream Load можно указать формат сжатия с помощью `format=gzip`, что указывает на файлы CSV, сжатые gzip. Форматы `deflate` и `bzip2` также поддерживаются.
>
> [2]: Broker Load не поддерживает указание формата сжатия файлов CSV с помощью параметра `format`. Broker Load определяет формат сжатия по расширению файла. Расширение файлов, сжатых gzip, — `.gz`, а файлов, сжатых zstd, — `.zst`. Кроме того, другие параметры, связанные с `format`, такие как `trim_space` и `enclose`, также не поддерживаются.
>
> [3]: Поддерживается указание формата сжатия с помощью `compression = gzip`.
>
> [4]: Поддерживается библиотекой Arrow. Настройка параметра `compression` не требуется.

Форматы сжатия для выгрузки

Формат файла	Формат сжатия	Метод выгрузки				
		INSERT INTO FILES	INSERT INTO Catalog			EXPORT
			Hive	Iceberg	Hudi/Delta	
CSV	• deflate • bzip2 • gzip • lz4_frame • zstd	Планируется	Планируется	Планируется	Планируется	Планируется
JSON	Н/Д	Н/Д	Н/Д	Н/Д	Н/Д	Н/Д
Parquet	• gzip • lz4 • snappy • zstd	Да (v1.5.2+)	Да (v1.5.2+)	Да (v1.5.2+)	Планируется	Н/Д
ORC						

4.7.4 Учётные данные

Загрузка — Аутентификация

Аутентификация	Метод загрузки

	Stream Load	INSERT from FILES	Broker Load	Routine Load	External Catalog
Единичный Kerberos	Н/Д	Да (v1.5.2+)	Да [1] (версии ранее v1.5.2)	Да [2] (v1.5.2+)	Да
Kerberos Ticket Granting Ticket (TGT)	Н/Д	Планируется			Да (v1.5.2+)
Единый KDC, множественный Kerberos	Н/Д				
Базовая аутентификация доступа (пара ключей доступа, IAM Role)	Н/Д	Да (HDFS и S3-совместимое объектное хранилище)		Да [3]	Да

ПРИМЕЧАНИЕ

[1]: Для HDFS Selena поддерживает как простую аутентификацию, так и аутентификацию Kerberos.

[2]: Когда протокол безопасности установлен на `sasl_plaintext` или `sasl_ssl`, поддерживаются как аутентификация SASL, так и GSSAPI (Kerberos).

[3]: Когда протокол безопасности установлен на `sasl_plaintext` или `sasl_ssl`, поддерживаются как аутентификация SASL, так и PLAIN.

Выгрузка — Аутентификация

	INSERT INTO FILES	EXPORT
Единичный Kerberos	Планируется	Планируется

4.7.5 Загрузка — Другие параметры и функции

Параметр и функция	Метод загрузки						
	Stream Load	INSERT from FILES	INSERT from SELECT/VALUES	Broker Load	PIPE	Routine Load	Spark Load
partial_update	Да (v1.5.2+)	Да [1] (v1.5.2+)		Да (v1.5.2+)	Н/Д	Да (v1.5.2+)	Планируется
partial_update_mode	Да (v1.5.2+)	Планируется		Да (v1.5.2+)	Н/Д	Планируется	Планируется
COLUMNS FROM PATH	Н/Д	Да (v1.5.2+)	Н/Д	Да	Н/Д	Н/Д	Да
timezone или сессионная переменная time_zone [2]	Да [3]	Да [4]	Да [4]	Да [4]	Планируется	Да [4]	Планируется

Точность времени — микросекунды	Да	Да	Да	Да (v1.5.2+)	Планируется	Да	Да

ПРИМЕЧАНИЕ

[1]: Начиная с v1.5.2, Selena поддерживает частичные обновления в режиме Row для INSERT INTO путём указания списка столбцов.

[2]: Установка часового пояса с помощью параметра или сессионной переменной влияет на результаты, возвращаемые функциями strftime(), alignment_timestamp() и from_unixtime().

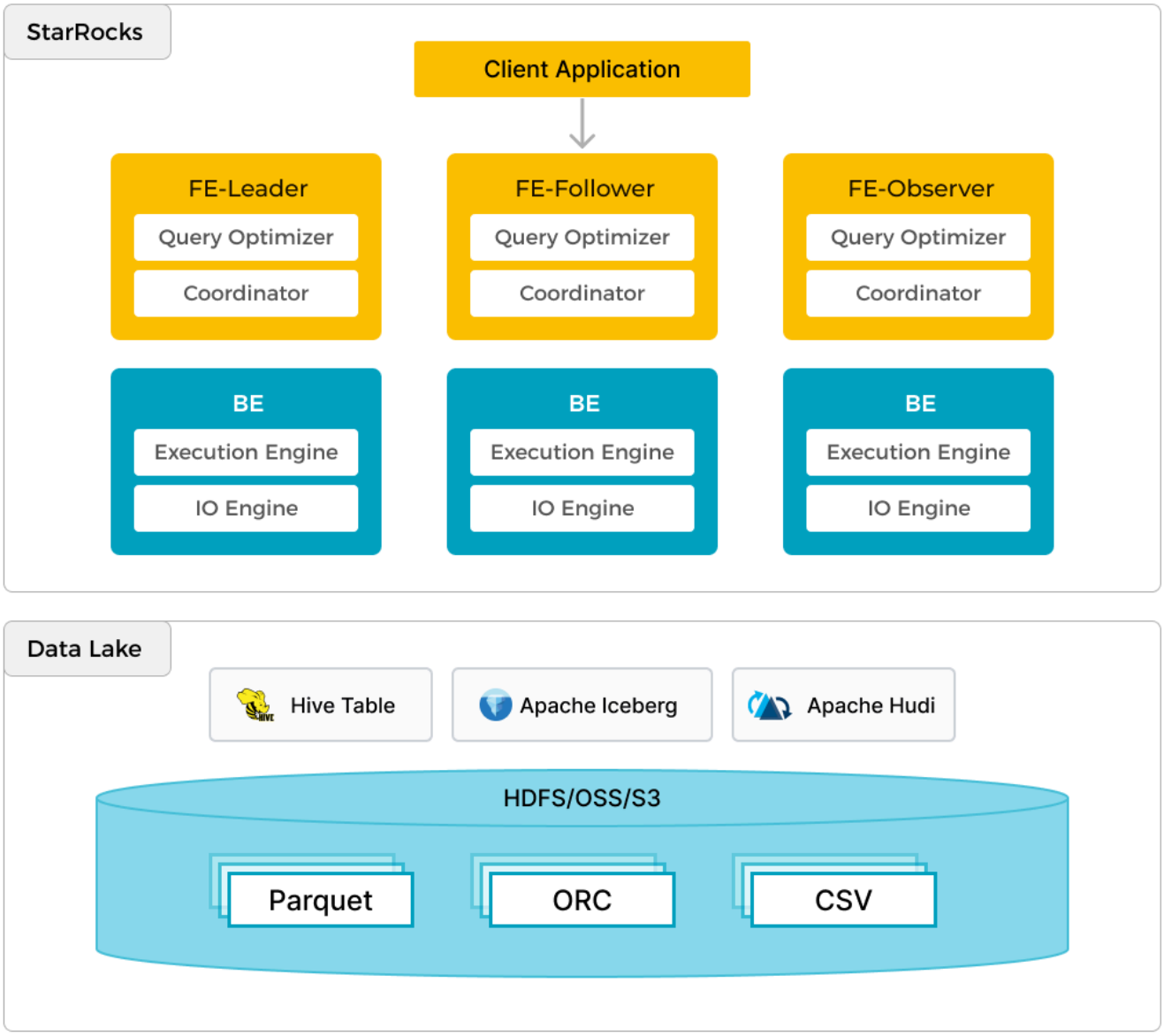
[3]: Поддерживается только параметр `timezone`.

[4]: Поддерживается только сессионная переменная `time_zone`.

4.7.6 Выгрузка — Другие параметры и функции

Параметр и функция	INSERT INTO FILES	EXPORT
target_max_file_size	Да (v1.5.2+)	Планируется
single		
Partitioned_by		
Сессионная переменная time_zone	Планируется	
Точность времени — микросекунды	Планируется	Планируется

5 Data Lakehouse



Помимо эффективной аналитики локальных данных, Selena может работать как вычислительный движок для анализа данных, хранящихся в озёрах данных, таких как Apache Hudi, Apache Iceberg и Delta Lake. Одной из ключевых особенностей Selena является внешний каталог, который служит связующим звеном с внешним metastore. Эта функциональность предоставляет пользователям возможность бесшовно выполнять запросы к внешним источникам данных, устраняя необходимость в миграции данных. Таким образом, пользователи могут анализировать данные из различных систем, таких как HDFS и Amazon S3, в различных форматах файлов, таких как Parquet, ORC, CSV и др.

На приведённом рисунке показан сценарий аналитики озера данных, где Selena отвечает за вычисления и анализ данных, а озеро данных — за хранение, организацию и обслуживание данных. Озёра данных позволяют пользователям хранить данные в открытых форматах хранения и использовать гибкие схемы для создания отчётов на основе «единого источника истины» для различных сценариев использования BI, AI, ad-hoc и отчётности. Selena в полной мере использует преимущества своего векторизованного движка и CBO, значительно повышая производительность аналитики озёр данных.

5.1 Ключевые идеи

- Открытые форматы данных: Поддерживает различные типы данных, включая JSON, Parquet и Avro, облегчая хранение и обработку как структурированных, так и неструктурированных данных.
- Управление метаданными: Реализует общий слой метаданных, часто используя форматы, такие как табличный формат Iceberg, для эффективной организации и управления данными.
- Разнообразные механизмы запросов: Включает несколько движков, таких как улучшенные версии Presto и Spark, для обслуживания различных сценариев аналитики и ИИ.
- Управление и безопасность: Содержит надёжные встроенные механизмы для безопасности данных, конфиденциальности и соответствия требованиям, обеспечивая целостность и достоверность данных.

5.2 Преимущества архитектуры Data Lakehouse

- Гибкость и масштабируемость: Беспрепятственно управляет разнообразными типами данных и масштабируется в соответствии с потребностями организации.
- Экономическая эффективность: Предлагает экономичную альтернативу для хранения и обработки данных по сравнению с традиционными методами.
- Улучшенное управление данными: Улучшает контроль, управление и целостность данных, обеспечивая надёжную и безопасную обработку данных.
- Готовность к ИИ и аналитике: Идеально подходит для сложных аналитических задач, включая машинное обучение и обработку данных на основе ИИ.

5.3 Подход Selena

Ключевые моменты для рассмотрения:

- Стандартизация интеграции с catalog или сервисами метаданных
- Эластичная масштабируемость вычислительных узлов
- Гибкие механизмы кэширования

5.4 Catalog

Selena имеет два типа catalog: внутренние и внешние. Внутренний catalog содержит метаданные для данных, хранящихся в базах данных Selena. Внешние catalog используются для работы с данными, хранящимися внешне, включая данные, управляемые Hive, Iceberg, Delta Lake и Hudi. Существует множество других внешних систем, ссылки находятся в разделе «Дополнительная информация» внизу страницы.

5.5 Масштабирование вычислительных узлов (CN)

Разделение хранения и вычислений снижает сложность масштабирования. Поскольку вычислительные узлы Selena хранят только локальный кэш, узлы могут добавляться или удаляться в зависимости от нагрузки.

5.6 Data Cache

Кэш на вычислительных узлах является опциональным. Если ваши вычислительные узлы быстро запускаются и останавливаются на основе быстро меняющихся паттернов нагрузки, или ваши запросы часто касаются только самых последних данных, может не иметь смысла кэшировать данные.

5.7 Обзор Catalog

5.7.1 Обзор

Эта тема описывает, что такое catalog, и как управлять и запрашивать внутренние и внешние данные с помощью catalog.

Selena поддерживает функцию catalog начиная с версии v1.5.2. Catalog позволяют управлять внутренними и внешними данными в одной системе и предлагают гибкий способ для легкого запроса и анализа данных, хранящихся в различных внешних системах.

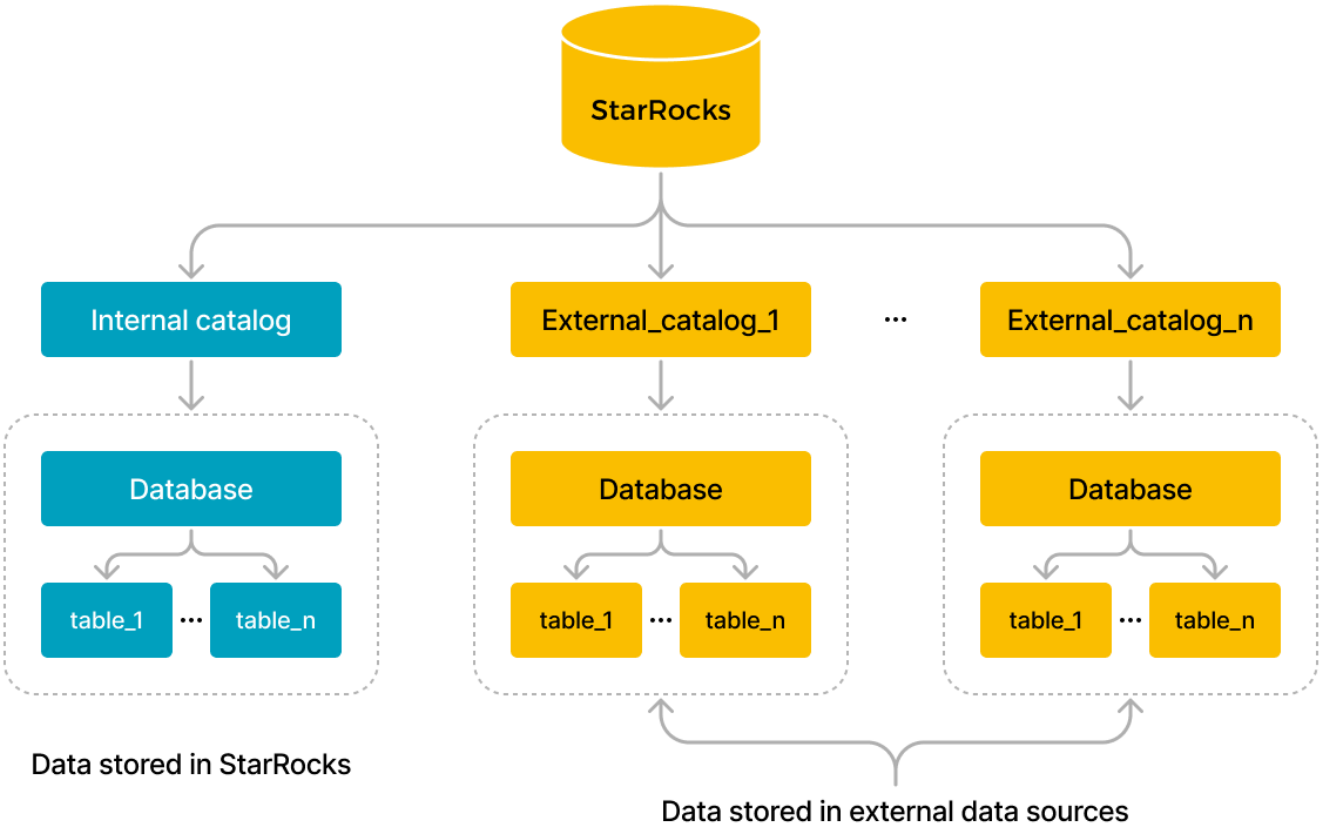
Основные понятия

- **Внутренние данные**: данные, хранящиеся в Selena.

- **Внешние данные**: данные, хранящиеся во внешнем источнике данных, таком как Apache Hive™, Apache Iceberg, Apache Hudi, Delta Lake и JDBC.

Catalog

В настоящее время Selena предоставляет два типа catalog: internal catalog и external catalog.



- **Internal catalog** управляет внутренними данными Selena. Например, если вы выполняете операторы CREATE DATABASE или CREATE TABLE для создания базы данных или таблицы, база данных или таблица сохраняется в internal catalog. Каждый cluster Selena имеет только один internal catalog с именем default_catalog.

- **External catalog** действует как ссылка на внешне управляемые metastore, что предоставляет Selena прямой доступ к внешним источникам данных. Вы можете запрашивать внешние данные напрямую без загрузки или миграции данных. В настоящее время Selena поддерживает следующие типы external catalog:

  - Hive catalog: используется для запроса данных из Hive.
  - Iceberg catalog: используется для запроса данных из Iceberg.

- Hudi catalog: используется для запроса данных из Hudi.
- Delta Lake catalog: используется для запроса данных из Delta Lake.
- JDBC catalog: используется для запроса данных из источников данных, совместимых с JDBC.
- Elasticsearch catalog: используется для запроса данных из Elasticsearch. Elasticsearch catalog поддерживаются начиная с версии v1.5.2.
- Paimon catalog: используется для запроса данных из Paimon. Paimon catalog поддерживаются начиная с версии v1.5.2.
- Unified catalog: используется для запроса данных из источников данных Hive, Iceberg, Hudi и Delta Lake как единого источника данных. Unified catalog поддерживаются начиная с версии v1.5.2.

Selena взаимодействует со следующими двумя компонентами внешних источников данных при запросе внешних данных:

- **Metastore service**: используется FE для доступа к метаданным внешних источников данных. FE генерируют план выполнения запроса на основе метаданных.
- **Data storage system**: используется для хранения внешних данных. Как распределенные файловые системы, так и системы объектного хранилища могут использоваться в качестве систем хранения данных для хранения файлов данных в различных форматах. После того как FE распределяют план выполнения запроса всем BE или CN, все BE или CN сканируют целевые внешние данные параллельно, выполняют вычисления, а затем возвращают результат запроса.

Доступ к catalog

Вы можете использовать оператор SET CATALOG для переключения на указанный catalog в текущей сессии. Затем вы можете запрашивать данные, используя этот catalog.

Запрос данных

Запрос внутренних данных

Для запроса данных в Selena см. Default catalog.

Запрос внешних данных

Для запроса данных из внешних источников данных см. Query external data.

Кросс-catalog запрос

Чтобы выполнить федеративный запрос между catalog из вашего текущего catalog, укажите данные, которые вы хотите запросить, в формате `catalog_name.database_name` или `catalog_name.database_name.table_name`.

- Запрос `hive_table` в `hive_db`, когда текущая сессия находится в `default_catalog.olap_db`.

```
SELECT * FROM hive_catalog.hive_db.hive_table;
```

- Запрос `olap_table` в `default_catalog`, когда текущая сессия находится в `hive_catalog.hive_db`.

```
SELECT * FROM default_catalog.olap_db.olap_table;
```

- Выполнение JOIN запроса к `hive_table` в `hive_catalog` и `olap_table` в `default_catalog`, когда текущая сессия находится в `hive_catalog.hive_db`.

```
SELECT * FROM hive_table h JOIN default_catalog.olap_db.olap_table o
WHERE h.id = o.id;
```

- Выполнение JOIN запроса к `hive_table` в `hive_catalog` и `olap_table` в `default_catalog` с использованием предложения JOIN, когда текущая сессия находится в другом catalog.

```
SELECT * FROM hive_catalog.hive_db.hive_table h JOIN
default_catalog.olap_db.olap_table o WHERE h.id = o.id;
```

5.7.2 Default catalog

В этом разделе описывается, что такое default catalog и как запрашивать внутренние данные Selena с его помощью.

Selena версии 2.3 и новее предоставляет внутренний catalog для управления внутренними данными Selena. Каждый cluster Selena имеет только один внутренний catalog с именем

`default_catalog`. В настоящее время вы не можете изменить имя внутреннего catalog или создать новый внутренний catalog.

Запрос внутренних данных

1. Подключитесь к вашему cluster Selena.

   - Если вы используете MySQL-клиент для подключения к cluster Selena, вы автоматически попадете в `default_catalog` после подключения.
   - Если вы используете JDBC для подключения к cluster Selena, вы можете перейти непосредственно к целевой базе данных в default catalog, указав `default_catalog.db_name` при подключении.
2. (Опционально) Используйте SHOW DATABASES для просмотра баз данных:

   ```
   SHOW DATABASES;
   ```

   Or

   ```
   SHOW DATABASES FROM <catalog_name>;
   ```

3. (Опционально) Используйте SET CATALOG для переключения на целевой catalog в текущей сессии:

   ```
   SET CATALOG <catalog_name>;
   ```

   Затем используйте USE для указания активной базы данных в текущей сессии:

   ```
   USE <db_name>;
   ```

   Или вы можете использовать USE для прямого перехода к активной базе данных в целевом catalog:

   ```
   USE <catalog_name>.<db_name>;
   ```

4. Используйте SELECT для запроса внутренних данных:

```
SELECT * FROM <table_name>;
```

Если вы не указали активную базу данных на предыдущих шагах, вы можете указать её непосредственно в запросе SELECT:

```
SELECT * FROM <db_name>.<table_name>;
```

Или

```
SELECT * FROM default_catalog.<db_name>.<table_name>;
```

Примеры

Для запроса данных в `olap_db.olap_table` вы можете выполнить одну из следующих операций:

```
USE olap_db;
SELECT * FROM olap_table limit 1;
```

Or

```
SELECT * FROM olap_db.olap_table limit 1;
```

Or

```
SELECT * FROM default_catalog.olap_db.olap_table limit 1;
```

Справочные материалы

Для запроса данных из внешних источников данных см. Запрос внешних данных.

5.7.3 Hive catalog

Hive catalog - это тип external catalog, который поддерживается Selena начиная с версии v1.5.2. В рамках Hive catalog вы можете:

- Напрямую запрашивать данные, хранящиеся в Hive, без необходимости вручную создавать таблицы.
- Использовать INSERT INTO или асинхронные материализованные представления (которые поддерживаются начиная с версии v1.5.2) для обработки данных, хранящихся в Hive, и загрузки данных в Selena.
- Выполнять операции в Selena для создания или удаления баз данных и таблиц Hive, или загружать данные из таблиц Selena в таблицы Hive в форматах Parquet (поддерживается начиная с версии v1.5.2), ORC или Textfile (поддерживается начиная с версии v1.5.2) с помощью INSERT INTO.

Чтобы обеспечить успешное выполнение SQL-операций на вашем Hive cluster, ваш Selena cluster должен иметь доступ к системе хранения и metastore вашего Hive cluster. Selena поддерживает следующие системы хранения и metastore:

- Distributed file system (HDFS) или object storage, такие как AWS S3, Microsoft Azure Storage, Google GCS или другие S3-совместимые системы хранения (например, MinIO)

- Metastore, такие как Hive metastore или AWS Glue

> **ПРИМЕЧАНИЕ**
>
> Если вы выберете AWS S3 в качестве хранилища, вы можете использовать HMS или AWS Glue в качестве metastore. Если вы выберете любую другую систему хранения, вы можете использовать только HMS в качестве metastore.

Примечания по использованию

- Selena поддерживает запросы к таблицам Hive в форматах файлов Parquet, ORC, Textfile, Avro, RCFile и SequenceFile:

    - Файлы Parquet поддерживают следующие форматы сжатия: SNAPPY, LZ4, ZSTD, GZIP и NO_COMPRESSION. Начиная с версии v1.5.2, файлы Parquet также поддерживают формат сжатия LZO.
    - Файлы ORC поддерживают следующие форматы сжатия: ZLIB, SNAPPY, LZO, LZ4, ZSTD и NO_COMPRESSION.
    - Файлы Textfile поддерживают формат сжатия LZO начиная с версии v1.5.2.

- Типы данных Hive, которые Selena не поддерживает: INTERVAL, BINARY и UNION. Кроме того, Selena не поддерживает типы данных MAP и STRUCT для таблиц Hive в формате Textfile.

- Selena поддерживает загрузку данных в таблицы Hive в форматах Parquet (поддерживается начиная с версии v1.5.2), ORC или Textfile (поддерживается начиная с версии v1.5.2):

  - Файлы Parquet и ORC поддерживают следующие форматы сжатия: NO_COMPRESSION, SNAPPY, LZ4, ZSTD и GZIP.
  - Файлы Textfile поддерживают формат сжатия NO_COMPRESSION.

  Вы можете использовать свойство таблицы `compression_codec` или системную переменную `connector_sink_compression_codec` для указания алгоритма сжатия, используемого при загрузке данных в таблицы Hive.

  При записи в таблицу Hive, если свойства таблицы включают compression codec, Selena будет приоритетно использовать этот алгоритм для сжатия записываемых данных. В противном случае будет использован алгоритм сжатия, установленный в системной переменной `connector_sink_compression_codec`.

Подготовка к интеграции

Перед созданием Hive catalog убедитесь, что ваш Selena cluster может интегрироваться с системой хранения и metastore вашего Hive cluster.

AWS IAM

Если ваш Hive cluster использует AWS S3 в качестве хранилища или AWS Glue в качестве metastore, выберите подходящий метод аутентификации и выполните необходимые подготовительные действия, чтобы ваш Selena cluster мог получить доступ к соответствующим облачным ресурсам AWS.

Рекомендуются следующие методы аутентификации:

- Instance profile
- Assumed role
- IAM user

Из этих трех методов аутентификации наиболее широко используется instance profile.

Для получения дополнительной информации см. Подготовка к аутентификации в AWS IAM.

HDFS

Если вы выбрали HDFS в качестве хранилища, настройте ваш Selena cluster следующим образом:

- (Опционально) Установите имя пользователя, которое используется для доступа к вашему HDFS cluster и Hive metastore. По умолчанию Selena использует имя пользователя процессов FE и BE или CN для доступа к вашему HDFS cluster и Hive metastore. Вы также можете установить имя пользователя, добавив `export HADOOP_USER_NAME="<user_name>"` в начало файла **fe/conf/hadoop_env.sh** каждого FE и в начало файла **be/conf/hadoop_env.sh** каждого BE или файла **cn/conf/hadoop_env.sh** каждого CN. После установки имени пользователя в этих файлах перезапустите каждый FE и каждый BE или CN, чтобы настройки параметров вступили в силу. Вы можете установить только одно имя пользователя для каждого Selena cluster.

- Когда вы запрашиваете данные Hive, FE и BE или CN вашего Selena cluster используют HDFS client для доступа к вашему HDFS cluster. В большинстве случаев вам не нужно настраивать ваш Selena cluster для этой цели, и Selena запускает HDFS client с использованием конфигураций по умолчанию. Вам нужно настроить ваш Selena cluster только в следующих ситуациях:

  - High availability (HA) включена для вашего HDFS cluster: Добавьте файл **hdfs-site.xml** вашего HDFS cluster в путь **$FE_HOME/conf** каждого FE и в путь **$BE_HOME/conf** каждого BE или путь **$CN_HOME/conf** каждого CN.
  - View File System (ViewFs) включена для вашего HDFS cluster: Добавьте файл **core-site.xml** вашего HDFS cluster в путь **$FE_HOME/conf** каждого FE и в путь **$BE_HOME/conf** каждого BE или путь **$CN_HOME/conf** каждого CN.

> **ПРИМЕЧАНИЕ**
>
> Если при отправке запроса возвращается ошибка, указывающая на неизвестный хост, вы должны добавить сопоставление между именами хостов и IP-адресами узлов вашего HDFS cluster в путь **/etc/hosts**.

Аутентификация Kerberos

Если аутентификация Kerberos включена для вашего HDFS cluster или Hive metastore, настройте ваш Selena cluster следующим образом:

- Выполните команду `kinit -kt keytab_path principal` на каждом FE и каждом BE или CN для получения Ticket Granting Ticket (TGT) из Key Distribution Center (KDC). Для выполнения этой команды вы должны иметь разрешения на доступ к вашему HDFS cluster и Hive metastore. Обратите внимание, что доступ к KDC с помощью этой команды зависит от времени. Поэтому вам нужно использовать cron для периодического выполнения этой команды.
- Добавьте `JAVA_OPTS="-Djava.security.krb5.conf=/etc/krb5.conf"` в файл **$FE_HOME/conf/fe.conf** каждого FE и в файл **$BE_HOME/conf/be.conf** каждого BE или файл **$CN_HOME/conf/cn.conf** каждого CN. В этом примере `/etc/krb5.conf` - это путь сохранения файла **krb5.conf**. Вы можете изменить путь в соответствии с вашими потребностями.

Создание Hive catalog

Синтаксис

```
CREATE EXTERNAL CATALOG <catalog_name>
[COMMENT <comment>]
PROPERTIES
(
    "type" = "hive",
    GeneralParams,
    MetastoreParams,
    StorageCredentialParams,
    MetadataUpdateParams
)
```

Параметры

catalog_name

Имя Hive catalog. Соглашения об именовании следующие:

- Имя может содержать буквы, цифры (0-9) и подчеркивания (_). Оно должно начинаться с буквы.
- Имя чувствительно к регистру и не может превышать 1023 символа в длину.

comment

Описание Hive catalog. Этот параметр необязателен.

type

Тип вашего источника данных. Установите значение `hive`.

GeneralParams

Набор общих параметров.

В следующей таблице описаны параметры, которые вы можете настроить в `GeneralParams`.

Параметр	Обязательный	Описание
enable_recursive_listing	Нет	Указывает, читает ли Selena данные из таблицы и ее partitions, а также из подкаталогов в физических расположениях таблицы и ее partitions. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`. Значение `true` указывает на рекурсивный список подкаталогов, а значение `false` указывает на игнорирование подкаталогов.

MetastoreParams

Набор параметров о том, как Selena интегрируется с metastore вашего источника данных.

Hive metastore

Если вы выбрали Hive metastore в качестве metastore вашего источника данных, настройте `MetastoreParams` следующим образом:

```
"hive.metastore.type" = "hive",
"hive.metastore.uris" = "<hive_metastore_uri>"
```

> **ПРИМЕЧАНИЕ**
>
> Перед запросом данных Hive вы должны добавить сопоставление между именами хостов и IP-адресами узлов вашего Hive metastore в путь `/etc/hosts`. В противном случае Selena может не получить доступ к вашему Hive metastore при запуске запроса.

В следующей таблице описан параметр, который вам нужно настроить в `MetastoreParams`.

Параметр	Обязательный	Описание
hive.metastore.type	Да	Тип metastore, который вы используете для вашего Hive cluster. Установите значение `hive`.
hive.metastore.uris	Да	URI вашего Hive metastore. Формат: `thrift://<metastore_IP_address>:<metastore_port>`. Если high availability (HA) включена для вашего Hive metastore, вы можете указать несколько URI metastore и разделить их запятыми (`,`), например, `"thrift://<metastore_IP_address_1>:<metastore_port_1>,thrift://<metastore_IP_address_2>:<metastore_port_2>,thrift://<metastore_IP_address_3>:<metastore_port_3>"`.

AWS Glue

Если вы выбрали AWS Glue в качестве metastore вашего источника данных, что поддерживается только при выборе AWS S3 в качестве хранилища, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации на основе instance profile, настройте `MetastoreParams` следующим образом:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "true",
"aws.glue.region" = "<aws_glue_region>"
```

- Чтобы выбрать метод аутентификации на основе assumed role, настройте `MetastoreParams` следующим образом:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "true",
"aws.glue.iam_role_arn" = "<iam_role_arn>",
"aws.glue.region" = "<aws_glue_region>"
```

- Чтобы выбрать метод аутентификации на основе IAM user, настройте `MetastoreParams` следующим образом:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "false",
"aws.glue.access_key" = "<iam_user_access_key>",
"aws.glue.secret_key" = "<iam_user_secret_key>",
"aws.glue.region" = "<aws_s3_region>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `MetastoreParams`.

Параметр	Обязательный	Описание
hive.metastore.type	Да	Тип metastore, который вы используете для вашего Hive cluster. Установите значение `glue`.
aws.glue.use_instance_profile	Да	Указывает, включить ли метод аутентификации на основе instance profile и метод аутентификации на основе assumed role. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`.
aws.glue.iam_role_arn	Нет	ARN IAM роли, которая имеет привилегии на ваш AWS Glue Data Catalog. Если вы используете метод аутентификации на основе assumed role для доступа к AWS Glue, вы должны указать этот параметр.
aws.glue.region	Да	Регион, в котором находится ваш AWS Glue Data Catalog. Пример: `us-west-1`.
aws.glue.access_key	Нет	Access key вашего AWS IAM user. Если вы используете метод аутентификации на основе IAM user для доступа к AWS Glue, вы должны указать этот параметр.
aws.glue.secret_key	Нет	Secret key вашего AWS IAM user. Если вы используете метод аутентификации на основе IAM user для доступа к AWS Glue, вы должны указать этот параметр.
hive.metastore.glue.catalogid	Нет	ID AWS Glue Data Catalog для использования. Если не указан, используется catalog в текущей учетной записи AWS. Вы должны указать этот параметр, когда вам нужно получить доступ к Glue Data Catalog в другой учетной записи AWS (cross-account access).

Для получения информации о том, как выбрать метод аутентификации для доступа к AWS Glue и как настроить политику контроля доступа в AWS IAM Console, см. Параметры аутентификации для доступа к AWS Glue.

StorageCredentialParams

Набор параметров о том, как Selena интегрируется с вашей системой хранения. Этот набор параметров необязателен.

Если вы используете HDFS в качестве хранилища, вам не нужно настраивать `StorageCredentialParams`.

Если вы используете AWS S3, другую S3-совместимую систему хранения, Microsoft Azure Storage или Google GCS в качестве хранилища, вы должны настроить `StorageCredentialParams`.

AWS S3

Если вы выбрали AWS S3 в качестве хранилища для вашего Hive cluster, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации на основе instance profile, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.region" = "<aws_s3_region>"
```

- Чтобы выбрать метод аутентификации на основе assumed role, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.iam_role_arn" = "<iam_role_arn>",
"aws.s3.region" = "<aws_s3_region>"
```

- Чтобы выбрать метод аутентификации на основе IAM user, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "false",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>",
"aws.s3.region" = "<aws_s3_region>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
aws.s3.use_instance_profile	Да	Указывает, включить ли метод аутентификации на основе instance profile и метод аутентификации на основе assumed role. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`.
aws.s3.iam_role_arn	Нет	ARN IAM роли, которая имеет привилегии на ваш AWS S3 bucket. Если вы используете метод аутентификации на основе assumed role для доступа к AWS S3, вы должны указать этот параметр.
aws.s3.region	Да	Регион, в котором находится ваш AWS S3 bucket. Пример: `us-west-1`.
aws.s3.access_key	Нет	Access key вашего IAM user. Если вы используете метод аутентификации на основе IAM user для доступа к AWS S3, вы должны указать этот параметр.
aws.s3.secret_key	Нет	Secret key вашего IAM user. Если вы используете метод аутентификации на основе IAM user для доступа к AWS S3, вы должны указать этот параметр.

Для получения информации о том, как выбрать метод аутентификации для доступа к AWS S3 и как настроить политику контроля доступа в AWS IAM Console, см. Параметры аутентификации для доступа к AWS S3.

S3-совместимая система хранения

Hive catalogs поддерживают S3-совместимые системы хранения начиная с версии v1.5.2.

Если вы выбрали S3-совместимую систему хранения, такую как MinIO, в качестве хранилища для вашего Hive cluster, настройте `StorageCredentialParams` следующим образом для обеспечения успешной интеграции:

```
"aws.s3.enable_ssl" = "false",
"aws.s3.enable_path_style_access" = "true",
"aws.s3.endpoint" = "<s3_endpoint>",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
aws.s3.enable_ssl	Да	Указывает, включить ли SSL-соединение. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`.
aws.s3.enable_path_style_access	Да	Указывает, включить ли доступ в стиле пути. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`. Для MinIO вы должны установить значение `true`. URL-адреса в стиле пути используют следующий формат: `https://s3.<region_code>.amazonaws.com/<bucket_name>/<key_name>`. Например, если вы создаете bucket с именем `DOC-EXAMPLE-BUCKET1` в регионе US West (Oregon) и хотите получить доступ к объекту `alice.jpg` в этом bucket, вы можете использовать следующий URL в стиле пути: `https://s3.us-west-2.amazonaws.com/DOC-EXAMPLE-BUCKET1/alice.jpg`.
aws.s3.endpoint	Да	Endpoint, который используется для подключения к вашей S3-совместимой системе хранения вместо AWS S3.
aws.s3.access_key	Да	Access key вашего IAM user.
aws.s3.secret_key	Да	Secret key вашего IAM user.

Microsoft Azure Storage

Hive catalogs поддерживают Microsoft Azure Storage начиная с версии v1.5.2.

Azure Blob Storage

Если вы выбрали Blob Storage в качестве хранилища для вашего Hive cluster, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Shared Key, настройте `StorageCredentialParams` следующим образом:

  ```
  "azure.blob.storage_account" = "<storage_account_name>",
  "azure.blob.shared_key" = "<storage_account_shared_key>"
  ```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.blob.storage_account	Да	Имя пользователя вашей учетной записи Blob Storage.
azure.blob.shared_key	Да	Shared key вашей учетной записи Blob Storage.

- Чтобы выбрать метод аутентификации SAS Token, настройте `StorageCredentialParams` следующим образом:

```
"azure.blob.storage_account" = "<storage_account_name>",
"azure.blob.container" = "<container_name>",
"azure.blob.sas_token" = "<storage_account_SAS_token>"
```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.blob.storage_account	Да	Имя пользователя вашей учетной записи Blob Storage.
azure.blob.container	Да	Имя blob container, в котором хранятся ваши данные.
azure.blob.sas_token	Да	SAS token, который используется для доступа к вашей учетной записи Blob Storage.

Azure Data Lake Storage Gen2

Если вы выбрали Data Lake Storage Gen2 в качестве хранилища для вашего Hive cluster, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Managed Identity, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.oauth2_use_managed_identity" = "true",
"azure.adls2.oauth2_tenant_id" = "<service_principal_tenant_id>",
"azure.adls2.oauth2_client_id" = "<service_client_id>"
```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls2.oauth2_use_managed_identity	Да	Указывает, включить ли метод аутентификации Managed Identity. Установите значение `true`.
azure.adls2.oauth2_tenant_id	Да	ID tenant, данные которого вы хотите получить.
azure.adls2.oauth2_client_id	Да	Client (application) ID managed identity.

- Чтобы выбрать метод аутентификации Shared Key, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.storage_account" = "<storage_account_name>",
"azure.adls2.shared_key" = "<storage_account_shared_key>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls2.storage_account	Да	Имя пользователя вашей учетной записи хранения Data Lake Storage Gen2.
azure.adls2.shared_key	Да	Shared key вашей учетной записи хранения Data Lake Storage Gen2.

- Чтобы выбрать метод аутентификации Service Principal, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.oauth2_client_id" = "<service_client_id>",
"azure.adls2.oauth2_client_secret" = "
<service_principal_client_secret>",
"azure.adls2.oauth2_client_endpoint" = "
<service_principal_client_endpoint>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls2.oauth2_client_id	Да	Client (application) ID service principal.

Параметр	Обязательный	Описание
azure.adls2.oauth2_client_secret	Да	Значение нового созданного client (application) secret.
azure.adls2.oauth2_client_endpoint	Да	OAuth 2.0 token endpoint (v1) service principal или application.

Azure Data Lake Storage Gen1

Если вы выбрали Data Lake Storage Gen1 в качестве хранилища для вашего Hive cluster, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Managed Service Identity, настройте `StorageCredentialParams` следующим образом:

  ```
  "azure.adls1.use_managed_service_identity" = "true"
  ```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls1.use_managed_service_identity	Да	Указывает, включить ли метод аутентификации Managed Service Identity. Установите значение `true`.

- Чтобы выбрать метод аутентификации Service Principal, настройте `StorageCredentialParams` следующим образом:

  ```
  "azure.adls1.oauth2_client_id" = "<application_client_id>",
  "azure.adls1.oauth2_credential" = "<application_client_credential>",
  "azure.adls1.oauth2_endpoint" = "<OAuth_2.0_authorization_endpoint_v2>"
  ```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls1.oauth2_client_id	Да	Client (application) ID service principal.
azure.adls1.oauth2_credential	Да	Значение нового созданного client (application) secret.

Параметр	Обязательный	Описание
azure.adls1.oauth2_endpoint	Да	OAuth 2.0 token endpoint (v1) service principal или application.

Google GCS

Hive catalogs поддерживают Google GCS начиная с версии v1.5.2.

Если вы выбрали Google GCS в качестве хранилища для вашего Hive cluster, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации на основе VM, настройте `StorageCredentialParams` следующим образом:

  ```
  "gcp.gcs.use_compute_engine_service_account" = "true"
  ```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.use_compute_engine_service_account	false	true	Указывает, использовать ли напрямую service account, привязанный к вашему Compute Engine.

- Чтобы выбрать метод аутентификации на основе service account, настройте `StorageCredentialParams` следующим образом:

  ```
  "gcp.gcs.service_account_email" = "<google_service_account_email>",
  "gcp.gcs.service_account_private_key_id" = "
  <google_service_private_key_id>",
  "gcp.gcs.service_account_private_key" = "<google_service_private_key>"
  ```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.service_account_email	""	"user@hello.iam.gserviceaccount.com"	Адрес электронной почты в JSON-файле, созданном при создании service account.
gcp.gcs.service_account_private_key_id	""	"61d257bd8479547cb3e04f0b9b6b9ca07af3b7ea"	ID private key в JSON-файле, созданном при создании service account.
gcp.gcs.service_account_private_key	""	"-----BEGIN PRIVATE KEY----xxxx-----END PRIVATE KEY----\n"	Private key в JSON-файле, созданном при создании service account.

- Чтобы выбрать метод аутентификации на основе impersonation, настройте `StorageCredentialParams` следующим образом:
  - Заставить VM instance имперсонировать service account:

    ```
    "gcp.gcs.use_compute_engine_service_account" = "true",
    "gcp.gcs.impersonation_service_account" = "
    <assumed_google_service_account_email>"
    ```

    В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.use_compute_engine_service_account	false	true	Указывает, использовать ли напрямую service account, привязанный к вашему Compute Engine.
gcp.gcs.impersonation_service_account	""	"hello"	Service account, который вы хотите имперсонировать.

  - Заставить service account (временно названный meta service account) имперсонировать другой service account (временно названный data service account):

```
"gcp.gcs.service_account_email" = "<google_service_account_email>",
"gcp.gcs.service_account_private_key_id" = "
<meta_google_service_account_email>",
"gcp.gcs.service_account_private_key" = "
<meta_google_service_account_email>",
"gcp.gcs.impersonation_service_account" = "
<data_google_service_account_email>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.service_account_email	""	"user@hello.iam.gserviceaccount.com"	Адрес электронной почты в JSON-файле, созданном при создании meta service account.
gcp.gcs.service_account_private_key_id	""	"61d257bd8479547cb3e04f0b9b6b9ca07af3b7ea"	ID private key в JSON-файле, созданном при создании meta service account.
gcp.gcs.service_account_private_key	""	"-----BEGIN PRIVATE KEY-----xxxx-----END PRIVATE KEY-----\n"	Private key в JSON-файле, созданном при создании meta service account.
gcp.gcs.impersonation_service_account	""	"hello"	Data service account, который вы хотите имперсонировать.

MetadataUpdateParams

Набор параметров о том, как Selena обновляет кэшированные метаданные Hive. Этот набор параметров необязателен.

Selena реализует политику автоматического асинхронного обновления по умолчанию.

В большинстве случаев вы можете игнорировать `MetadataUpdateParams` и не настраивать параметры политики в нем, потому что значения по умолчанию этих параметров уже обеспечивают готовую к использованию производительность.

Однако, если частота обновления данных в Hive высока, вы можете настроить эти параметры для дальнейшей оптимизации производительности автоматических

асинхронных обновлений.

> **ПРИМЕЧАНИЕ**
>
> В большинстве случаев, если ваши данные Hive обновляются с детализацией 1 час или меньше, частота обновления данных считается высокой.

Параметр	Обязательный	Описание
enable_metastore_cache	Нет	Указывает, кэширует ли Selena метаданные таблиц Hive. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`. Значение `true` включает кэш, а значение `false` отключает кэш.
enable_remote_file_cache	Нет	Указывает, кэширует ли Selena метаданные базовых файлов данных таблиц или partitions Hive. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`. Значение `true` включает кэш, а значение `false` отключает кэш.
metastore_cache_refresh_interval_sec	Нет	Интервал времени, с которым Selena асинхронно обновляет метаданные таблиц или partitions Hive, кэшированные в себе. Единица измерения: секунды. Значение по умолчанию: `60`, что составляет одну минуту. Начиная с версии v1.5.2, значение по умолчанию этого свойства изменено с `7200` на `60`.
remote_file_cache_refresh_interval_sec	Нет	Интервал времени, с которым Selena асинхронно обновляет метаданные базовых файлов данных таблиц или partitions Hive, кэшированные в себе. Единица измерения: секунды. Значение по умолчанию: `60`.
metastore_cache_ttl_sec	Нет	Интервал времени, с которым Selena автоматически отбрасывает метаданные таблиц или partitions Hive, кэшированные в себе. Единица измерения: секунды. Значение по умолчанию: `86400`, что составляет 24 часа.
remote_file_cache_ttl_sec	Нет	Интервал времени, с которым Selena автоматически отбрасывает метаданные базовых файлов данных таблиц или partitions Hive, кэшированные в себе. Единица измерения: секунды. Значение по умолчанию: `129600`, что составляет 36 часов.
enable_cache_list_names	Нет	Указывает, кэширует ли Selena имена partitions Hive. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`. Значение `true` включает кэш, а значение `false` отключает кэш.
remote_file_cache_memory_ratio	Нет	Максимальное соотношение использования памяти для кэша удаленных файлов. Значение по умолчанию: `0.1`, что составляет 10%. Поддерживается начиная с версии v1.5.2.

Примеры

Следующие примеры создают Hive catalog с именем `hive_catalog_hms` или `hive_catalog_glue`, в зависимости от типа metastore, который вы используете, для

запроса данных из вашего Hive cluster.

HDFS

Если вы используете HDFS в качестве хранилища, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
);
```

AWS S3

Аутентификация на основе instance profile

- Если вы используете Hive metastore в вашем Hive cluster, выполните команду, подобную приведенной ниже:

  ```
  CREATE EXTERNAL CATALOG hive_catalog_hms
  PROPERTIES
  (
      "type" = "hive",
      "hive.metastore.type" = "hive",
      "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
      "aws.s3.use_instance_profile" = "true",
      "aws.s3.region" = "us-west-2"
  );
  ```

- Если вы используете AWS Glue в вашем Amazon EMR Hive cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_glue
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2"
);
```

Аутентификация на основе assumed role

- Если вы используете Hive metastore в вашем Hive cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

- Если вы используете AWS Glue в вашем Amazon EMR Hive cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_glue
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_glue_role",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

Аутентификация на основе IAM user

- Если вы используете Hive metastore в вашем Hive cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_access_key>",
    "aws.s3.region" = "us-west-2"
);
```

- Если вы используете AWS Glue в вашем Amazon EMR Hive cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_glue
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "false",
    "aws.glue.access_key" = "<iam_user_access_key>",
    "aws.glue.secret_key" = "<iam_user_secret_key>",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_secret_key>",
    "aws.s3.region" = "us-west-2"
);
```

S3-совместимая система хранения

Используйте MinIO в качестве примера. Выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.enable_ssl" = "true",
    "aws.s3.enable_path_style_access" = "true",
    "aws.s3.endpoint" = "<s3_endpoint>",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_secret_key>"
);
```

Microsoft Azure Storage

Azure Blob Storage

- Если вы выберете метод аутентификации Shared Key, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.blob.storage_account" = "<blob_storage_account_name>",
    "azure.blob.shared_key" = "<blob_storage_account_shared_key>"
);
```

- Если вы выберете метод аутентификации SAS Token, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.blob.storage_account" = "<blob_storage_account_name>",
    "azure.blob.container" = "<blob_container_name>",
    "azure.blob.sas_token" = "<blob_storage_account_SAS_token>"
);
```

Azure Data Lake Storage Gen1

- Если вы выберете метод аутентификации Managed Service Identity, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls1.use_managed_service_identity" = "true"
);
```

- Если вы выберете метод аутентификации Service Principal, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls1.oauth2_client_id" = "<application_client_id>",
    "azure.adls1.oauth2_credential" = "
<application_client_credential>",
    "azure.adls1.oauth2_endpoint" = "
<OAuth_2.0_authorization_endpoint_v2>"
);
```

Azure Data Lake Storage Gen2

- Если вы выберете метод аутентификации Managed Identity, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls2.oauth2_use_managed_identity" = "true",
    "azure.adls2.oauth2_tenant_id" = "<service_principal_tenant_id>",
    "azure.adls2.oauth2_client_id" = "<service_client_id>"
);
```

- Если вы выберете метод аутентификации Shared Key, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls2.storage_account" = "<storage_account_name>",
    "azure.adls2.shared_key" = "<shared_key>"
);
```

- Если вы выберете метод аутентификации Service Principal, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls2.oauth2_client_id" = "<service_client_id>",
    "azure.adls2.oauth2_client_secret" = "
<service_principal_client_secret>",
    "azure.adls2.oauth2_client_endpoint" = "
<service_principal_client_endpoint>"
);
```

Google GCS

- Если вы выберете метод аутентификации на основе VM, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "gcp.gcs.use_compute_engine_service_account" = "true"
);
```

- Если вы выберете метод аутентификации на основе service account, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "gcp.gcs.service_account_email" = "<google_service_account_email>",
    "gcp.gcs.service_account_private_key_id" = "
<google_service_private_key_id>",
    "gcp.gcs.service_account_private_key" = "
<google_service_private_key>"
);
```

- Если вы выберете метод аутентификации на основе impersonation:
  - Если вы заставляете VM instance имперсонировать service account, выполните команду, подобную приведенной ниже:

    ```
    CREATE EXTERNAL CATALOG hive_catalog_hms
    PROPERTIES
    (
        "type" = "hive",
        "hive.metastore.type" = "hive",
        "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
        "gcp.gcs.use_compute_engine_service_account" = "true",
        "gcp.gcs.impersonation_service_account" = "
    <assumed_google_service_account_email>"
    );
    ```

  - Если вы заставляете service account имперсонировать другой service account, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hive_catalog_hms
PROPERTIES
(
    "type" = "hive",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "gcp.gcs.service_account_email" = "
<google_service_account_email>",
    "gcp.gcs.service_account_private_key_id" = "
<meta_google_service_account_email>",
    "gcp.gcs.service_account_private_key" = "
<meta_google_service_account_email>",
    "gcp.gcs.impersonation_service_account" = "
<data_google_service_account_email>"
);
```

Просмотр Hive catalogs

Вы можете использовать SHOW CATALOGS для запроса всех catalogs в текущем Selena cluster:

```
SHOW CATALOGS;
```

Вы также можете использовать SHOW CREATE CATALOG для запроса инструкции создания external catalog. Следующий пример запрашивает инструкцию создания Hive catalog с именем `hive_catalog_glue`:

```
SHOW CREATE CATALOG hive_catalog_glue;
```

Переключение на Hive Catalog и базу данных в нем

Вы можете использовать один из следующих методов для переключения на Hive catalog и базу данных в нем:

- Используйте SET CATALOG для указания Hive catalog в текущей сессии, а затем используйте USE для указания активной базы данных:

```
-- Переключитесь на указанный catalog в текущей сессии:
SET CATALOG <catalog_name>
-- Укажите активную базу данных в текущей сессии:
USE <db_name>
```

- Непосредственно используйте USE для переключения на Hive catalog и базу данных в нем:

```
USE <catalog_name>.<db_name>
```

Удаление Hive catalog

Вы можете использовать DROP CATALOG для удаления external catalog.

Следующий пример удаляет Hive catalog с именем `hive_catalog_glue`:

```
DROP Catalog hive_catalog_glue;
```

Просмотр схемы таблицы Hive

Вы можете использовать один из следующих синтаксисов для просмотра схемы таблицы Hive:

- Просмотр схемы

```
DESC[RIBE] <catalog_name>.<database_name>.<table_name>
```

- Просмотр схемы и расположения из инструкции CREATE

```
SHOW CREATE TABLE <catalog_name>.<database_name>.<table_name>
```

Запрос таблицы Hive

1. Используйте SHOW DATABASES для просмотра баз данных в вашем Hive cluster:

```
SHOW DATABASES FROM <catalog_name>
```

2. Переключитесь на Hive Catalog и базу данных в нем.

3. Используйте SELECT для запроса целевой таблицы в указанной базе данных:

```
SELECT count(*) FROM <table_name> LIMIT 10
```

Загрузка данных из Hive

Предположим, у вас есть OLAP-таблица с именем `olap_tbl`, вы можете преобразовать и загрузить данные следующим образом:

```
INSERT INTO default_catalog.olap_db.olap_tbl SELECT * FROM hive_table
```

Предоставление привилегий на таблицы и представления Hive

Вы можете использовать инструкцию GRANT для предоставления привилегий на все таблицы и представления в Hive catalog определенной роли. Синтаксис команды следующий:

```
GRANT SELECT ON ALL TABLES IN ALL DATABASES TO ROLE <role_name>
```

Например, используйте следующие команды для создания роли с именем `hive_role_table`, переключитесь на Hive catalog `hive_catalog`, а затем предоставьте роли `hive_role_table` привилегию запроса всех таблиц и представлений в Hive catalog `hive_catalog`:

```
-- Создайте роль с именем hive_role_table.
CREATE ROLE hive_role_table;

-- Переключитесь на Hive catalog hive_catalog.
SET CATALOG hive_catalog;

-- Предоставьте роли hive_role_table привилегию запроса всех таблиц и
представлений в Hive catalog hive_catalog.
GRANT SELECT ON ALL TABLES IN ALL DATABASES TO ROLE hive_role_table;
```

Создание базы данных Hive

Подобно внутреннему catalog Selena, если у вас есть привилегия CREATE DATABASE на Hive catalog, вы можете использовать инструкцию CREATE DATABASE для создания базы данных в этом Hive catalog. Эта функция поддерживается начиная с версии v1.5.2.

ПРИМЕЧАНИЕ

Вы можете предоставлять и отзывать привилегии с помощью GRANT и REVOKE.

ПРИМЕЧАНИЕ

Переключитесь на Hive catalog, а затем используйте следующую инструкцию для создания базы данных Hive в этом catalog:

```
CREATE DATABASE <database_name>
[PROPERTIES ("location" = "
<prefix>://<path_to_database>/<database_name.db>")]
```

Параметр `location` указывает путь к файлу, в котором вы хотите создать базу данных, который может быть либо в HDFS, либо в облачном хранилище.

- Когда вы используете Hive metastore в качестве metastore вашего Hive cluster, параметр `location` по умолчанию принимает значение `<warehouse_location>/<database_name.db>`, которое поддерживается Hive metastore, если вы не указываете этот параметр при создании базы данных.
- Когда вы используете AWS Glue в качестве metastore вашего Hive cluster, параметр `location` не имеет значения по умолчанию, и поэтому вы должны указать этот параметр при создании базы данных.

Значение `prefix` зависит от системы хранения, которую вы используете:

Система хранения	Значение `prefix`
HDFS	`hdfs`
Google GCS	`gs`
Azure Blob Storage	• Если ваша учетная запись хранения разрешает доступ по HTTP, `prefix` - это `wasb`. • Если ваша учетная запись хранения разрешает доступ по HTTPS, `prefix` - это `wasbs`.
Azure Data Lake Storage Gen1	`adl`
Azure Data Lake Storage Gen2	• Если ваша учетная запись хранения разрешает доступ по HTTP, `prefix` - это `abfs`. • Если ваша учетная запись хранения разрешает доступ по HTTPS, `prefix` - это `abfss`.
AWS S3 или другое S3-совместимое хранилище (например, MinIO)	`s3`

Удаление базы данных Hive

Подобно внутренним базам данных Selena, если у вас есть привилегия DROP на базу данных Hive, вы можете использовать инструкцию DROP DATABASE для удаления этой базы данных Hive. Эта функция поддерживается начиная с версии v1.5.2. Вы можете удалять только пустые базы данных.

ПРИМЕЧАНИЕ

Вы можете предоставлять и отзывать привилегии с помощью GRANT и REVOKE.

Когда вы удаляете базу данных Hive, путь к файлу базы данных в вашем HDFS cluster или облачном хранилище не будет удален вместе с базой данных.

Переключитесь на Hive catalog, а затем используйте следующую инструкцию для удаления базы данных Hive в этом catalog:

```
DROP DATABASE <database_name>
```

Создание таблицы Hive

Подобно внутренним базам данных Selena, если у вас есть привилегия CREATE TABLE на базу данных Hive, вы можете использовать инструкции CREATE TABLE, CREATE TABLE AS SELECT или CREATE TABLE LIKE для создания управляемой таблицы в этой базе данных Hive.

Эта функция поддерживается с версии v1.5.2, в которой Selena поддерживает только создание таблиц Hive в формате Parquet. Начиная с версии v1.5.2, Selena также поддерживает создание таблиц Hive в форматах ORC и Textfile.

ПРИМЕЧАНИЕ

- Вы можете предоставлять и отзывать привилегии с помощью GRANT и REVOKE.
- Hive catalogs поддерживают CREATE TABLE LIKE начиная с версии v1.5.2.

Переключитесь на Hive catalog и базу данных в нем, а затем используйте следующий синтаксис для создания управляемой таблицы Hive в этой базе данных.

Синтаксис

```
CREATE TABLE [IF NOT EXISTS] [database.]table_name
(column_definition1[, column_definition2, ...
partition_column_definition1,partition_column_definition2...])
[partition_desc]
[PROPERTIES ("key" = "value", ...)]
[AS SELECT query]
[LIKE [database.]<source_table_name>]
```

Параметры

column_definition

Синтаксис `column_definition` следующий:

```
col_name col_type [COMMENT 'comment']
```

В следующей таблице описаны параметры.

Параметр	Описание
col_name	Имя столбца.
col_type	Тип данных столбца. Поддерживаются следующие типы данных: TINYINT, SMALLINT, INT, BIGINT, FLOAT, DOUBLE, DECIMAL, DATE, DATETIME, CHAR, VARCHAR[(length)], ARRAY, MAP и STRUCT. Типы данных LARGEINT, HLL и BITMAP не поддерживаются.

> **ПРИМЕЧАНИЕ**
>
> Все не-partition столбцы должны использовать `NULL` в качестве значения по умолчанию. Это означает, что вы должны указать `DEFAULT "NULL"` для каждого из не-partition столбцов в инструкции создания таблицы. Кроме того, partition столбцы должны быть определены после не-partition столбцов и не могут использовать `NULL` в качестве значения по умолчанию.

partition_desc

Синтаксис `partition_desc` следующий:

```
PARTITION BY (par_col1[, par_col2...])
```

В настоящее время Selena поддерживает только identity transforms, что означает, что Selena создает partition для каждого уникального значения partition.

> **ПРИМЕЧАНИЕ**
>
> Partition столбцы должны быть определены после не-partition столбцов. Partition столбцы поддерживают все типы данных, за исключением FLOAT, DOUBLE, DECIMAL и DATETIME, и не могут использовать `NULL` в качестве значения по умолчанию. Кроме того, последовательность partition столбцов, объявленных в `partition_desc`, должна соответствовать последовательности столбцов, определенных в `column_definition`.

PROPERTIES

Вы можете указать атрибуты таблицы в формате `"key" = "value"` в `properties`.

В следующей таблице описано несколько ключевых свойств.

Свойство	Описание
location	Путь к файлу, в котором вы хотите создать управляемую таблицу. Когда вы используете HMS в качестве metastore, вам не нужно указывать параметр `location`, потому что Selena создаст таблицу в пути к файлу по умолчанию текущего Hive catalog. Когда вы используете AWS Glue в качестве службы метаданных: • Если вы указали параметр `location` для базы данных, в которой вы хотите создать таблицу, вам не нужно указывать параметр `location` для таблицы. Таким образом, таблица по умолчанию принимает путь к файлу базы данных, к которой она принадлежит. • Если вы не указали `location` для базы данных, в которой вы хотите создать таблицу, вы должны указать параметр `location` для таблицы.
file_format	Формат файла управляемой таблицы. Поддерживаемые форматы файлов: Parquet, ORC и Textfile. Форматы ORC и Textfile поддерживаются начиная с версии v1.5.2. Допустимые значения: `parquet`, `orc` и `textfile`. Значение по умолчанию: `parquet`.

Свойство	Описание
compression_codec	Алгоритм сжатия, используемый для управляемой таблицы.

Примеры

Следующие DDL используют формат файла Parquet по умолчанию в качестве примера.

1. Создайте не-партиционированную таблицу с именем `unpartition_tbl`. Таблица состоит из двух столбцов, `id` и `score`, как показано ниже:

```
CREATE TABLE unpartition_tbl
(
    id int,
    score double
);
```

2. Создайте партиционированную таблицу с именем `partition_tbl_1`. Таблица состоит из трех столбцов, `action`, `id` и `dt`, из которых `id` и `dt` определены как partition столбцы, как показано ниже:

```
CREATE TABLE partition_tbl_1
(
    action varchar(20),
    id int,
    dt date
)
PARTITION BY (id,dt);
```

3. Запросите существующую таблицу с именем `partition_tbl_1` и создайте партиционированную таблицу с именем `partition_tbl_2` на основе результата запроса `partition_tbl_1`. Для `partition_tbl_2` `id` и `dt` определены как partition столбцы, как показано ниже:

```
CREATE TABLE partition_tbl_2
PARTITION BY (k1, k2)
AS SELECT * from partition_tbl_1;
```

Загрузка данных в таблицу Hive

Подобно внутренним таблицам Selena, если у вас есть привилегия INSERT на таблицу Hive (которая может быть управляемой таблицей или внешней таблицей), вы можете использовать инструкцию INSERT для загрузки данных из таблицы Selena в эту таблицу Hive.

Эта функция поддерживается с версии v1.5.2, в которой данные могут быть загружены только в таблицы Hive в формате Parquet. Начиная с версии v1.5.2, Selena также поддерживает загрузку данных в таблицы Hive в форматах ORC и Textfile.

Обратите внимание, что загрузка данных во внешние таблицы по умолчанию отключена. Чтобы загружать данные во внешние таблицы, вы должны установить системную переменную `ENABLE_WRITE_HIVE_EXTERNAL_TABLE` в `true`.

> **ПРИМЕЧАНИЕ**
>
> - Вы можете предоставлять и отзывать привилегии с помощью GRANT и REVOKE.
> - Вы можете использовать свойство таблицы `compression_codec` или системную переменную `connector_sink_compression_codec` для указания алгоритма сжатия, используемого при загрузке данных в таблицы Hive. Selena будет приоритетно использовать compression codec, указанный в свойстве таблицы.

Переключитесь на Hive catalog и базу данных в нем, а затем используйте следующий синтаксис для загрузки данных из таблицы Selena в таблицу Hive в формате Parquet в этой базе данных.

Синтаксис

```
INSERT {INTO | OVERWRITE} <table_name>
[ (column_name [, ...]) ]
{ VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }

-- Если вы хотите загрузить данные в указанные partitions, используйте
следующий синтаксис:
INSERT {INTO | OVERWRITE} <table_name>
PARTITION (par_col1=<value> [, par_col2=<value>...])
{ VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }
```

> **ПРИМЕЧАНИЕ**
>
> Partition столбцы не допускают значения `NULL`. Поэтому вы должны убедиться, что в partition столбцы таблицы Hive не загружаются пустые значения.

Параметры

Параметр	Описание
INTO	Для добавления данных из таблицы Selena в таблицу Hive.
OVERWRITE	Для перезаписи существующих данных таблицы Hive данными из таблицы Selena.
column_name	Имя целевого столбца, в который вы хотите загрузить данные. Вы можете указать один или несколько столбцов. Если вы указываете несколько столбцов, разделите их запятыми (`,`). Вы можете указывать только столбцы, которые фактически существуют в таблице Hive, и целевые столбцы, которые вы указываете, должны включать partition столбцы таблицы Hive. Целевые столбцы, которые вы указываете, сопоставляются один к одному в последовательности со столбцами таблицы Selena, независимо от того, какие имена целевых столбцов. Если целевые столбцы не указаны, данные загружаются во все столбцы таблицы Hive. Если не-partition столбец таблицы Selena не может быть сопоставлен ни с одним столбцом таблицы Hive, Selena записывает значение по умолчанию `NULL` в столбец таблицы Hive. Если инструкция INSERT содержит инструкцию запроса, типы возвращаемых столбцов которой отличаются от типов данных целевых столбцов, Selena выполняет неявное преобразование несоответствующих столбцов. Если преобразование не удается, будет возвращена ошибка синтаксического анализа.
expression	Выражение, которое присваивает значения целевому столбцу.
DEFAULT	Присваивает значение по умолчанию целевому столбцу.
query	Инструкция запроса, результат которой будет загружен в таблицу Hive. Это может быть любая SQL-инструкция, поддерживаемая Selena.
PART	Partitions, в которые вы хотите загрузить данные. Вы должны указать все partition столбцы таблицы Hive в этом свойстве. Partition столбцы, которые вы указываете в этом свойстве, могут быть в другой

Параметр	Описание
ITION	последовательности, чем partition столбцы, которые вы определили в инструкции создания таблицы. Если вы указываете это свойство, вы не можете указывать свойство `column_name`.

Примеры

Следующие DML используют формат файла Parquet по умолчанию в качестве примера.

1. Вставьте три строки данных в таблицу `partition_tbl_1`:

```
INSERT INTO partition_tbl_1
VALUES
    ("buy", 1, "2023-09-01"),
    ("sell", 2, "2023-09-02"),
    ("buy", 3, "2023-09-03");
```

2. Вставьте результат запроса SELECT, который содержит простые вычисления, в таблицу `partition_tbl_1`:

```
INSERT INTO partition_tbl_1 (id, action, dt) SELECT 1+1, 'buy', '2023-
09-03';
```

3. Вставьте результат запроса SELECT, который читает данные из таблицы `partition_tbl_1`, в ту же таблицу:

```
INSERT INTO partition_tbl_1 SELECT 'buy', 1, date_add(dt, INTERVAL 2
DAY)
FROM partition_tbl_1
WHERE id=1;
```

4. Вставьте результат запроса SELECT в partitions, которые соответствуют двум условиям, `dt='2023-09-01'` и `id=1`, таблицы `partition_tbl_2`:

```
INSERT INTO partition_tbl_2 SELECT 'order', 1, '2023-09-01';
```

Или

```
INSERT INTO partition_tbl_2 partition(dt='2023-09-01',id=1) SELECT
'order';
```

5. Перезаписать все значения столбца `action` в partitions, которые соответствуют двум условиям, `dt='2023-09-01'` и `id=1`, таблицы `partition_tbl_1` значением `close`:

```
INSERT OVERWRITE partition_tbl_1 SELECT 'close', 1, '2023-09-01';
```

Или

```
INSERT OVERWRITE partition_tbl_1 partition(dt='2023-09-01',id=1) SELECT
'close';
```

Удаление таблицы Hive

Подобно внутренним таблицам Selena, если у вас есть привилегия DROP на таблицу Hive, вы можете использовать инструкцию DROP TABLE для удаления этой таблицы Hive. Эта функция поддерживается начиная с версии v1.5.2. Обратите внимание, что в настоящее время Selena поддерживает удаление только управляемых таблиц Hive.

> **ПРИМЕЧАНИЕ**
>
> Вы можете предоставлять и отзывать привилегии с помощью GRANT и REVOKE.

Когда вы удаляете таблицу Hive, вы должны указать ключевое слово `FORCE` в инструкции DROP TABLE. После завершения операции путь к файлу таблицы сохраняется, но все данные таблицы в вашем HDFS cluster или облачном хранилище удаляются вместе с таблицей. Будьте осторожны при выполнении этой операции для удаления таблицы Hive.

Переключитесь на Hive catalog и базу данных в нем, а затем используйте следующую инструкцию для удаления таблицы Hive в этой базе данных.

```
DROP TABLE <table_name> FORCE
```

Ручное или автоматическое обновление кэша метаданных

Ручное обновление

По умолчанию Selena кэширует метаданные Hive и автоматически обновляет метаданные в асинхронном режиме для повышения производительности. Кроме того, после внесения некоторых изменений схемы или обновлений таблицы в таблицу Hive, вы также можете использовать REFRESH EXTERNAL TABLE для ручного обновления ее метаданных, тем самым гарантируя, что Selena сможет получить актуальные метаданные при первой возможности и сгенерировать соответствующие планы выполнения:

```
REFRESH EXTERNAL TABLE <table_name> [PARTITION ('partition_name', ...)]
```

Вам необходимо вручную обновлять метаданные в следующих ситуациях:

- Файл данных в существующем partition изменяется, например, путем выполнения команды `INSERT OVERWRITE ... PARTITION ...`.
- Изменения схемы вносятся в таблицу Hive.
- Существующая таблица Hive удаляется с помощью инструкции DROP, и создается новая таблица Hive с тем же именем, что и удаленная таблица Hive.
- Вы указали `"enable_cache_list_names" = "true"` в `PROPERTIES` при создании вашего Hive catalog, и вы хотите запросить новые partitions, которые вы только что создали в вашем Hive cluster.

> **ПРИМЕЧАНИЕ**
>
> Начиная с версии v1.5.2, Selena предоставляет функцию периодического обновления кэша метаданных Hive. Для получения дополнительной информации см. ниже раздел "Периодическое обновление кэша метаданных" этой темы. После включения этой функции Selena обновляет кэш метаданных Hive каждые 10 минут по умолчанию. Поэтому ручное обновление не требуется в большинстве случаев. Вам нужно выполнить ручное обновление только тогда, когда вы хотите немедленно запросить новые partitions после создания новых partitions в вашем Hive cluster.

Обратите внимание, что REFRESH EXTERNAL TABLE обновляет только таблицы и partitions, кэшированные в ваших FE.

Периодическое обновление кэша метаданных

Начиная с версии v1.5.2, Selena может периодически обновлять кэшированные метаданные часто используемых Hive catalogs для восприятия изменений данных. Вы можете настроить обновление кэша метаданных Hive с помощью следующих параметров FE:

Элемент конфигурации	По умолчанию	Описание
enable_background_refresh_connector_metadata	`true` в v1.5.2 `false` в v1.5.2	Включить ли периодическое обновление кэша метаданных Hive. После включения Selena опрашивает metastore (Hive Metastore или AWS Glue) вашего Hive cluster и обновляет кэшированные метаданные часто используемых Hive catalogs для восприятия изменений данных. `true` указывает на включение обновления кэша метаданных Hive, а `false` указывает на отключение. Это динамический параметр FE. Вы можете изменить его с помощью команды ADMIN SET FRONTEND CONFIG.
background_refresh_metadata_interval_millis	`600000` (10 минут)	Интервал между двумя последовательными обновлениями кэша метаданных Hive. Единица измерения: миллисекунды. Это динамический параметр FE. Вы можете изменить его с помощью команды ADMIN SET FRONTEND CONFIG.
background_refresh_metadata_time_secs_since_last_access_secs	`86400` (24 часа)	Время истечения задачи обновления кэша метаданных Hive. Для Hive catalog, к которому был осуществлен доступ, если к нему не было доступа более указанного времени, Selena прекращает обновление его кэшированных метаданных. Для Hive catalog, к которому не было доступа, Selena не будет обновлять его кэшированные метаданные. Единица измерения: секунды. Это динамический параметр FE. Вы можете изменить его с помощью команды ADMIN SET FRONTEND CONFIG.

Использование функции периодического обновления кэша метаданных Hive и политики автоматического асинхронного обновления метаданных вместе значительно ускоряет доступ к данным, снижает нагрузку на чтение из внешних источников данных и повышает производительность запросов.

Приложение: Понимание автоматического асинхронного обновления метаданных

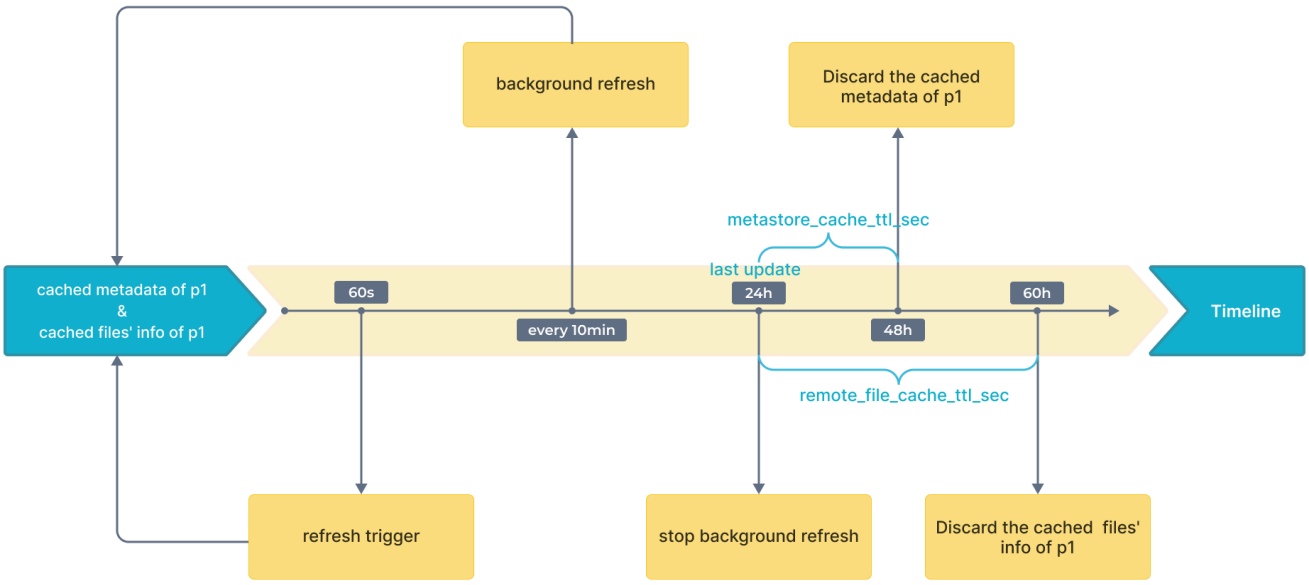
Автоматическое асинхронное обновление - это политика по умолчанию, которую Selena использует для обновления метаданных в Hive catalogs.

По умолчанию (а именно, когда параметры `enable_metastore_cache` и `enable_remote_file_cache` оба установлены в `true`), если запрос попадает в partition таблицы Hive, Selena автоматически кэширует метаданные partition и метаданные базовых файлов данных partition. Кэшированные метаданные обновляются с использованием политики ленивого обновления.

Например, есть таблица Hive с именем `table2`, которая имеет четыре partitions: `p1`, `p2`, `p3` и `p4`. Запрос попадает в `p1`, и Selena кэширует метаданные `p1` и метаданные базовых файлов данных `p1`. Предположим, что интервалы времени по умолчанию для обновления и отбрасывания кэшированных метаданных следующие:

- Интервал времени (указанный параметром `metastore_cache_refresh_interval_sec`) для асинхронного обновления кэшированных метаданных `p1` составляет 60 секунд.
- Интервал времени (указанный параметром `remote_file_cache_refresh_interval_sec`) для асинхронного обновления кэшированных метаданных базовых файлов данных `p1` составляет 60 секунд.
- Интервал времени (указанный параметром `metastore_cache_ttl_sec`) для автоматического отбрасывания кэшированных метаданных `p1` составляет 24 часа.
- Интервал времени (указанный параметром `remote_file_cache_ttl_sec`) для автоматического отбрасывания кэшированных метаданных базовых файлов данных `p1` составляет 36 часов.

На следующем рисунке показаны интервалы времени на временной шкале для более легкого понимания.



Затем Selena обновляет или отбрасывает метаданные в соответствии со следующими правилами:

- Если другой запрос снова попадает в `p1` и текущее время с момента последнего обновления составляет менее 60 секунд, Selena не обновляет кэшированные метаданные `p1` или кэшированные метаданные базовых файлов данных `p1`.
- Если другой запрос снова попадает в `p1` и текущее время с момента последнего обновления составляет более 60 секунд, Selena обновляет кэшированные метаданные `p1` и кэшированные метаданные базовых файлов данных `p1`.
- Если к таблице был осуществлен доступ в течение 24 часов, соответствующий кэш будет обновляться каждые 10 минут в фоновом режиме.
- Если к `p1` не было доступа в течение 24 часов с момента последнего обновления, Selena отбрасывает кэшированные метаданные `p1`. Метаданные будут кэшированы при следующем запросе.
- Если к `p1` не было доступа в течение 36 часов с момента последнего обновления, Selena отбрасывает кэшированные метаданные базовых файлов данных `p1`. Метаданные будут кэшированы при следующем запросе.

5.7.4 Iceberg catalog

> **ПОДСКАЗКА**
>
> Для быстрого старта с Selena см. раздел Быстрый старт.

Iceberg catalog — это тип внешнего каталога, который поддерживается Selena начиная с версии v1.5.2. С Iceberg catalogs вы можете:

- Напрямую запрашивать данные, хранящиеся в Iceberg, без необходимости вручную создавать таблицы.
- Использовать INSERT INTO или асинхронные материализованные представления (которые поддерживаются с v1.5.2) для обработки данных, хранящихся в Iceberg, и загрузки данных в Selena.
- Выполнять операции в Selena для создания или удаления баз данных и таблиц Iceberg, или записывать данные из таблиц Selena в таблицы Iceberg формата Parquet с помощью INSERT INTO (эта функция поддерживается с v1.5.2).

Чтобы обеспечить успешное выполнение SQL-запросов в вашем Iceberg cluster, ваш Selena cluster должен иметь доступ к системе хранения и metastore вашего Iceberg cluster. Selena поддерживает следующие системы хранения и metastore:

- Распределенная файловая система (HDFS) или объектное хранилище, такое как AWS S3, Microsoft Azure Storage, Google GCS или другая S3-совместимая система хранения (например, MinIO)

- Metastore, такой как Hive metastore, AWS Glue или Tabular

> **ПРИМЕЧАНИЕ**
>
> - Если вы выбираете AWS S3 в качестве хранилища, вы можете использовать HMS или AWS Glue в качестве metastore. Если вы выбираете любую другую систему хранения, вы можете использовать только HMS в качестве metastore.
> - Если вы выбираете Tabular в качестве metastore, вам нужно использовать Iceberg REST catalog.

Примечания по использованию

Обратите внимание на следующие моменты при использовании Selena для запроса данных из Iceberg:

Формат файла	Формат сжатия	Версия таблицы Iceberg
Parquet	SNAPPY, LZ4, ZSTD, GZIP и NO_COMPRESSION	• v1 tables: поддерживается. • v2 tables: поддерживается с Selena v1.5.2, при этом запросы к этим v2 tables поддерживают position deletes. В более поздних версиях запросы к v2 tables также поддерживают equality deletes.
ORC	ZLIB, SNAPPY, LZO, LZ4, ZSTD и NO_COMPRESSION	• v1 tables: поддерживается. • v2 tables: поддерживается с Selena v1.5.2, при этом запросы к этим v2 tables поддерживают position deletes. В более поздних версиях запросы к v2 tables также поддерживают equality deletes.

Подготовка к интеграции

Перед созданием Iceberg catalog убедитесь, что ваш Selena cluster может интегрироваться с системой хранения и metastore вашего Iceberg cluster.

Хранилище

Выберите вкладку, соответствующую вашему типу хранилища:

AWS S3 HDFS

Если ваш Iceberg cluster использует AWS S3 в качестве хранилища или AWS Glue в качестве metastore, выберите подходящий метод аутентификации и выполните необходимые подготовительные действия, чтобы убедиться, что ваш Selena cluster может получить доступ к соответствующим облачным ресурсам AWS.

Рекомендуются следующие методы аутентификации:

- Instance profile
- Assumed role
- IAM user

Из вышеупомянутых трех методов аутентификации instance profile является наиболее широко используемым.

Для получения дополнительной информации см. Подготовка к аутентификации в AWS IAM.

Создание Iceberg catalog

Синтаксис

```
CREATE EXTERNAL CATALOG <catalog_name>
[COMMENT <comment>]
PROPERTIES
(
    "type" = "iceberg",
    [SecurityParams],
    MetastoreParams,
    StorageCredentialParams,
    MetadataRelatedParams
)
```

Параметры

catalog_name

Имя Iceberg catalog. Соглашения об именовании следующие:

- Имя может содержать буквы, цифры (0-9) и символы подчеркивания (_). Оно должно начинаться с буквы.
- Имя чувствительно к регистру и не может превышать 1023 символа.

comment

Описание Iceberg catalog. Этот параметр является опциональным.

type

Тип вашего источника данных. Установите значение `iceberg`.

SecurityParams

Параметр(ы) о том, как Selena управляет доступом к данным каталога.

Подробные инструкции по управлению доступом к данным для Iceberg-каталогов см. в разделе Настройка безопасности для Iceberg REST Catalog.

catalog.access.control

Политика контроля доступа к данным. Допустимые значения:

- `native` (по умолчанию): используется встроенная система контроля доступа к данным Selena.
- `allowall`: все проверки доступа к данным делегируются самому Catalog.
- `ranger`: проверки доступа к данным делегируются Apache Ranger.

MetastoreParams

Набор параметров о том, как Selena интегрируется с metastore вашего источника данных. Выберите вкладку, соответствующую вашему типу metastore:

Hive metastore | AWS Glue | REST | JDBC

Hive metastore

Если вы выбираете Hive metastore в качестве metastore вашего источника данных, настройте `MetastoreParams` следующим образом:

```
"iceberg.catalog.type" = "hive",
"hive.metastore.uris" = "<hive_metastore_uri>"
```

> **ПРИМЕЧАНИЕ**
>
> Перед запросом данных Iceberg вы должны добавить сопоставление между именами хостов и IP-адресами узлов вашего Hive metastore в путь `/etc/hosts`. В противном случае Selena может не получить доступ к вашему Hive metastore при запуске запроса.

В следующей таблице описан параметр, который необходимо настроить в `MetastoreParams`.

- `iceberg.catalog.type`

  - Обязательный: Да
  - Описание: Тип metastore, который вы используете для вашего Iceberg cluster. Установите значение `hive`.
- `hive.metastore.uris`
  - Обязательный: Да
  - Описание: URI вашего Hive metastore. Формат: `thrift://<metastore_IP_address>:<metastore_port>`.
    Если для вашего Hive metastore включена высокая доступность (HA), вы можете указать несколько URI metastore и разделить их запятыми (`,`), например, `"thrift://<metastore_IP_address_1>:<metastore_port_1>,thrift://<metastore_IP_address_2>:<metastore_port_2>,thrift://<metastore_IP_address_3>:<metastore_port_3>"`.

`StorageCredentialParams`

Набор параметров о том, как Selena интегрируется с вашей системой хранения. Этот набор параметров является опциональным.

Обратите внимание на следующие моменты:

- Если вы используете HDFS в качестве хранилища, вам не нужно настраивать `StorageCredentialParams`, и вы можете пропустить этот раздел. Если вы используете AWS S3, другую S3-совместимую систему хранения, Microsoft Azure Storage или Google GCS в качестве хранилища, вы должны настроить `StorageCredentialParams`.

- Если вы используете Tabular в качестве metastore, вам не нужно настраивать `StorageCredentialParams`, и вы можете пропустить этот раздел. Если вы используете HMS или AWS Glue в качестве metastore, вы должны настроить `StorageCredentialParams`.

Выберите вкладку, соответствующую вашему типу хранилища:

AWS S3 | **HDFS** | **MinIO** | **Microsoft Azure Blob Storage** | **Google GCS**

AWS S3

Если вы выбираете AWS S3 в качестве хранилища для вашего Iceberg cluster, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации на основе instance profile, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.region" = "<aws_s3_region>"
```

- Чтобы выбрать метод аутентификации на основе assumed role, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.iam_role_arn" = "<iam_role_arn>",
"aws.s3.region" = "<aws_s3_region>"
```

- Чтобы выбрать метод аутентификации на основе IAM user, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "false",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>",
"aws.s3.region" = "<aws_s3_region>"
```

- Чтобы выбрать vended credential (поддерживается с v2.0.0) с REST catalog, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.region" = "<aws_s3_region>"
```

`StorageCredentialParams` для AWS S3:

aws.s3.use_instance_profile

- Обязательный: Да

- Описание: Указывает, следует ли включить метод аутентификации на основе instance profile и метод аутентификации на основе assumed role. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`.

aws.s3.iam_role_arn

- Обязательный: Нет
- Описание: ARN роли IAM, которая имеет привилегии в вашем AWS S3 bucket. Если вы используете метод аутентификации на основе assumed role для доступа к AWS S3, вы должны указать этот параметр.

aws.s3.region

- Обязательный: Да
- Описание: Регион, в котором находится ваш AWS S3 bucket. Пример: `us-west-1`.

aws.s3.access_key

- Обязательный: Нет
- Описание: Ключ доступа вашего пользователя IAM. Если вы используете метод аутентификации на основе IAM user для доступа к AWS S3, вы должны указать этот параметр.

aws.s3.secret_key

- Обязательный: Нет
- Описание: Секретный ключ вашего пользователя IAM. Если вы используете метод аутентификации на основе IAM user для доступа к AWS S3, вы должны указать этот параметр.

Для получения информации о том, как выбрать метод аутентификации для доступа к AWS S3 и как настроить политику контроля доступа в консоли AWS IAM, см. Параметры аутентификации для доступа к AWS S3.

MetadataRelatedParams

Набор параметров о кэше метаданных Iceberg в Selena. Этот набор параметров является опциональным.

С v1.5.2 Selena поддерживает стратегию периодического обновления метаданных. В большинстве случаев вы можете игнорировать приведенные ниже параметры и не настраивать параметры политики, поскольку значения по умолчанию этих параметров уже

обеспечивают производительность из коробки. Вы можете настроить режим парсинга метаданных Iceberg с помощью системной переменной `plan_mode`.

Параметр	По умолчанию	Описание
enable_iceberg_metadata_cache	true	Кэшировать ли метаданные, связанные с Iceberg, включая Table Cache, Partition Name Cache, а также Data File Cache и Delete Data File Cache в Manifest.
iceberg_manifest_cache_with_column_statistics	false	Кэшировать ли статистику столбцов.
refresh_iceberg_manifest_min_length	2 * 1024 * 1024	Минимальная длина файла Manifest, которая запускает обновление Data File Cache.
iceberg_data_file_cache_memory_usage_ratio	0.1	Максимальное соотношение использования памяти для кэша Manifest файлов данных. Поддерживается с v1.5.2.
iceberg_delete_file_cache_memory_usage_ratio	0.1	Максимальное соотношение использования памяти для кэша Manifest файлов удаления. Поддерживается с v1.5.2.
iceberg_table_cache_refresh_interval_sec	60	Интервал (в секундах), с которым запускается асинхронное обновление кэша таблиц Iceberg. Поддерживается с v1.5.2.

Начиная с v1.5.2, Selena может получать статистику таблиц Iceberg, читая метаданные Iceberg через установку следующих параметров, без активного запуска сбора статистики таблиц Iceberg.

Параметр	По умолчанию	Описание
enable_get_stats_from_external_metadata	false	Получать ли статистику из метаданных Iceberg. Когда этот параметр установлен в `true`, вы можете дополнительно контролировать, какой тип статистики собирать, через системную переменную `enable_get_stats_from_external_metadata`.

Примеры

Следующие примеры создают Iceberg catalog с именем `iceberg_catalog_hms` или `iceberg_catalog_glue`, в зависимости от типа metastore, который вы используете, для запроса данных из вашего Iceberg cluster. Выберите вкладку, соответствующую вашему типу хранилища:

AWS S3 | HDFS | MinIO | Microsoft Azure Blob Storage | Google GCS

AWS S3

Если вы выбираете учетные данные на основе instance profile

- Если вы используете Hive metastore в вашем Iceberg cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG iceberg_catalog_hms
PROPERTIES
(
    "type" = "iceberg",
    "iceberg.catalog.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2"
);
```

- Если вы используете AWS Glue в вашем Amazon EMR Iceberg cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG iceberg_catalog_glue
PROPERTIES
(
    "type" = "iceberg",
    "iceberg.catalog.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2"
);
```

Если вы выбираете учетные данные на основе assumed role

- Если вы используете Hive metastore в вашем Iceberg cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG iceberg_catalog_hms
PROPERTIES
(
    "type" = "iceberg",
    "iceberg.catalog.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

- Если вы используете AWS Glue в вашем Amazon EMR Iceberg cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG iceberg_catalog_glue
PROPERTIES
(
    "type" = "iceberg",
    "iceberg.catalog.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_glue_role",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

Если вы выбираете учетные данные на основе IAM user

- Если вы используете Hive metastore в вашем Iceberg cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG iceberg_catalog_hms
PROPERTIES
(
    "type" = "iceberg",
    "iceberg.catalog.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_access_key>",
    "aws.s3.region" = "us-west-2"
);
```

- Если вы используете AWS Glue в вашем Amazon EMR Iceberg cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG iceberg_catalog_glue
PROPERTIES
(
    "type" = "iceberg",
    "iceberg.catalog.type" = "glue",
    "aws.glue.use_instance_profile" = "false",
    "aws.glue.access_key" = "<iam_user_access_key>",
    "aws.glue.secret_key" = "<iam_user_secret_key>",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_secret_key>",
    "aws.s3.region" = "us-west-2"
);
```

Если вы выбираете vended credential

Если вы выбираете REST catalog с vended credential, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG polaris_s3
PROPERTIES
(
    "type" = "iceberg",
    "iceberg.catalog.uri" = "http://xxx:xxx/api/catalog",
    "iceberg.catalog.type" = "rest",
    "iceberg.catalog.rest.nested-namespace-enabled"="true",
    "iceberg.catalog.security" = "oauth2",
    "iceberg.catalog.oauth2.credential" = "xxxxx:xxxx",
    "iceberg.catalog.oauth2.scope"='PRINCIPAL_ROLE:ALL',
    "iceberg.catalog.warehouse" = "iceberg_catalog",
    "aws.s3.region" = "us-west-2"
);
```

Использование вашего каталога

Просмотр Iceberg catalogs

Вы можете использовать SHOW CATALOGS для запроса всех каталогов в текущем Selena cluster:

```
SHOW CATALOGS;
```

Вы также можете использовать SHOW CREATE CATALOG для запроса оператора создания внешнего каталога. Следующий пример запрашивает оператор создания Iceberg catalog с именем `iceberg_catalog_glue`:

```
SHOW CREATE CATALOG iceberg_catalog_glue;
```

Переключение на Iceberg Catalog и базу данных в нем

Вы можете использовать один из следующих методов для переключения на Iceberg catalog и базу данных в нем:

- Используйте SET CATALOG для указания Iceberg catalog в текущей сессии, а затем используйте USE для указания активной базы данных:

```
-- Переключиться на указанный каталог в текущей сессии:
SET CATALOG <catalog_name>
-- Указать активную базу данных в текущей сессии:
USE <db_name>
```

- Непосредственно используйте USE для переключения на Iceberg catalog и базу данных в нем:

```
USE <catalog_name>.<db_name>
```

Удаление Iceberg catalog

Вы можете использовать DROP CATALOG для удаления внешнего каталога.

Следующий пример удаляет Iceberg catalog с именем `iceberg_catalog_glue`:

```
DROP Catalog iceberg_catalog_glue;
```

Просмотр схемы таблицы Iceberg

Вы можете использовать один из следующих синтаксисов для просмотра схемы таблицы Iceberg:

- Просмотр схемы

```
DESC[RIBE] <catalog_name>.<database_name>.<table_name>
```

- Просмотр схемы и местоположения из оператора CREATE

```
SHOW CREATE TABLE <catalog_name>.<database_name>.<table_name>
```

Запрос таблицы Iceberg

1. Используйте SHOW DATABASES для просмотра баз данных в вашем Iceberg cluster:

```
SHOW DATABASES FROM <catalog_name>
```

2. Переключитесь на Iceberg catalog и базу данных в нем.

3. Используйте SELECT для запроса целевой таблицы в указанной базе данных:

```
SELECT count(*) FROM <table_name> LIMIT 10
```

Создание базы данных Iceberg

Аналогично внутреннему каталогу Selena, если у вас есть привилегия CREATE DATABASE на Iceberg catalog, вы можете использовать оператор CREATE DATABASE для создания баз данных в этом Iceberg catalog. Эта функция поддерживается с v1.5.2.

> **ПОДСКАЗКА**
>
> Вы можете предоставлять и отзывать привилегии с помощью GRANT и REVOKE.

Переключитесь на Iceberg catalog, а затем используйте следующий оператор для создания базы данных Iceberg в этом каталоге:

```
CREATE DATABASE <database_name>
[PROPERTIES ("location" = "
<prefix>://<path_to_database>/<database_name.db>/")]
```

Вы можете использовать параметр `location` для указания пути к файлу, в котором вы хотите создать базу данных. Поддерживаются как HDFS, так и облачное хранилище. Если вы не указываете параметр `location`, Selena создает базу данных в пути к файлу по умолчанию Iceberg catalog.

`prefix` варьируется в зависимости от используемой вами системы хранения:

HDFS

Значение `Prefix`: `hdfs`

Google GCS

Значение `Prefix`: `gs`

Azure Blob Storage

Значение `Prefix`:

- Если ваша учетная запись хранения разрешает доступ по HTTP, `prefix` — `wasb`.
- Если ваша учетная запись хранения разрешает доступ по HTTPS, `prefix` — `wasbs`.

Azure Data Lake Storage Gen1

Значение `Prefix`: `adl`

Azure Data Lake Storage Gen2

Значение `Prefix`:

- Если ваша учетная запись хранения разрешает доступ по HTTP, `prefix` — `abfs`.
- Если ваша учетная запись хранения разрешает доступ по HTTPS, `prefix` — `abfss`.

AWS S3 или другое S3-совместимое хранилище (например, MinIO)

Значение `Prefix`: `s3`

Удаление базы данных Iceberg

Аналогично внутренним базам данных Selena, если у вас есть привилегия DROP на базу данных Iceberg, вы можете использовать оператор DROP DATABASE для удаления этой базы данных Iceberg. Эта функция поддерживается с v1.5.2. Вы можете удалять только пустые базы данных.

Когда вы удаляете базу данных Iceberg, путь к файлу базы данных на вашем HDFS cluster или облачном хранилище не будет удален вместе с базой данных.

Переключитесь на Iceberg catalog, а затем используйте следующий оператор для удаления базы данных Iceberg в этом каталоге:

```
DROP DATABASE <database_name>;
```

Создание таблицы Iceberg

Аналогично внутренним базам данных Selena, если у вас есть привилегия CREATE TABLE на базу данных Iceberg, вы можете использовать оператор CREATE TABLE или CREATE

TABLE AS SELECT для создания таблицы в этой базе данных Iceberg. Эта функция поддерживается с v1.5.2.

Переключитесь на Iceberg catalog и базу данных в нем, а затем используйте следующий синтаксис для создания таблицы Iceberg в этой базе данных.

Синтаксис

```
CREATE TABLE [IF NOT EXISTS] [database.]table_name
(column_definition1[, column_definition2, ...
partition_column_definition1,partition_column_definition2...])
[partition_desc]
[ORDER BY sort_desc)]
[PROPERTIES ("key" = "value", ...)]
[AS SELECT query]
```

Параметры

column_definition

Синтаксис `column_definition` следующий:

```
col_name col_type [COMMENT 'comment']
```

> **ПРИМЕЧАНИЕ**
>
> Все столбцы, не являющиеся partition columns, должны использовать `NULL` в качестве значения по умолчанию. Это означает, что вы должны указать `DEFAULT "NULL"` для каждого столбца, не являющегося partition column, в операторе создания таблицы. Кроме того, partition columns должны быть определены после столбцов, не являющихся partition columns, и не могут использовать `NULL` в качестве значения по умолчанию.

partition_desc

Синтаксис `partition_desc` следующий:

```
PARTITION BY (partition_expr[, partition_expr...])
```

Каждый `partition_expr` может быть одной из следующих форм:

```
column_name
| transform_expr(column_name)
| transform_expr(column_name, parameter)
```

В настоящее время Selena поддерживает выражения преобразования partition, определенные в спецификации Apache Iceberg transform expr. Это позволяет Selena создавать таблицы Iceberg со скрытыми partition на основе преобразованных значений столбцов.

> **ПРИМЕЧАНИЕ**
>
> Partition columns должны быть определены после столбцов, не являющихся partition columns. Partition columns поддерживают все типы данных, за исключением FLOAT, DOUBLE, DECIMAL и DATETIME, и не могут использовать `NULL` в качестве значения по умолчанию.

ORDER BY

С v2.0.0 Selena поддерживает указание ключей сортировки для таблицы Iceberg через предложение ORDER BY, которое можно использовать для сортировки данных в одном файле данных в соответствии с указанным ключом сортировки.

Предложение ORDER BY может содержать более одного ключа сортировки в следующем формате:

```
ORDER BY (column_name [sort_direction] [nulls_order], ...)
```

- `column_name`: Имя столбца, который будет использоваться в качестве ключа сортировки. Это должен быть столбец, существующий в схеме таблицы. В настоящее время выражения преобразования не поддерживаются.
- `sort_direction`: Направление сортировки. Допустимые значения: `ASC` (по умолчанию) и `DESC`.
- `nulls_order`: Порядок NULL значений. Допустимые значения: `NULLS FIRST` (по умолчанию, когда указано `ASC`) и `NULLS LAST` (по умолчанию, когда указано `DESC`).

`sort_direction` и `nulls_order` являются опциональными. Например, каждый из следующих является допустимым `sort_desc`:

- `column_name`
- `column_name ASC`
- `column_name DESC NULLS FIRST`

PROPERTIES

Вы можете указать атрибуты таблицы в формате `"key" = "value"` в `PROPERTIES`. См. Атрибуты таблиц Iceberg.

В следующей таблице описаны несколько ключевых свойств.

location

Описание: Путь к файлу, в котором вы хотите создать таблицу Iceberg. Когда вы используете HMS в качестве metastore, вам не нужно указывать параметр `location`, поскольку Selena создаст таблицу в пути к файлу по умолчанию текущего Iceberg catalog. Когда вы используете AWS Glue в качестве metastore:

- Если вы указали параметр `location` для базы данных, в которой вы хотите создать таблицу, вам не нужно указывать параметр `location` для таблицы. В этом случае таблица по умолчанию находится в пути к файлу базы данных, к которой она принадлежит.
- Если вы не указали `location` для базы данных, в которой вы хотите создать таблицу, вы должны указать параметр `location` для таблицы.

file_format

Описание: Формат файла таблицы Iceberg. Поддерживается только формат Parquet. Значение по умолчанию: `parquet`.

compression_codec

Описание: Алгоритм сжатия, используемый для таблицы Iceberg. Поддерживаемые алгоритмы сжатия: SNAPPY, GZIP, ZSTD и LZ4. Значение по умолчанию: `zstd`.

Примеры

1. Создайте таблицу без partition с именем `unpartition_tbl`. Таблица состоит из двух столбцов, `id` и `score`, как показано ниже:

```
CREATE TABLE unpartition_tbl
(
    id int,
    score double
);
```

2. Создайте таблицу с partition с именем `partition_tbl_1`. Таблица состоит из трех столбцов, `action`, `id` и `dt`, из которых `id` и `dt` определены как partition columns, как показано ниже:

```
CREATE TABLE partition_tbl_1
(
    action varchar(20),
    id int,
    dt date
)
PARTITION BY (id,dt);
```

3. Запросите существующую таблицу с именем `partition_tbl_1` и создайте таблицу с partition с именем `partition_tbl_2` на основе результата запроса `partition_tbl_1`. Для `partition_tbl_2` `id` и `dt` определены как partition columns, как показано ниже:

```
CREATE TABLE partition_tbl_2
PARTITION BY (id, dt)
AS SELECT * from employee;
```

4. Создайте таблицу с именем `partition_tbl_3` со скрытыми partition. Таблица содержит три столбца: `action`, `id` и `dt`. Среди них `id` и `dt` используются в качестве ключей partition, но partition определены выражениями преобразования, поэтому эти partition являются скрытыми.

```
CREATE TABLE partition_tbl_3 (
  action VARCHAR(20),
  id INT,
  dt DATE
)
PARTITION BY bucket(id, 10), year(dt);
```

Эволюция спецификации partition (ADD/DROP PARTITION COLUMN)

Selena поддерживает эволюцию спецификации partition таблицы Iceberg путем добавления или удаления partition columns (включая выражения преобразования) с помощью `ALTER TABLE ... ADD|DROP PARTITION COLUMN`.

Синтаксис

```
ALTER TABLE [catalog.][database.]table_name
ADD PARTITION COLUMN partition_expr [, partition_expr ...];

ALTER TABLE [catalog.][database.]table_name
DROP PARTITION COLUMN partition_expr [, partition_expr ...];
```

`partition_expr` может быть именем столбца (identity transform) или одним из поддерживаемых выражений преобразования. Поддерживаемые выражения преобразования: `year`, `month`, `day`, `hour`, `truncate` и `bucket`.

Примеры

```
ALTER TABLE test_part_evo
ADD PARTITION COLUMN dt, truncate(value, 10), bucket(id, 10);

ALTER TABLE test_part_evo
DROP PARTITION COLUMN dt;

ALTER TABLE test_part_evo
ADD PARTITION COLUMN month(dt);
```

Запись данных в таблицу Iceberg

Аналогично внутренним таблицам Selena, если у вас есть привилегия INSERT на таблицу Iceberg, вы можете использовать оператор INSERT для записи данных таблицы Selena в эту таблицу Iceberg (в настоящее время поддерживаются только таблицы Iceberg формата Parquet). Эта функция поддерживается с v1.5.2.

Переключитесь на Iceberg catalog и базу данных в нем, а затем используйте следующий синтаксис для записи данных таблицы Selena в таблицу Iceberg формата Parquet в этой базе данных.

Синтаксис

```
INSERT {INTO | OVERWRITE} <table_name>
[ (column_name [, ...]) ]
{ VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }

-- Если вы хотите записать данные в указанные partition, используйте
следующий синтаксис:
INSERT {INTO | OVERWRITE} <table_name>
PARTITION (par_col1=<value> [, par_col2=<value>...])
{ VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }
```

> **ПРИМЕЧАНИЕ**
>
> Partition columns не допускают значений `NULL`. Поэтому вы должны убедиться, что в partition columns таблицы Iceberg не загружаются пустые значения.

Параметры

INTO

Для добавления данных таблицы Selena к таблице Iceberg.

OVERWRITE

Для перезаписи существующих данных таблицы Iceberg данными таблицы Selena.

column_name

Имя целевого столбца, в который вы хотите загрузить данные. Вы можете указать один или несколько столбцов. Если вы указываете несколько столбцов, разделите их запятыми (`,`). Вы можете указывать только столбцы, которые фактически существуют в таблице Iceberg, и целевые столбцы, которые вы указываете, должны включать partition columns таблицы Iceberg. Целевые столбцы, которые вы указываете, сопоставляются один к одному последовательно со столбцами таблицы Selena, независимо от названий целевых столбцов. Если целевые столбцы не указаны, данные загружаются во все столбцы таблицы Iceberg. Если столбец таблицы Selena, не являющийся partition column, не может быть сопоставлен ни с одним столбцом таблицы Iceberg, Selena записывает значение по умолчанию `NULL` в столбец таблицы Iceberg. Если оператор INSERT содержит оператор запроса, возвращаемые типы столбцов которого отличаются от типов данных целевых

столбцов, Selena выполняет неявное преобразование несовпадающих столбцов. Если преобразование не удается, будет возвращена ошибка синтаксического анализа.

expression

Выражение, которое присваивает значения целевому столбцу.

DEFAULT

Присваивает значение по умолчанию целевому столбцу.

query

Оператор запроса, результат которого будет загружен в таблицу Iceberg. Это может быть любой оператор SQL, поддерживаемый Selena.

PARTITION

Partition, в которые вы хотите загрузить данные. Вы должны указать все partition columns таблицы Iceberg в этом свойстве. Partition columns, которые вы указываете в этом свойстве, могут быть в другой последовательности, чем partition columns, которые вы определили в операторе создания таблицы. Если вы указываете это свойство, вы не можете указать свойство `column_name`.

Примеры

1. Вставьте три строки данных в таблицу `partition_tbl_1`:

```
INSERT INTO partition_tbl_1
VALUES
    ("buy", 1, "2023-09-01"),
    ("sell", 2, "2023-09-02"),
    ("buy", 3, "2023-09-03");
```

2. Вставьте результат запроса SELECT, который содержит простые вычисления, в таблицу `partition_tbl_1`:

```
INSERT INTO partition_tbl_1 (id, action, dt) SELECT 1+1, 'buy', '2023-
09-03';
```

3. Вставьте результат запроса SELECT, который читает данные из таблицы `partition_tbl_1`, в ту же таблицу:

```
INSERT INTO partition_tbl_1 SELECT 'buy', 1, date_add(dt, INTERVAL 2
DAY)
FROM partition_tbl_1
WHERE id=1;
```

4. Вставьте результат запроса SELECT в partition, которые соответствуют двум условиям, `dt='2023-09-01'` и `id=1`, таблицы `partition_tbl_2`:

```
INSERT INTO partition_tbl_2 SELECT 'order', 1, '2023-09-01';
```

Или

```
INSERT INTO partition_tbl_2 partition(dt='2023-09-01',id=1) SELECT
'order';
```

5. Перезаписать все значения столбца `action` в partition, которые соответствуют двум условиям, `dt='2023-09-01'` и `id=1`, таблицы `partition_tbl_1` на `close`:

```
INSERT OVERWRITE partition_tbl_1 SELECT 'close', 1, '2023-09-01';
```

Или

```
INSERT OVERWRITE partition_tbl_1 partition(dt='2023-09-01',id=1) SELECT
'close';
```

Удаление таблицы Iceberg

Аналогично внутренним таблицам Selena, если у вас есть привилегия DROP на таблицу Iceberg, вы можете использовать оператор DROP TABLE для удаления этой таблицы Iceberg. Эта функция поддерживается с v1.5.2.

Когда вы удаляете таблицу Iceberg, путь к файлу таблицы и данные на вашем HDFS cluster или облачном хранилище не будут удалены вместе с таблицей.

Когда вы принудительно удаляете таблицу Iceberg (а именно, с ключевым словом `FORCE`, указанным в операторе DROP TABLE), данные таблицы на вашем HDFS cluster или облачном хранилище будут удалены вместе с таблицей, но путь к файлу таблицы сохраняется.

Переключитесь на Iceberg catalog и базу данных в нем, а затем используйте следующий оператор для удаления таблицы Iceberg в этой базе данных.

```
DROP TABLE <table_name> [FORCE];
```

Создание представления Iceberg

Вы можете определять представления Iceberg в Selena или добавлять диалект Selena к существующему представлению Iceberg. Запросы к таким представлениям Iceberg поддерживают абстрагирование диалекта Selena этих представлений. Эта функция поддерживается с v1.5.2.

```
CREATE VIEW [IF NOT EXISTS]
[<catalog>.<database>.]<view_name>
(
    <column_name>[ COMMENT 'column comment']
    [, <column_name>[ COMMENT 'column comment'], ...]
)
[COMMENT 'view comment']
[PROPERTIES ("key" = "value", ...)]
AS <query_statement>
```

Пример

Создайте представление Iceberg `iceberg_view1` на основе таблицы Iceberg `iceberg_table`.

```
CREATE VIEW IF NOT EXISTS iceberg.iceberg_db.iceberg_view1 AS
SELECT k1, k2 FROM iceberg.iceberg_db.iceberg_table;
```

С v2.0.0 вы можете указывать атрибуты представления в формате `"key" = "value"` в `PROPERTIES`.

```
CREATE VIEW IF NOT EXISTS iceberg.iceberg_db.iceberg_view1
PROPERTIES (
  "key1" = "value1",
  "key2" = "value2"
)
AS
SELECT k1, k2 FROM iceberg.iceberg_db.iceberg_table;
```

Добавление или изменение диалекта Selena для существующего представления Iceberg

Если ваши представления Iceberg созданы из других систем, таких как Apache Spark, при этом вы хотите запрашивать эти представления из Selena, вы можете добавить диалект Selena к этим представлениям. Эта функция поддерживается с v1.5.2.

> **ПРИМЕЧАНИЕ**
>
> - Вы должны гарантировать, что основные значения обоих диалектов представления идентичны. Selena и другие системы не гарантируют согласованность между различными определениями.
> - Вы можете определить только один диалект Selena для каждого представления Iceberg. Вы можете изменить определение диалекта с помощью предложения MODIFY.

```
ALTER VIEW
[<catalog>.<database>.]<view_name>
(
    <column_name>
    [, <column_name>]
)
{ ADD | MODIFY } DIALECT
<query_statement>
```

Пример

1. Добавьте диалект Selena к существующему представлению Iceberg `iceberg_view2`.

```
ALTER VIEW iceberg.iceberg_db.iceberg_view2 ADD DIALECT SELECT k1, k2 FROM
iceberg.iceberg_db.iceberg_table;
```

2. Измените диалект Selena для существующего представления Iceberg `iceberg_view2`.

```
ALTER VIEW iceberg.iceberg_db.iceberg_view2 MODIFY DIALECT SELECT k1, k2,
k3 FROM iceberg.iceberg_db.iceberg_table;
```

Ручная компакция

С v2.0.0 Selena поддерживает ручную компакцию таблиц Iceberg.

Каждый раз, когда данные загружаются в таблицу Iceberg, генерируются новые файлы данных и метаданных. С течением времени чрезмерное количество файлов данных может значительно замедлить генерацию плана запроса и оказать влияние на производительность.

В этом случае вам нужно выполнить ручную компакцию таблицы или partition, чтобы объединить небольшие файлы данных и тем самым улучшить производительность.

Синтаксис

```
ALTER TABLE [catalog.][database.]table_name
EXECUTE rewrite_data_files
("key"=value [,"key"=value, ...])
[WHERE <predicate>]
```

Параметры

Свойства `rewrite_data_files`

Пары `"key"=value`, которые объявляют поведение ручной компакции. Обратите внимание, что вам нужно обернуть ключ в двойные кавычки.

`min_file_size_bytes`

- Описание: Верхний предел небольшого файла данных. Файлы данных, размер которых меньше этого значения, будут объединены во время компакции.
- Единица: Байт
- Тип: Int

- По умолчанию: 268,435,456 (256 МБ)

`batch_size`

- Описание: Максимальный размер данных, которые могут быть обработаны в каждой партии.
- Единица: Байт
- Тип: Int
- По умолчанию: 10,737,418,240 (10 ГБ)

`rewrite_all`

- Описание: Перезаписывать ли все файлы данных во время компакции, игнорируя параметры, которые фильтруют файлы данных с определенными требованиями.
- Единица: -
- Тип: Boolean
- По умолчанию: false

Предложение `WHERE`

- Описание: Предикат фильтра, используемый для указания partition, которые будут участвовать в компакции.

Пример

Следующий пример выполняет ручную компакцию определенных partition в таблице Iceberg `t1`. Partition представлены предложением `WHERE part_col = 'p1'`. В этих partition файлы данных, которые меньше 134,217,728 байт (128 МБ), будут объединены во время компакции.

```
ALTER TABLE t1 EXECUTE rewrite_data_files("min_file_size_bytes"= 134217728)
WHERE part_col = 'p1';
```

Настройка кэширования метаданных

Файлы метаданных вашего Iceberg cluster могут храниться в удаленном хранилище, таком как AWS S3 или HDFS. По умолчанию Selena кэширует метаданные Iceberg в памяти. Чтобы ускорить запросы, Selena использует двухуровневый механизм кэширования метаданных, с помощью которого она может кэшировать метаданные как в памяти, так и на диске. Для каждого первоначального запроса Selena кэширует результаты их вычислений. Если последующий запрос, семантически эквивалентный предыдущему,

выполняется, Selena сначала пытается извлечь запрошенные метаданные из своих кэшей, и она извлекает метаданные из удаленного хранилища только тогда, когда метаданные не могут быть найдены в ее кэшах.

Selena использует алгоритм Least Recently Used (LRU) для кэширования и вытеснения данных. Основные правила следующие:

- Selena сначала пытается извлечь запрошенные метаданные из памяти. Если метаданные не могут быть найдены в памяти, Selena пытается извлечь метаданные с дисков. Метаданные, которые Selena извлекла с дисков, будут загружены в память. Если метаданные также не могут быть найдены на дисках, Selena извлекает метаданные из удаленного хранилища и кэширует извлеченные метаданные в памяти.
- Selena записывает метаданные, вытесненные из памяти, на диски, но напрямую отбрасывает метаданные, вытесненные с дисков.

С v1.5.2 Selena поддерживает стратегию периодического обновления метаданных. Вы можете настроить план кэширования метаданных Iceberg с помощью системной переменной `plan_mode`.

Конфигурации FE по кэшированию метаданных Iceberg

enable_iceberg_metadata_disk_cache

- Единица: Н/Д
- Значение по умолчанию: `false`
- Описание: Указывает, следует ли включить кэш диска.

iceberg_metadata_cache_disk_path

- Единица: Н/Д
- Значение по умолчанию: `SelenaFE.SELENA_HOME_DIR + "/caches/iceberg"`
- Описание: Путь сохранения кэшированных файлов метаданных на диске.

iceberg_metadata_disk_cache_capacity

- Единица: Байты
- Значение по умолчанию: `2147483648`, эквивалентно 2 ГБ
- Описание: Максимальный размер кэшированных метаданных, разрешенных на диске.

iceberg_metadata_memory_cache_capacity

- Единица: Байты
- Значение по умолчанию: `536870912`, эквивалентно 512 МБ
- Описание: Максимальный размер кэшированных метаданных, разрешенных в памяти.

iceberg_metadata_memory_cache_expiration_seconds

- Единица: Секунды
- Значение по умолчанию: `86500`
- Описание: Количество времени, после которого запись кэша в памяти истекает, отсчитываемое от ее последнего доступа.

iceberg_metadata_disk_cache_expiration_seconds

- Единица: Секунды
- Значение по умолчанию: `604800`, эквивалентно одной неделе
- Описание: Количество времени, после которого запись кэша на диске истекает, отсчитываемое от ее последнего доступа.

iceberg_metadata_cache_max_entry_size

- Единица: Байты
- Значение по умолчанию: `8388608`, эквивалентно 8 МБ
- Описание: Максимальный размер файла, который может быть кэширован. Файлы, размер которых превышает значение этого параметра, не могут быть кэшированы. Если запрос запрашивает эти файлы, Selena извлекает их из удаленного хранилища.

enable_background_refresh_connector_metadata

- Единица: -
- Значение по умолчанию: true
- Описание: Включать ли периодическое обновление кэша метаданных Iceberg. После включения Selena опрашивает metastore (Hive Metastore или AWS Glue) вашего Iceberg cluster и обновляет кэшированные метаданные часто используемых Iceberg catalogs для восприятия изменений данных. `true` указывает на включение обновления кэша метаданных Iceberg, а `false` указывает на отключение.

background_refresh_metadata_interval_millis

- Единица: Миллисекунда
- Значение по умолчанию: 600000

- Описание: Интервал между двумя последовательными обновлениями кэша метаданных Iceberg. - Единица: миллисекунда.

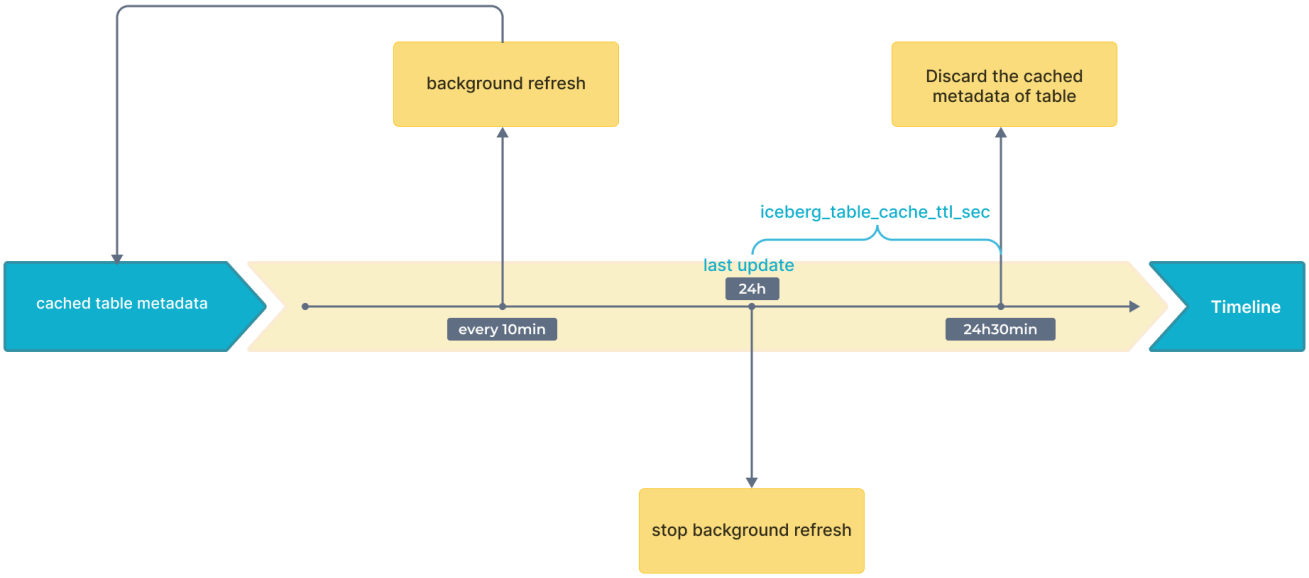
background_refresh_metadata_time_secs_since_last_access_sec

- Единица: Секунда
- Значение по умолчанию: 86400
- Описание: Время истечения задачи обновления кэша метаданных Iceberg. Для Iceberg catalog, к которому был осуществлен доступ, если к нему не было доступа более указанного времени, Selena прекращает обновление его кэшированных метаданных. Для Iceberg catalog, к которому не было доступа, Selena не будет обновлять его кэшированные метаданные.

Приложение A: Стратегия периодического обновления метаданных

Iceberg поддерживает snapshots. С новейшим snapshot вы можете получить новейший результат. Поэтому только кэшированные snapshots могут влиять на свежесть данных. В результате вам нужно обращать внимание только на стратегию обновления кэша, содержащего snapshot.

Следующая блок-схема показывает временные интервалы на временной шкале.



Приложение B: Парсинг файлов метаданных

- **Распределенный план для большого объема метаданных**

  Для эффективной обработки большого объема метаданных Selena использует распределенный подход с использованием нескольких узлов BE и CN. Этот метод использует возможности параллельных вычислений современных движков запросов, которые могут распределять задачи, такие как чтение, декомпрессия и фильтрация файлов manifest, по нескольким узлам. Обрабатывая эти файлы manifest параллельно, время, необходимое для извлечения метаданных, значительно сокращается, что приводит к более быстрому планированию заданий. Это особенно полезно для больших запросов, включающих множество файлов manifest, поскольку устраняет узкие места в одной точке и повышает общую эффективность выполнения запросов.

- **Локальный план для небольшого объема метаданных**

  Для меньших запросов, где повторная декомпрессия и парсинг файлов manifest могут вводить ненужные задержки, используется другая стратегия. Selena кэширует десериализованные объекты памяти, особенно файлы Avro, для решения этой проблемы. Храня эти десериализованные файлы в памяти, система может обойти этапы декомпрессии и парсинга для последующих запросов. Этот механизм кэширования позволяет получить прямой доступ к необходимым метаданным, значительно сокращая время извлечения. В результате система становится более отзывчивой и лучше подходит для удовлетворения высоких требований к запросам и потребностей перезаписи материализованных представлений.

- **Адаптивная стратегия извлечения метаданных** (по умолчанию)

  Selena спроектирована для автоматического выбора подходящего метода извлечения метаданных на основе различных факторов, включая количество узлов FE и BE/CN, количество их ядер CPU и количество файлов manifest, необходимых для текущего запроса. Этот адаптивный подход гарантирует, что система динамически оптимизирует извлечение метаданных без необходимости ручной настройки параметров, связанных с метаданными. Таким образом, Selena обеспечивает бесшовный опыт, балансируя между распределенными и локальными планами для достижения оптимальной производительности запросов в различных условиях.

Вы можете настроить план кэширования метаданных Iceberg с помощью системной переменной `plan_mode`.

Time Travel с Iceberg Catalog

> *Beta* **BETA FEATURE**
>
> Рекомендации по использованию бета-функций

Эта тема описывает функцию Time Travel в Selena для Iceberg catalogs. Эта функция поддерживается начиная с версии v1.5.2.

Обзор

Каждая Iceberg таблица поддерживает лог metadata snapshots, который представляет изменения, применённые к ней. Базы данных могут выполнять Time Travel запросы к Iceberg таблицам, обращаясь к этим историческим snapshots. Iceberg поддерживает ветвление (branching) и тегирование (tagging) snapshots для сложного управления жизненным циклом snapshots, позволяя каждой ветке (branch) или тегу (tag) поддерживать свой собственный жизненный цикл на основе настроенных политик хранения. Для получения дополнительной информации о функции branching и tagging в Iceberg см. Официальную документацию.

Интегрируя функцию snapshot branching и tagging Iceberg, Selena поддерживает создание и управление branches и tags в Iceberg catalogs, а также Time Travel запросы к таблицам внутри них.

Управление branches, tags и snapshots

Создание branch

Синтаксис `CREATE BRANCH`

```
ALTER TABLE [catalog.][database.]table_name
CREATE [OR REPLACE] BRANCH [IF NOT EXISTS] <branch_name>
[AS OF VERSION <snapshot_id>]
[RETAIN <int> { DAYS | HOURS | MINUTES }]
[WITH SNAPSHOT RETENTION
    { minSnapshotsToKeep | maxSnapshotAge | minSnapshotsToKeep
maxSnapshotAge }]

minSnapshotsToKeep ::= <int> SNAPSHOTS

maxSnapshotAge ::= <int> { DAYS | HOURS | MINUTES }
```

Параметры

- `branch_name`: Имя создаваемой ветки.
- `AS OF VERSION`: ID snapshot (версии), на основе которого создаётся ветка.
- `RETAIN`: Время хранения ветки. Формат: `<int> <unit>`. Поддерживаемые единицы: `DAYS`, `HOURS` и `MINUTES`. Пример: `7 DAYS`, `12 HOURS` или `30 MINUTES`.
- `WITH SNAPSHOT RETENTION`: Минимальное количество snapshots для хранения и/или максимальное время хранения snapshots.

Пример

Создать ветку `test-branch` на основе версии (snapshot ID) `12345` таблицы `iceberg.sales.order`, сохранить ветку на `7` дней и хранить минимум `2` snapshots на ветке.

```
ALTER TABLE iceberg.sales.order CREATE BRANCH `test-branch`
AS OF VERSION 12345
RETAIN 7 DAYS
WITH SNAPSHOT RETENTION 2 SNAPSHOTS;
```

Создать ветку `test-branch2` на основе версии (snapshot ID) `12345` таблицы `iceberg.sales.order`, сохранить ветку на `7` дней и хранить snapshots на ветке максимум `2` дня.

```
ALTER TABLE iceberg.sales.order CREATE BRANCH `test-branch2`
AS OF VERSION 12345
RETAIN 7 DAYS
WITH SNAPSHOT RETENTION 2 DAYS;
```

Создать ветку `test-branch3` на основе версии (snapshot ID) `12345` таблицы `iceberg.sales.order`, сохранить ветку на `7` дней и хранить минимум `2` snapshots на ветке, каждый максимум `2` дня.

```
ALTER TABLE iceberg.sales.order CREATE BRANCH `test-branch3`
AS OF VERSION 12345
RETAIN 7 DAYS
WITH SNAPSHOT RETENTION 2 SNAPSHOTS 2 DAYS;
```

Загрузка данных в конкретную ветку таблицы

Синтаксис `VERSION AS OF`

```
INSERT INTO [catalog.][database.]table_name
[FOR] VERSION AS OF <branch_name>
<query_statement>
```

Параметры

- `branch_name`: Имя ветки таблицы, в которую загружаются данные.
- `query_statement`: Запрос, результат которого будет загружен в целевую таблицу. Это может быть любой SQL запрос, поддерживаемый Selena.

Пример

Загрузить результат запроса в ветку `test-branch` таблицы `iceberg.sales.order`.

```
INSERT INTO iceberg.sales.order
FOR VERSION AS OF `test-branch`
SELECT c1, k1 FROM tbl;
```

Создание tag

Синтаксис `CREATE TAG`

```
ALTER TABLE [catalog.][database.]table_name
CREATE [OR REPLACE] TAG [IF NOT EXISTS] <tag_name>
[AS OF VERSION <snapshot_id>]
[RETAIN <int> { DAYS | HOURS | MINUTES }]
```

Параметры

- `tag_name`: Имя создаваемого тега.
- `AS OF VERSION`: ID snapshot (версии), на основе которого создаётся тег.
- `RETAIN`: Время хранения тега. Формат: `<int> <unit>`. Поддерживаемые единицы: `DAYS`, `HOURS` и `MINUTES`. Пример: `7 DAYS`, `12 HOURS` или `30 MINUTES`.

Пример

Создать тег `test-tag` на основе версии (snapshot ID) `12345` таблицы `iceberg.sales.order` и сохранить тег на `7` дней.

```
ALTER TABLE iceberg.sales.order CREATE TAG `test-tag`
AS OF VERSION 12345
RETAIN 7 DAYS;
```

Ускоренное перемещение одной ветки к другой (Fast forward)

Синтаксис `fast_forward`

```
ALTER TABLE [catalog.][database.]table_name
EXECUTE fast_forward('<from_branch>', '<to_branch>')
```

Параметры

- `from_branch`: Ветка, которую вы хотите переместить (fast forward). Оберните имя ветки в кавычки.
- `to_branch`: Ветка, к которой вы хотите переместить `from_branch`. Оберните имя ветки в кавычки.

Пример

Переместить ветку `main` к ветке `test-branch`.

```
ALTER TABLE iceberg.sales.order
EXECUTE fast_forward('main', 'test-branch');
```

Cherry pick snapshot

Вы можете выбрать (cherry pick) конкретный snapshot и применить его к текущему состоянию таблицы. Эта операция создаст новый snapshot на основе существующего snapshot, и оригинальный snapshot не будет затронут.

Синтаксис `cherrypick_snapshot`

```
ALTER TABLE [catalog.][database.]table_name
EXECUTE cherrypick_snapshot(<snapshot_id>)
```

Параметр

`snapshot_id`: ID snapshot, который вы хотите выбрать (cherry pick).

Пример

```
ALTER TABLE iceberg.sales.order
EXECUTE cherrypick_snapshot(54321);
```

Истечение срока snapshots (Expire snapshots)

Вы можете установить истечение срока для snapshots раньше определённого момента времени. Эта операция удалит файлы данных истёкших snapshots.

Синтаксис `expire_snapshots`

```
ALTER TABLE [catalog.][database.]table_name
EXECUTE expire_snapshots('<datetime>')
```

Пример

```
ALTER TABLE iceberg.sales.order
EXECUTE expire_snapshots('2023-12-17 00:14:38')
```

Удаление branch или tag

Синтаксис `DROP BRANCH`, `DROP TAG`

```
ALTER TABLE [catalog.][database.]table_name
DROP { BRANCH <branch_name> | TAG <tag_name> }
```

Пример

```
ALTER TABLE iceberg.sales.order
DROP BRANCH `test-branch`;

ALTER TABLE iceberg.sales.order
DROP TAG `test-tag`;
```

Запросы с Time Travel

Time Travel к конкретной ветке или тегу

Синтаксис `VERSION AS OF`

```
[FOR] VERSION AS OF '<branch_or_tag>'
```

Параметр

`tag_or_branch`: Имя ветки или тега, к которому вы хотите выполнить Time Travel. Если указано имя ветки, запрос выполнит Time Travel к head snapshot ветки. Если указано имя тега, запрос выполнит Time Travel к snapshot, на который ссылается тег.

Пример

```
-- Time Travel к head snapshot ветки.
SELECT * FROM iceberg.sales.order VERSION AS OF 'test-branch';
-- Time Travel к snapshot, на который ссылается тег.
SELECT * FROM iceberg.sales.order VERSION AS OF 'test-tag';
```

Time Travel к конкретному snapshot

Синтаксис `VERSION AS OF`

```
[FOR] VERSION AS OF '<snapshot_id>'
```

Параметр

`snapshot_id`: ID snapshot, к которому вы хотите выполнить Time Travel.

Пример

```
SELECT * FROM iceberg.sales.order VERSION AS OF 12345;
```

Time Travel к конкретной дате и времени или дате

Синтаксис `TIMESTAMP AS OF`

```
[FOR] TIMESTAMP AS OF { '<datetime>' | '<date>' | date_and_time_function }
```

Параметр

`date_and_time_function`: Любые функции даты и времени, поддерживаемые Selena.

Пример

```
SELECT * FROM iceberg.sales.order TIMESTAMP AS OF '1986-10-26 01:21:00';
SELECT * FROM iceberg.sales.order TIMESTAMP AS OF '1986-10-26';
SELECT * FROM iceberg.sales.order TIMESTAMP AS OF now();
```

Таблицы метаданных Iceberg

Этот раздел описывает, как проверять информацию о метаданных таблиц Iceberg в Selena.

Обзор

Начиная с v1.5.2, Selena поддерживает таблицы метаданных Iceberg. Эти таблицы метаданных содержат различную информацию о таблицах Iceberg, такую как история изменений таблицы, snapshots и manifests. Вы можете запросить каждую таблицу метаданных, добавив имя таблицы метаданных к исходному имени таблицы.

В настоящее время Selena поддерживает следующие таблицы метаданных Iceberg:

Таблица метаданных	Описание
`history`	Показывает журнал изменений метаданных, внесенных в таблицу.
`metadata_log_entries`	Показывает записи журнала метаданных для таблицы.
`snapshots`	Показывает сведения о snapshots таблицы.
`manifests`	Показывает обзор manifests, связанных со snapshots в журнале таблицы.
`partitions`	Показывает сведения о partition в таблице.
`files`	Показывает сведения о файлах данных и файлах удаления в текущем snapshot таблицы.
`refs`	Показывает сведения о ссылках Iceberg, включая branches и tags.

Таблица history

Использование:

```
SELECT * FROM [<catalog>.][<database>.]table$history;
```

Вывод:

Поле	Описание
made_current_at	Время, когда snapshot стал текущим snapshot.
snapshot_id	ID snapshot.
parent_id	ID родительского snapshot.
is_current_ancestor	Является ли этот snapshot предком текущего snapshot.

Таблица metadata_log_entries

Использование:

```
SELECT * FROM [<catalog>.][<database>.]table$metadata_log_entries;
```

Вывод:

Поле	Описание
timestamp	Время, когда были записаны метаданные.
file	Местоположение файла метаданных.
latest_snapshot_id	ID последнего snapshot, когда метаданные были обновлены.
latest_schema_id	ID последней схемы, когда метаданные были обновлены.
latest_sequence_number	Порядковый номер данных файла метаданных.

Таблица snapshots

Использование:

```
SELECT * FROM [<catalog>.][<database>.]table$snapshots;
```

Вывод:

Поле	Описание
committed_at	Время, когда snapshot был зафиксирован.
snapshot_id	ID для snapshot.
parent_id	ID для родительского snapshot.
operation	Тип операции, выполненной на таблице Iceberg. Допустимые значения: • `append`: Новые данные добавлены. • `replace`: Файлы удалены и заменены без изменения данных в таблице. • `overwrite`: Старые данные перезаписаны новыми данными. • `delete`: Данные удалены из таблицы.
manifest_list	Список файлов manifest Avro, которые содержат подробную информацию об изменениях snapshot.
summary	Сводка изменений, внесенных из предыдущего snapshot в текущий snapshot.

Таблица manifests

Использование:

```
SELECT * FROM [<catalog>.][<database>.]table$manifests;
```

Вывод:

Поле	Описание
path	Местоположение файла manifest.
length	Длина файла manifest.
partition_spec_id	ID спецификации partition, которая используется для записи файла manifest.
added_snapshot_id	ID snapshot, во время которого эта запись manifest была добавлена.
added_data_files_count	Количество файлов данных со статусом `ADDED` в файле manifest.
added_rows_count	Общее количество строк во всех файлах данных со статусом `ADDED` в файле manifest.
existing_data_files_count	Количество файлов данных со статусом `EXISTING` в файле manifest.
existing_rows_count	Общее количество строк во всех файлах данных со статусом `EXISTING` в файле manifest.
deleted_data_files_count	Количество файлов данных со статусом `DELETED` в файле manifest.
deleted_rows_count	Общее количество строк во всех файлах данных со статусом `DELETED` в файле manifest.
partition_summaries	Метаданные диапазона partition.

Таблица partitions

Использование:

```
SELECT * FROM [<catalog>.][<database>.]table$partitions;
```

Вывод:

Поле	Описание
partition_value	Сопоставление имен столбцов partition со значениями столбцов partition.
spec_id	Spec ID partition файлов.
record_count	Количество записей в partition.
file_count	Количество файлов, сопоставленных в partition.
total_data_file_size_in_bytes	Размер всех файлов данных в partition.
position_delete_record_count	Общее количество строк файлов Position Delete в partition.
position_delete_file_count	Количество файлов Position Delete в partition.

Поле	Описание
equality_delete_record_count	Общее количество строк файлов Equality Delete в partition.
equality_delete_file_count	Количество файлов Position Equality в partition.
last_updated_at	Время, когда partition был обновлен в последний раз.

Таблица files

Использование:

```
SELECT * FROM [<catalog>.][<database>.]table$files;
```

Вывод:

Поле	Описание
content	Тип содержимого, хранящегося в файле. Допустимые значения: `DATA(0)`, `POSITION_DELETES(1)` и `EQUALITY_DELETES(2)`.
file_path	Местоположение файла данных.
file_format	Формат файла данных.
spec_id	Spec ID, который используется для отслеживания файла, содержащего строку.
record_count	Количество записей, содержащихся в файле данных.
file_size_in_bytes	Размер файла данных.
column_sizes	Сопоставление между ID столбца Iceberg и его соответствующим размером в файле.
value_counts	Сопоставление между ID столбца Iceberg и его соответствующим количеством записей в файле.
null_value_counts	Сопоставление между ID столбца Iceberg и его соответствующим количеством значений `NULL` в файле.
nan_value_counts	Сопоставление между ID столбца Iceberg и его соответствующим количеством нечисловых значений в файле.
lower_bounds	Сопоставление между ID столбца Iceberg и его соответствующей нижней границей в файле.
upper_bounds	Сопоставление между ID столбца Iceberg и его соответствующей верхней границей в файле.
split_offsets	Список рекомендуемых местоположений для разделения.
sort_id	ID, представляющий порядок сортировки для этого файла.
equality_ids	Набор ID полей, используемых для сравнения равенства в файлах equality delete.
key_metadata	Метаданные о ключе шифрования, который используется для шифрования этого файла, если применимо.

Таблица refs

Использование:

```
SELECT * FROM [<catalog>.][<database>.]table$refs;
```

Вывод:

Поле	Описание
name	Имя ссылки.
type	Тип ссылки. Допустимые значения: `BRANCH` или `TAG`.
snapshot_id	Snapshot ID ссылки.
max_reference_age_in_ms	Максимальный возраст ссылки до того, как она может истечь.
min_snapshots_to_keep	Только для branch, минимальное количество snapshots для сохранения в branch.
max_snapshot_age_in_ms	Только для branch, максимальный возраст snapshot, разрешенный в branch. Более старые snapshots в branch истекут.

Создание Iceberg REST Catalog для AWS S3 Tables

Эта статья объясняет, как создать Iceberg REST Catalog в Selena для доступа к данным в AWS S3 tables через AWS Glue Iceberg REST endpoint.

AWS Glue Iceberg REST endpoint реализует спецификацию Iceberg REST Catalog Open API, которая предоставляет стандартизированный интерфейс для взаимодействия с Iceberg таблицами. Для доступа к S3 tables с использованием этого endpoint необходимо настроить credentials путём комбинирования IAM политик и авторизации AWS Lake Formation. Следующие разделы проведут вас через настройку прав доступа, включая определение необходимых политик, установку прав Lake Formation на уровне баз данных и таблиц, а также использование Selena для создания Iceberg REST catalog для доступа к S3 tables.

(Опционально) Создание table bucket

Вы можете пропустить этот шаг, если у вас уже есть table bucket для S3 tables.

1. Войдите в Amazon S3 Console как пользователь с правами администратора.
2. В верхнем правом углу страницы выберите ваш AWS регион.

3. В левой навигационной панели выберите **Table buckets** из навигационной панели.
4. Нажмите **Create table bucket**, чтобы создать table bucket.
5. После создания table bucket выберите его и нажмите **Create table with Athena**.
6. Создайте namespace.
7. После создания namespace снова нажмите **Create table with Athena**, чтобы создать таблицу.

> **ПРИМЕЧАНИЕ**
>
> Вы можете создать Database и Table с помощью Athena, а затем выполнять запросы к ним с помощью Selena. Альтернативно, вы можете просто создать table bucket, а затем использовать Selena для создания базы данных и таблицы.

Создание IAM Policy

Для доступа к S3 tables через AWS Glue endpoint создайте IAM Policy с правами для AWS Glue и операций Lake Formation:

1. Войдите в Amazon IAM Console как пользователь с правами администратора.
2. В верхнем правом углу страницы выберите ваш AWS регион.
3. В левой навигационной панели выберите **Policies** из навигационной панели.
4. Выберите **Create a policy**, выберите **JSON** в редакторе политик.
5. Добавьте следующую политику для предоставления доступа к AWS Glue и действиям Lake Formation.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "glue:GetCatalog",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:GetTable",
                "glue:GetTables",
                "glue:CreateTable",
                "glue:UpdateTable"
            ],
            "Resource": [
                "arn:aws:glue:<region>:<account-id>:catalog",
                "arn:aws:glue:<region>:<account-
id>:catalog/s3tablescatalog",
                "arn:aws:glue:<region>:<account-
id>:catalog/s3tablescatalog/<s3_table_bucket_name>",
                "arn:aws:glue:<region>:<account-
id>:table/s3tablescatalog/<s3_table_bucket_name>/<namespace>/*",
                "arn:aws:glue:<region>:<account-
id>:database/s3tablescatalog/<s3_table_bucket_name>/<namespace>"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lakeformation:GetDataAccess"
            ],
            "Resource": "*"
        }
    ]
}
```

После создания IAM политики прикрепите её к целевому IAM пользователю:

1. Выберите **Users** из навигационной панели.
2. Выберите пользователя, которому требуется доступ к S3 table.
3. Нажмите **Add permissions** и выберите **Attach policies directly**.
4. Прикрепите вновь созданную политику.

Управление правами через Lake Formation

Для доступа к S3 tables Selena требуется, чтобы Lake Formation сначала настроил права, позволяющие сторонним query engines получать доступ к S3 tables.

1. Войдите в Lake Formation Console как пользователь с правами администратора.
2. В верхнем правом углу страницы выберите ваш AWS регион.
3. В левой навигационной панели выберите **Application integration settings** из навигационной панели.
4. Выберите **Allow external engines to access data in Amazon S3 locations with full table access.**

Затем предоставьте вышеуказанному IAM пользователю права доступа в Lake Formation.

1. В левой навигационной панели Lake Formation Console выберите **Data permissions** из навигационной панели.
2. Нажмите **Grant**.
3. В разделе **Principals** выберите **IAM users and roles** и выберите авторизованного IAM пользователя из выпадающего списка **IAM users and roles**.
4. В разделе **LF-Tags or catalog resources** выберите **Named Data Catalog resources** и выберите ваш созданный table bucket в выпадающем списке **Catalogs**.
5. В разделе **Catalog permissions** выберите **Super** для **Catalog permissions**.
6. Нажмите **Grant**.

> **ПРИМЕЧАНИЕ**
>
> Здесь права Super предоставлены для удобства тестирования. Вам необходимо назначить соответствующие права на основе фактических требований в production окружении.

Создание Iceberg REST Catalog

Создайте Iceberg REST catalog в Selena:

```
CREATE EXTERNAL CATALOG selena_lakehouse_s3tables PROPERTIES(
  "type"="iceberg",
  "iceberg.catalog.type" = "rest",
  "iceberg.catalog.uri"  = "https://glue.<region>.amazonaws.com/iceberg",
  "iceberg.catalog.rest.sigv4-enabled" = "true",
  "iceberg.catalog.rest.signing-name" = "glue",
  "iceberg.catalog.rest.access-key-id" = "<iam_user_access_key>",
  "iceberg.catalog.rest.secret-access-key" = "<iam_user_secret_key>",
  "iceberg.catalog.warehouse" = "<accountid>:s3tablescatalog/<table-bucket-
name>",
  "aws.s3.region" = "<region>"
);
```

После этого вы можете создавать базы данных и таблицы и выполнять в них запросы.

Пример:

```
-- Переключиться на catalog
Selena> SET CATALOG selena_lakehouse_s3tables;
-- Создать базу данных
Selena> CREATE DATABASE s3table_db;
Query OK, 0 rows affected

-- Переключить базу данных
Selena> USE s3table_db;
Database changed

-- Создать таблицу
Selena> CREATE TABLE taxis (
    trip_id BIGINT,
    trip_distance FLOAT,
    fare_amount DOUBLE,
    store_and_fwd_flag STRING,
    vendor_id BIGINT
) PARTITION BY (vendor_id);
Query OK, 0 rows affected

-- Вставить данные
Selena> INSERT INTO taxis
VALUES (1000371, 1.8, 15.32, 'N', 1),
       (1000372, 2.5, 22.15, 'N', 2),
       (1000373, 0.9, 9.01, 'N', 2),
       (1000374, 8.4, 42.13, 'Y', 1);
Query OK, 4 rows affected

-- Запросить данные
Selena> SELECT * FROM taxis;
+---------+---------------+-------------+--------------------+-----------+
| trip_id | trip_distance | fare_amount | store_and_fwd_flag | vendor_id |
+---------+---------------+-------------+--------------------+-----------+
| 1000372 |           2.5 |       22.15 | N                  |         2 |
| 1000373 |           0.9 |        9.01 | N                  |         2 |
| 1000371 |           1.8 |       15.32 | N                  |         1 |
| 1000374 |           8.4 |       42.13 | Y                  |         1 |
+---------+---------------+-------------+--------------------+-----------+
4 rows in set
```

Настройка безопасности для Iceberg REST Catalog

В сценариях, где несколько узлов совместно обращаются к одному и тому же data lake, основная проблема заключается в **том, как обеспечить безопасное, унифицированное**

и проверяемое управление правами. В традиционных моделях пользователям необходимо настраивать storage credentials и выполнять локальный контроль прав индивидуально для каждого узла, что не только увеличивает затраты на обслуживание, но и создаёт риски утечки credentials и несогласованности прав.

Интегрируя Iceberg REST Catalog с Selena, можно достичь безопасного доступа и унифицированного управления правами через комбинацию JWT (JSON Web Token) аутентификации и Vended Credentials (временных credentials). Эта настройка помогает:

- **Уменьшить риски credentials**: Нет необходимости хранить информацию учётных записей с высокими привилегиями внутри Selena. Credentials выдаются временно Catalog, избегая утечки.
- **Унифицировать и упростить права**: Контроль доступа ко всем базам данных, таблицам и представлениям централизованно управляется Catalog, обеспечивая согласованность между различными узлами и избегая избыточной конфигурации.
- **Соответствие требованиям и упрощение операций**: Действия пользователей отслеживаются, что облегчает аудит. Это также снижает затраты на поддержание прав и storage credentials внутри Selena.

Механизмы безопасности

- **JWT аутентификация**
  - Когда пользователь входит в Selena, полученный JWT Token может быть передан в Iceberg REST Session Catalog.
  - Catalog будет аутентифицировать пользователя на основе JWT Token и выполнять запросы под идентичностью реального пользователя.
  - Преимущество в том, что Selena не нужно хранить учётные записи с высокими привилегиями, что значительно снижает риски безопасности.
- **Vended credentials**
  - После аутентификации Catalog может сгенерировать временные storage access credentials для пользователя.
  - Пользователям не нужно настраивать credentials для storage layer в Selena.
  - При каждом обращении к object storage Selena будет использовать временные credentials, выданные Catalog.
  - Это повышает безопасность и упрощает управление credentials.

Использование

Шаг 1. Настройка JWT аутентификации

В Selena настройте **Security Integration на основе JWT** или создайте пользователя с JWT аутентификацией.

Шаг 2. Создание Iceberg REST Catalog и настройка параметров безопасности

```
CREATE EXTERNAL CATALOG iceberg_rest_catalog
PROPERTIES (
  "iceberg.catalog.type" = "rest",
  "iceberg.catalog.uri" = "<rest_server_api_endpoint>",
  "iceberg.catalog.security" = "jwt",
  "iceberg.catalog.warehouse" = "<identifier_or_path_to_warehouse>",
  "iceberg.catalog.vended-credentials-enabled" = "true"
);
```

Свойства:

- `iceberg.catalog.type`: Установите это свойство в `rest`, указывая на использование REST Catalog.
- `iceberg.catalog.uri`: API endpoint для сервиса REST Catalog.
- `iceberg.catalog.security`: Установите это свойство в `jwt` для включения JWT аутентификации. Selena будет передавать информацию аутентификации текущего пользователя в Catalog.
- `iceberg.catalog.warehouse`: Указывает путь к Iceberg data warehouse или идентификатор.
- `iceberg.catalog.vended-credentials-enabled`: Установите это свойство в `true` для включения Vended Credentials и разрешения выдачи временных credentials.

Шаг 3. Предоставление прав

Когда пользователь запрашивает Catalog через Selena, права обрабатываются на двух уровнях:

1. **Права внутренних объектов Selena**

   Имеет ли пользователь право просматривать объект Catalog внутри Selena и переключать сессию на этот Catalog с помощью `SET CATALOG`.

2. **Права внутри Catalog**

   Имеет ли пользователь соответствующие права доступа при обращении к базам данных, таблицам или представлениям внутри Catalog.

В идеальном workflow Selena отвечает за управление базовыми правами, необходимыми пользователю для доступа к объектам Catalog после входа в кластер Selena. Детальные права доступа к данным внутри Catalog полностью управляются самим Catalog. Это означает:

- Пользователи должны иметь право `USAGE` на объект Catalog в Selena, чтобы они могли переключиться на catalog с помощью `SET CATALOG` или просматривать информацию о catalog с помощью `SHOW CATALOGS`.
- Проверки прав на уровне данных полностью обрабатываются Catalog.

Права внутренних объектов Catalog в Selena

Для выполнения `SHOW CATALOGS` или `SET CATALOG` в Selena пользователю необходимо право `USAGE` на соответствующий объект Catalog.

Вариант 1: Предоставление доступа всем пользователям

Если вы разрешаете всем пользователям просматривать и переключаться на указанный Catalog, вы можете предоставить право `USAGE` Catalog роли `public`.

```
GRANT USAGE ON CATALOG <catalog_name> TO ROLE public;
```

Это автоматически предоставит всем пользователям при входе в Selena роль `public` и, соответственно, право `USAGE` этого Catalog.

Вариант 2: Детальное управление правами

Если вы хотите, чтобы только определённые пользователи или группы могли просматривать и переключаться на Catalog, вы можете использовать **Group Provider** и **Role** для более детального управления:

1. Синхронизируйте информацию о внешних группах пользователей с Selena, используя Group Provider.

2. Создайте соответствующую роль Selena.

```
CREATE ROLE <role_name>;
```

3. Привяжите роль к внешней группе пользователей:

```
GRANT <role_name> TO EXTERNAL GROUP <group_name>;
```

4. Предоставьте роли право `USAGE` Catalog:

```
GRANT USAGE ON CATALOG <catalog_name> TO ROLE <role_name>;
```

В этом случае члены внешней группы автоматически унаследуют назначенную роль при входе в Selena и получат право `USAGE` Catalog.

Права на данные внутри Catalog

Настраивая свойства Catalog, проверки прав для всех объектов внутри Catalog делегируются самому Catalog. Catalog будет централизованно управлять аутентификацией и правами, упрощая конфигурацию прав в Selena, избегая избыточных авторизаций и обеспечивая согласованность правил прав между различными engines.

```
ALTER CATALOG iceberg_rest_catalog
SET PROPERTIES (
  "catalog.access.control" = "allowall"
);
```

Свойства:

- `catalog.access.control`: Установите это свойство в `allowall`. Selena не будет выполнять дополнительные проверки прав на объекты внутри Catalog. Вместо этого всё управление правами будет обрабатываться самим Catalog.

5.7.5 Hudi catalog

Hudi catalog - это тип external catalog, который позволяет вам запрашивать данные из Apache Hudi без необходимости загрузки.

Также вы можете напрямую трансформировать и загружать данные из Hudi с помощью INSERT INTO на основе Hudi catalogs. Selena поддерживает Hudi catalogs начиная с версии v1.5.2.

Чтобы обеспечить успешное выполнение SQL-операций на вашем Hudi cluster, ваш Selena cluster должен иметь доступ к системе хранения и metastore вашего Hudi cluster. Selena поддерживает следующие системы хранения и metastore:

- Distributed file system (HDFS) или object storage, такие как AWS S3, Microsoft Azure Storage, Google GCS или другие S3-совместимые системы хранения (например, MinIO)

- Metastore, такие как Hive metastore или AWS Glue

> **ПРИМЕЧАНИЕ**
>
> Если вы выберете AWS S3 в качестве хранилища, вы можете использовать HMS или AWS Glue в качестве metastore. Если вы выберете любую другую систему хранения, вы можете использовать только HMS в качестве metastore.

Примечания по использованию

- Формат файлов Hudi, который поддерживает Selena, - это Parquet. Файлы Parquet поддерживают следующие форматы сжатия: SNAPPY, LZ4, ZSTD, GZIP и NO_COMPRESSION.
- Selena обеспечивает полную поддержку таблиц Copy On Write (COW) и Merge On Read (MOR) из Hudi.

Подготовка к интеграции

Перед созданием Hudi catalog убедитесь, что ваш Selena cluster может интегрироваться с системой хранения и metastore вашего Hudi cluster.

AWS IAM

Если ваш Hudi cluster использует AWS S3 в качестве хранилища или AWS Glue в качестве metastore, выберите подходящий метод аутентификации и выполните необходимые подготовительные действия, чтобы ваш Selena cluster мог получить доступ к соответствующим облачным ресурсам AWS.

Рекомендуются следующие методы аутентификации:

- Instance profile
- Assumed role
- IAM user

Из этих трех методов аутентификации наиболее широко используется instance profile.

Для получения дополнительной информации см. Подготовка к аутентификации в AWS IAM.

HDFS

Если вы выбрали HDFS в качестве хранилища, настройте ваш Selena cluster следующим образом:

- (Опционально) Установите имя пользователя, которое используется для доступа к вашему HDFS cluster и Hive metastore. По умолчанию Selena использует имя пользователя процессов FE и BE или CN для доступа к вашему HDFS cluster и Hive metastore. Вы также можете установить имя пользователя, добавив `export HADOOP_USER_NAME="<user_name>"` в начало файла **fe/conf/hadoop_env.sh** каждого FE и в начало файла **be/conf/hadoop_env.sh** каждого BE или файла **cn/conf/hadoop_env.sh** каждого CN. После установки имени пользователя в этих файлах перезапустите каждый FE и каждый BE или CN, чтобы настройки параметров вступили в силу. Вы можете установить только одно имя пользователя для каждого Selena cluster.

- Когда вы запрашиваете данные Hudi, FE и BE или CN вашего Selena cluster используют HDFS client для доступа к вашему HDFS cluster. В большинстве случаев вам не нужно настраивать ваш Selena cluster для этой цели, и Selena запускает HDFS client с использованием конфигураций по умолчанию. Вам нужно настроить ваш Selena cluster только в следующих ситуациях:

    - High availability (HA) включена для вашего HDFS cluster: Добавьте файл **hdfs-site.xml** вашего HDFS cluster в путь **$FE_HOME/conf** каждого FE и в путь **$BE_HOME/conf** каждого BE или путь **$CN_HOME/conf** каждого CN.
    - View File System (ViewFs) включена для вашего HDFS cluster: Добавьте файл **core-site.xml** вашего HDFS cluster в путь **$FE_HOME/conf** каждого FE и в путь **$BE_HOME/conf** каждого BE или путь **$CN_HOME/conf** каждого CN.

> **ПРИМЕЧАНИЕ**
>
> Если при отправке запроса возвращается ошибка, указывающая на неизвестный хост, вы должны добавить сопоставление между именами хостов и IP-адресами узлов вашего HDFS cluster в путь **/etc/hosts**.

Аутентификация Kerberos

Если аутентификация Kerberos включена для вашего HDFS cluster или Hive metastore, настройте ваш Selena cluster следующим образом:

- Выполните команду `kinit -kt keytab_path principal` на каждом FE и каждом BE или CN для получения Ticket Granting Ticket (TGT) из Key Distribution Center (KDC). Для выполнения этой команды вы должны иметь разрешения на доступ к вашему HDFS cluster и Hive metastore. Обратите внимание, что доступ к KDC с помощью этой команды зависит от времени. Поэтому вам нужно использовать cron для периодического выполнения этой команды.
- Добавьте `JAVA_OPTS="-Djava.security.krb5.conf=/etc/krb5.conf"` в файл **$FE_HOME/conf/fe.conf** каждого FE и в файл **$BE_HOME/conf/be.conf** каждого BE или файл **$CN_HOME/conf/cn.conf** каждого CN. В этом примере `/etc/krb5.conf` - это путь сохранения файла **krb5.conf**. Вы можете изменить путь в соответствии с вашими потребностями.

Создание Hudi catalog

Синтаксис

```
CREATE EXTERNAL CATALOG <catalog_name>
[COMMENT <comment>]
PROPERTIES
(
    "type" = "hudi",
    MetastoreParams,
    StorageCredentialParams,
    MetadataUpdateParams
)
```

Параметры

catalog_name

Имя Hudi catalog. Соглашения об именовании следующие:

- Имя может содержать буквы, цифры (0-9) и подчеркивания (_). Оно должно начинаться с буквы.
- Имя чувствительно к регистру и не может превышать 1023 символа в длину.

comment

Описание Hudi catalog. Этот параметр необязателен.

type

Тип вашего источника данных. Установите значение `hudi`.

MetastoreParams

Набор параметров о том, как Selena интегрируется с metastore вашего источника данных.

Hive metastore

Если вы выбрали Hive metastore в качестве metastore вашего источника данных, настройте `MetastoreParams` следующим образом:

```
"hive.metastore.type" = "hive",
"hive.metastore.uris" = "<hive_metastore_uri>"
```

> **ПРИМЕЧАНИЕ**
>
> Перед запросом данных Hudi вы должны добавить сопоставление между именами хостов и IP-адресами узлов вашего Hive metastore в путь `/etc/hosts`. В противном случае Selena может не получить доступ к вашему Hive metastore при запуске запроса.

В следующей таблице описан параметр, который вам нужно настроить в `MetastoreParams`.

Параметр	Обязательный	Описание
hive.metastore.type	Да	Тип metastore, который вы используете для вашего Hudi cluster. Установите значение `hive`.
hive.metastore.uris	Да	URI вашего Hive metastore. Формат: `thrift://<metastore_IP_address>:<metastore_port>`. Если high availability (HA) включена для вашего Hive metastore, вы можете указать несколько URI metastore и разделить их запятыми (`,`), например, `"thrift://<metastore_IP_address_1>:<metastore_port_1>,thrift://<metastore_IP_address_2>:<metastore_port_2>,thrift://<metastore_IP_address_3>:<metastore_port_3>"`.

AWS Glue

Если вы выбрали AWS Glue в качестве metastore вашего источника данных, что поддерживается только при выборе AWS S3 в качестве хранилища, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации на основе instance profile, настройте `MetastoreParams` следующим образом:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "true",
"aws.glue.region" = "<aws_glue_region>"
```

- Чтобы выбрать метод аутентификации на основе assumed role, настройте `MetastoreParams` следующим образом:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "true",
"aws.glue.iam_role_arn" = "<iam_role_arn>",
"aws.glue.region" = "<aws_glue_region>"
```

- Чтобы выбрать метод аутентификации на основе IAM user, настройте `MetastoreParams` следующим образом:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "false",
"aws.glue.access_key" = "<iam_user_access_key>",
"aws.glue.secret_key" = "<iam_user_secret_key>",
"aws.glue.region" = "<aws_s3_region>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `MetastoreParams`.

Параметр	Обязательный	Описание
hive.metastore.type	Да	Тип metastore, который вы используете для вашего Hudi cluster. Установите значение `glue`.
aws.glue.use_instance_profile	Да	Указывает, включить ли метод аутентификации на основе instance profile и метод аутентификации на основе assumed role. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`.
aws.glue.iam_role_arn	Нет	ARN IAM роли, которая имеет привилегии на ваш AWS Glue Data Catalog. Если вы используете метод аутентификации на основе assumed role для доступа к AWS Glue, вы должны указать этот параметр.
aws.glue.region	Да	Регион, в котором находится ваш AWS Glue Data Catalog. Пример: `us-west-1`.
aws.glue.access_key	Нет	Access key вашего AWS IAM user. Если вы используете метод аутентификации на основе IAM user для доступа к AWS Glue, вы должны указать этот параметр.
aws.glue.secret_key	Нет	Secret key вашего AWS IAM user. Если вы используете метод аутентификации на основе IAM user для доступа к AWS Glue, вы должны указать этот параметр.

Для получения информации о том, как выбрать метод аутентификации для доступа к AWS Glue и как настроить политику контроля доступа в AWS IAM Console, см. Параметры аутентификации для доступа к AWS Glue.

StorageCredentialParams

Набор параметров о том, как Selena интегрируется с вашей системой хранения. Этот набор параметров необязателен.

Если вы используете HDFS в качестве хранилища, вам не нужно настраивать `StorageCredentialParams`.

Если вы используете AWS S3, другую S3-совместимую систему хранения, Microsoft Azure Storage или Google GCS в качестве хранилища, вы должны настроить `StorageCredentialParams`.

AWS S3

Если вы выбрали AWS S3 в качестве хранилища для вашего Hudi cluster, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации на основе instance profile, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.region" = "<aws_s3_region>"
```

- Чтобы выбрать метод аутентификации на основе assumed role, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.iam_role_arn" = "<iam_role_arn>",
"aws.s3.region" = "<aws_s3_region>"
```

- Чтобы выбрать метод аутентификации на основе IAM user, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "false",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>",
"aws.s3.region" = "<aws_s3_region>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
aws.s3.use_instance_profi	Да	Указывает, включить ли метод аутентификации на основе instance profile и метод аутентификации на основе assumed role. Допустимые значения: `true` и `false`.

Параметр	Обязательный	Описание
le		Значение по умолчанию: `false`.
aws.s3.iam_role_arn	Нет	ARN IAM роли, которая имеет привилегии на ваш AWS S3 bucket. Если вы используете метод аутентификации на основе assumed role для доступа к AWS S3, вы должны указать этот параметр.
aws.s3.region	Да	Регион, в котором находится ваш AWS S3 bucket. Пример: `us-west-1`.
aws.s3.access_key	Нет	Access key вашего IAM user. Если вы используете метод аутентификации на основе IAM user для доступа к AWS S3, вы должны указать этот параметр.
aws.s3.secret_key	Нет	Secret key вашего IAM user. Если вы используете метод аутентификации на основе IAM user для доступа к AWS S3, вы должны указать этот параметр.

Для получения информации о том, как выбрать метод аутентификации для доступа к AWS S3 и как настроить политику контроля доступа в AWS IAM Console, см. Параметры аутентификации для доступа к AWS S3.

S3-совместимая система хранения

Hudi catalogs поддерживают S3-совместимые системы хранения начиная с версии v1.5.2.

Если вы выбрали S3-совместимую систему хранения, такую как MinIO, в качестве хранилища для вашего Hudi cluster, настройте `StorageCredentialParams` следующим образом для обеспечения успешной интеграции:

```
"aws.s3.enable_ssl" = "false",
"aws.s3.enable_path_style_access" = "true",
"aws.s3.endpoint" = "<s3_endpoint>",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
aws.s3.enable_ssl	Да	Указывает, включить ли SSL-соединение. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`.
aws.s3.enable_path_style_access	Да	Указывает, включить ли доступ в стиле пути. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`. Для MinIO вы должны установить значение `true`. URL-адреса в стиле пути используют следующий формат: `https://s3.<region_code>.amazonaws.com/<bucket_name>/<key_name>`. Например, если вы создаете bucket с именем `DOC-EXAMPLE-BUCKET1` в регионе US West (Oregon) и хотите получить доступ к объекту `alice.jpg` в этом bucket, вы можете использовать следующий URL в стиле пути: `https://s3.us-west-2.amazonaws.com/DOC-EXAMPLE-BUCKET1/alice.jpg`.
aws.s3.endpoint	Да	Endpoint, который используется для подключения к вашей S3-совместимой системе хранения вместо AWS S3.
aws.s3.access_key	Да	Access key вашего IAM user.
aws.s3.secret_key	Да	Secret key вашего IAM user.

Microsoft Azure Storage

Hudi catalogs поддерживают Microsoft Azure Storage начиная с версии v1.5.2.

Azure Blob Storage

Если вы выбрали Blob Storage в качестве хранилища для вашего Hudi cluster, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Shared Key, настройте `StorageCredentialParams` следующим образом:

  ```
  "azure.blob.storage_account" = "<storage_account_name>",
  "azure.blob.shared_key" = "<storage_account_shared_key>"
  ```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.blob.storage_account	Да	Имя пользователя вашей учетной записи Blob Storage.
azure.blob.shared_key	Да	Shared key вашей учетной записи Blob Storage.

- Чтобы выбрать метод аутентификации SAS Token, настройте `StorageCredentialParams` следующим образом:

```
"azure.blob.storage_account" = "<storage_account_name>",
"azure.blob.container" = "<container_name>",
"azure.blob.sas_token" = "<storage_account_SAS_token>"
```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.blob.storage_account	Да	Имя пользователя вашей учетной записи Blob Storage.
azure.blob.container	Да	Имя blob container, в котором хранятся ваши данные.
azure.blob.sas_token	Да	SAS token, который используется для доступа к вашей учетной записи Blob Storage.

Azure Data Lake Storage Gen2

Если вы выбрали Data Lake Storage Gen2 в качестве хранилища для вашего Hudi cluster, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Managed Identity, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.oauth2_use_managed_identity" = "true",
"azure.adls2.oauth2_tenant_id" = "<service_principal_tenant_id>",
"azure.adls2.oauth2_client_id" = "<service_client_id>"
```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls2.oauth2_use_managed_identity	Да	Указывает, включить ли метод аутентификации Managed Identity. Установите значение `true`.
azure.adls2.oauth2_tenant_id	Да	ID tenant, данные которого вы хотите получить.
azure.adls2.oauth2_client_id	Да	Client (application) ID managed identity.

- Чтобы выбрать метод аутентификации Shared Key, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.storage_account" = "<storage_account_name>",
"azure.adls2.shared_key" = "<storage_account_shared_key>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls2.storage_account	Да	Имя пользователя вашей учетной записи хранения Data Lake Storage Gen2.
azure.adls2.shared_key	Да	Shared key вашей учетной записи хранения Data Lake Storage Gen2.

- Чтобы выбрать метод аутентификации Service Principal, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.oauth2_client_id" = "<service_client_id>",
"azure.adls2.oauth2_client_secret" = "
<service_principal_client_secret>",
"azure.adls2.oauth2_client_endpoint" = "
<service_principal_client_endpoint>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls2.oauth2_client_id	Да	Client (application) ID service principal.

Параметр	Обязательный	Описание
azure.adls2.oauth2_client_secret	Да	Значение нового созданного client (application) secret.
azure.adls2.oauth2_client_endpoint	Да	OAuth 2.0 token endpoint (v1) service principal или application.

Azure Data Lake Storage Gen1

Если вы выбрали Data Lake Storage Gen1 в качестве хранилища для вашего Hudi cluster, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Managed Service Identity, настройте `StorageCredentialParams` следующим образом:

  ```
  "azure.adls1.use_managed_service_identity" = "true"
  ```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls1.use_managed_service_identity	Да	Указывает, включить ли метод аутентификации Managed Service Identity. Установите значение `true`.

- Чтобы выбрать метод аутентификации Service Principal, настройте `StorageCredentialParams` следующим образом:

  ```
  "azure.adls1.oauth2_client_id" = "<application_client_id>",
  "azure.adls1.oauth2_credential" = "<application_client_credential>",
  "azure.adls1.oauth2_endpoint" = "<OAuth_2.0_authorization_endpoint_v2>"
  ```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls1.oauth2_client_id	Да	Client (application) ID service principal.
azure.adls1.oauth2_credential	Да	Значение нового созданного client (application) secret.

Параметр	Обязательный	Описание
azure.adls1.oauth2_endpoint	Да	OAuth 2.0 token endpoint (v1) service principal или application.

Google GCS

Hudi catalogs поддерживают Google GCS начиная с версии v1.5.2.

Если вы выбрали Google GCS в качестве хранилища для вашего Hudi cluster, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации на основе VM, настройте `StorageCredentialParams` следующим образом:

  ```
  "gcp.gcs.use_compute_engine_service_account" = "true"
  ```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.use_compute_engine_service_account	false	true	Указывает, использовать ли напрямую service account, привязанный к вашему Compute Engine.

- Чтобы выбрать метод аутентификации на основе service account, настройте `StorageCredentialParams` следующим образом:

  ```
  "gcp.gcs.service_account_email" = "<google_service_account_email>",
  "gcp.gcs.service_account_private_key_id" = "
  <google_service_private_key_id>",
  "gcp.gcs.service_account_private_key" = "<google_service_private_key>"
  ```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.service_account_email	""	"user@hello.iam.gserviceaccount.com"	Адрес электронной почты в JSON-файле, созданном при создании service account.
gcp.gcs.service_account_private_key_id	""	"61d257bd8479547cb3e04f0b9b6b9ca07af3b7ea"	ID private key в JSON-файле, созданном при создании service account.
gcp.gcs.service_account_private_key	""	"-----BEGIN PRIVATE KEY----xxxx-----END PRIVATE KEY----\n"	Private key в JSON-файле, созданном при создании service account.

- Чтобы выбрать метод аутентификации на основе impersonation, настройте `StorageCredentialParams` следующим образом:
  - Заставить VM instance имперсонировать service account:

    ```
    "gcp.gcs.use_compute_engine_service_account" = "true",
    "gcp.gcs.impersonation_service_account" = "
    <assumed_google_service_account_email>"
    ```

    В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.use_compute_engine_service_account	false	true	Указывает, использовать ли напрямую service account, привязанный к вашему Compute Engine.
gcp.gcs.impersonation_service_account	""	"hello"	Service account, который вы хотите имперсонировать.

  - Заставить service account (временно названный meta service account) имперсонировать другой service account (временно названный data service account):

```
"gcp.gcs.service_account_email" = "<google_service_account_email>",
"gcp.gcs.service_account_private_key_id" = "
<meta_google_service_account_email>",
"gcp.gcs.service_account_private_key" = "
<meta_google_service_account_email>",
"gcp.gcs.impersonation_service_account" = "
<data_google_service_account_email>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.service_account_email	""	"user@hello.iam.gserviceaccount.com"	Адрес электронной почты в JSON-файле, созданном при создании meta service account.
gcp.gcs.service_account_private_key_id	""	"61d257bd8479547cb3e04f0b9b6b9ca07af3b7ea"	ID private key в JSON-файле, созданном при создании meta service account.
gcp.gcs.service_account_private_key	""	"-----BEGIN PRIVATE KEY-----xxxx-----END PRIVATE KEY-----\n"	Private key в JSON-файле, созданном при создании meta service account.
gcp.gcs.impersonation_service_account	""	"hello"	Data service account, который вы хотите имперсонировать.

MetadataUpdateParams

Набор параметров о том, как Selena обновляет кэшированные метаданные Hudi. Этот набор параметров необязателен.

Selena реализует политику автоматического асинхронного обновления по умолчанию.

В большинстве случаев вы можете игнорировать `MetadataUpdateParams` и не настраивать параметры политики в нем, потому что значения по умолчанию этих параметров уже обеспечивают готовую к использованию производительность.

Однако, если частота обновления данных в Hudi высока, вы можете настроить эти параметры для дальнейшей оптимизации производительности автоматических

асинхронных обновлений.

> **ПРИМЕЧАНИЕ**
>
> В большинстве случаев, если ваши данные Hudi обновляются с детализацией 1 час или меньше, частота обновления данных считается высокой.

Параметр	Обязательный	Описание
enable_metastore_cache	Нет	Указывает, кэширует ли Selena метаданные таблиц Hudi. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`. Значение `true` включает кэш, а значение `false` отключает кэш.
enable_remote_file_cache	Нет	Указывает, кэширует ли Selena метаданные базовых файлов данных таблиц или partitions Hudi. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`. Значение `true` включает кэш, а значение `false` отключает кэш.
metastore_cache_refresh_interval_sec	Нет	Интервал времени, с которым Selena асинхронно обновляет метаданные таблиц или partitions Hudi, кэшированные в себе. Единица измерения: секунды. Значение по умолчанию: `7200`, что составляет 2 часа.
remote_file_cache_refresh_interval_sec	Нет	Интервал времени, с которым Selena асинхронно обновляет метаданные базовых файлов данных таблиц или partitions Hudi, кэшированные в себе. Единица измерения: секунды. Значение по умолчанию: `60`.
metastore_cache_ttl_sec	Нет	Интервал времени, с которым Selena автоматически отбрасывает метаданные таблиц или partitions Hudi, кэшированные в себе. Единица измерения: секунды. Значение по умолчанию: `86400`, что составляет 24 часа.
remote_file_cache_ttl_sec	Нет	Интервал времени, с которым Selena автоматически отбрасывает метаданные базовых файлов данных таблиц или partitions Hudi, кэшированные в себе. Единица измерения: секунды. Значение по умолчанию: `129600`, что составляет 36 часов.

Примеры

Следующие примеры создают Hudi catalog с именем `hudi_catalog_hms` или `hudi_catalog_glue`, в зависимости от типа metastore, который вы используете, для запроса данных из вашего Hudi cluster.

HDFS

Если вы используете HDFS в качестве хранилища, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
);
```

AWS S3

Если вы выбираете учетные данные на основе instance profile

- Если вы используете Hive metastore в вашем Hudi cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2"
);
```

- Если вы используете AWS Glue в вашем Amazon EMR Hudi cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_glue
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2"
);
```

Если вы выбираете учетные данные на основе assumed role

- Если вы используете Hive metastore в вашем Hudi cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

- Если вы используете AWS Glue в вашем Amazon EMR Hudi cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_glue
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_glue_role",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

Если вы выбираете учетные данные на основе IAM user

- Если вы используете Hive metastore в вашем Hudi cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_access_key>",
    "aws.s3.region" = "us-west-2"
);
```

- Если вы используете AWS Glue в вашем Amazon EMR Hudi cluster, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_glue
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "false",
    "aws.glue.access_key" = "<iam_user_access_key>",
    "aws.glue.secret_key" = "<iam_user_secret_key>",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_secret_key>",
    "aws.s3.region" = "us-west-2"
);
```

S3-совместимая система хранения

Используйте MinIO в качестве примера. Выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.enable_ssl" = "true",
    "aws.s3.enable_path_style_access" = "true",
    "aws.s3.endpoint" = "<s3_endpoint>",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_secret_key>"
);
```

Microsoft Azure Storage

Azure Blob Storage

- Если вы выберете метод аутентификации Shared Key, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.blob.storage_account" = "<blob_storage_account_name>",
    "azure.blob.shared_key" = "<blob_storage_account_shared_key>"
);
```

- Если вы выберете метод аутентификации SAS Token, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.blob.storage_account" = "<blob_storage_account_name>",
    "azure.blob.container" = "<blob_container_name>",
    "azure.blob.sas_token" = "<blob_storage_account_SAS_token>"
);
```

Azure Data Lake Storage Gen1

- Если вы выберете метод аутентификации Managed Service Identity, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls1.use_managed_service_identity" = "true"
);
```

- Если вы выберете метод аутентификации Service Principal, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls1.oauth2_client_id" = "<application_client_id>",
    "azure.adls1.oauth2_credential" = "
<application_client_credential>",
    "azure.adls1.oauth2_endpoint" = "
<OAuth_2.0_authorization_endpoint_v2>"
);
```

Azure Data Lake Storage Gen2

- Если вы выберете метод аутентификации Managed Identity, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls2.oauth2_use_managed_identity" = "true",
    "azure.adls2.oauth2_tenant_id" = "<service_principal_tenant_id>",
    "azure.adls2.oauth2_client_id" = "<service_client_id>"
);
```

- Если вы выберете метод аутентификации Shared Key, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls2.storage_account" = "<storage_account_name>",
    "azure.adls2.shared_key" = "<shared_key>"
);
```

- Если вы выберете метод аутентификации Service Principal, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls2.oauth2_client_id" = "<service_client_id>",
    "azure.adls2.oauth2_client_secret" = "
<service_principal_client_secret>",
    "azure.adls2.oauth2_client_endpoint" = "
<service_principal_client_endpoint>"
);
```

Google GCS

- Если вы выберете метод аутентификации на основе VM, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "gcp.gcs.use_compute_engine_service_account" = "true"
);
```

- Если вы выберете метод аутентификации на основе service account, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "gcp.gcs.service_account_email" = "<google_service_account_email>",
    "gcp.gcs.service_account_private_key_id" = "
<google_service_private_key_id>",
    "gcp.gcs.service_account_private_key" = "
<google_service_private_key>"
);
```

- Если вы выберете метод аутентификации на основе impersonation:
    - Если вы заставляете VM instance имперсонировать service account, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "gcp.gcs.use_compute_engine_service_account" = "true",
    "gcp.gcs.impersonation_service_account" = "
<assumed_google_service_account_email>"
);
```

    - Если вы заставляете service account имперсонировать другой service account, выполните команду, подобную приведенной ниже:

```
CREATE EXTERNAL CATALOG hudi_catalog_hms
PROPERTIES
(
    "type" = "hudi",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "gcp.gcs.service_account_email" = "
<google_service_account_email>",
    "gcp.gcs.service_account_private_key_id" = "
<meta_google_service_account_email>",
    "gcp.gcs.service_account_private_key" = "
<meta_google_service_account_email>",
    "gcp.gcs.impersonation_service_account" = "
<data_google_service_account_email>"
);
```

Просмотр Hudi catalogs

Вы можете использовать SHOW CATALOGS для запроса всех catalogs в текущем Selena cluster:

```
SHOW CATALOGS;
```

Вы также можете использовать SHOW CREATE CATALOG для запроса инструкции создания external catalog. Следующий пример запрашивает инструкцию создания Hudi catalog с именем `hudi_catalog_glue`:

```
SHOW CREATE CATALOG hudi_catalog_glue;
```

Переключение на Hudi Catalog и базу данных в нем

Вы можете использовать один из следующих методов для переключения на Hudi catalog и базу данных в нем:

- Используйте SET CATALOG для указания Hudi catalog в текущей сессии, а затем используйте USE для указания активной базы данных:

  ```
  -- Переключитесь на указанный catalog в текущей сессии:
  SET CATALOG <catalog_name>
  -- Укажите активную базу данных в текущей сессии:
  USE <db_name>
  ```

- Непосредственно используйте USE для переключения на Hudi catalog и базу данных в нем:

  ```
  USE <catalog_name>.<db_name>
  ```

Удаление Hudi catalog

Вы можете использовать DROP CATALOG для удаления external catalog.

Следующий пример удаляет Hudi catalog с именем `hudi_catalog_glue`:

```
DROP Catalog hudi_catalog_glue;
```

Просмотр схемы таблицы Hudi

Вы можете использовать один из следующих синтаксисов для просмотра схемы таблицы Hudi:

- Просмотр схемы

```
DESC[RIBE] <catalog_name>.<database_name>.<table_name>
```

- Просмотр схемы и расположения из инструкции CREATE

```
SHOW CREATE TABLE <catalog_name>.<database_name>.<table_name>
```

Запрос таблицы Hudi

1. Используйте SHOW DATABASES для просмотра баз данных в вашем Hudi cluster:

```
SHOW DATABASES FROM <catalog_name>
```

2. Переключитесь на Hudi Catalog и базу данных в нем.

3. Используйте SELECT для запроса целевой таблицы в указанной базе данных:

```
SELECT count(*) FROM <table_name> LIMIT 10
```

Загрузка данных из Hudi

Предположим, у вас есть OLAP-таблица с именем `olap_tbl`, вы можете преобразовать и загрузить данные следующим образом:

```
INSERT INTO default_catalog.olap_db.olap_tbl SELECT * FROM hudi_table
```

Ручное или автоматическое обновление кэша метаданных

Ручное обновление

По умолчанию Selena кэширует метаданные Hudi и автоматически обновляет метаданные в асинхронном режиме для повышения производительности. Кроме того, после внесения некоторых изменений схемы или обновлений таблицы в таблицу Hudi, вы также можете использовать REFRESH EXTERNAL TABLE для ручного обновления ее метаданных, тем самым гарантируя, что Selena сможет получить актуальные метаданные при первой возможности и сгенерировать соответствующие планы выполнения:

```
REFRESH EXTERNAL TABLE <table_name> [PARTITION ('partition_name', ...)]
```

Приложение: Понимание автоматического асинхронного обновления метаданных

Автоматическое асинхронное обновление - это политика по умолчанию, которую Selena использует для обновления метаданных в Hudi catalogs.

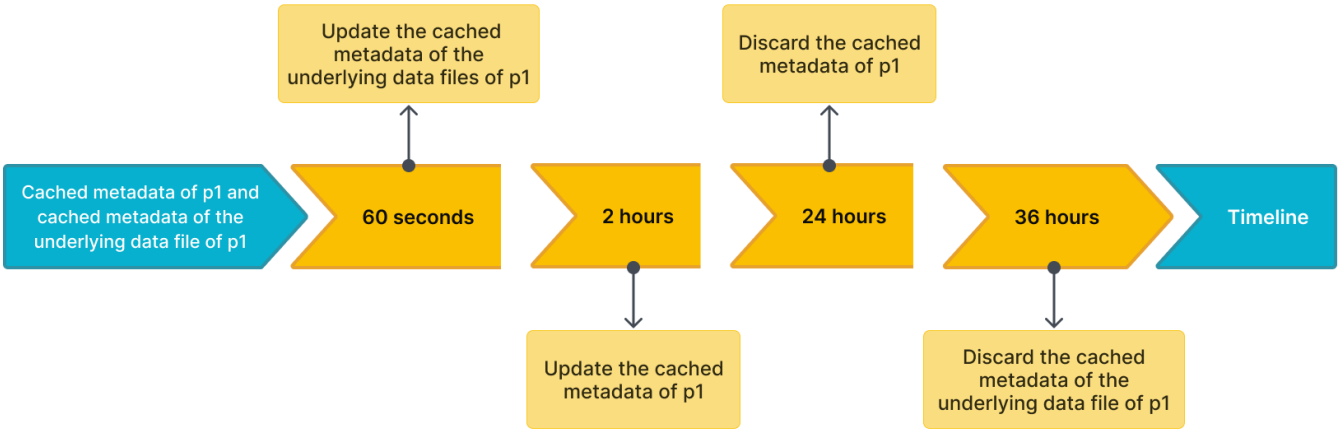
По умолчанию (а именно, когда параметры `enable_metastore_cache` и `enable_remote_file_cache` оба установлены в `true`), если запрос попадает в partition таблицы Hudi, Selena автоматически кэширует метаданные partition и метаданные базовых файлов данных partition. Кэшированные метаданные обновляются с использованием политики ленивого обновления.

Например, есть таблица Hudi с именем `table2`, которая имеет четыре partitions: `p1`, `p2`, `p3` и `p4`. Запрос попадает в `p1`, и Selena кэширует метаданные `p1` и метаданные базовых файлов данных `p1`. Предположим, что интервалы времени по умолчанию для обновления и отбрасывания кэшированных метаданных следующие:

- Интервал времени (указанный параметром `metastore_cache_refresh_interval_sec`) для асинхронного обновления кэшированных метаданных `p1` составляет 2 часа.
- Интервал времени (указанный параметром `remote_file_cache_refresh_interval_sec`) для асинхронного обновления кэшированных метаданных базовых файлов данных `p1` составляет 60 секунд.
- Интервал времени (указанный параметром `metastore_cache_ttl_sec`) для автоматического отбрасывания кэшированных метаданных `p1` составляет 24 часа.

- Интервал времени (указанный параметром `remote_file_cache_ttl_sec`) для автоматического отбрасывания кэшированных метаданных базовых файлов данных `p1` составляет 36 часов.

На следующем рисунке показаны интервалы времени на временной шкале для более легкого понимания.



Затем Selena обновляет или отбрасывает метаданные в соответствии со следующими правилами:

- Если другой запрос снова попадает в `p1` и текущее время с момента последнего обновления составляет менее 60 секунд, Selena не обновляет кэшированные метаданные `p1` или кэшированные метаданные базовых файлов данных `p1`.
- Если другой запрос снова попадает в `p1` и текущее время с момента последнего обновления составляет более 60 секунд, Selena обновляет кэшированные метаданные базовых файлов данных `p1`.
- Если другой запрос снова попадает в `p1` и текущее время с момента последнего обновления составляет более 2 часов, Selena обновляет кэшированные метаданные `p1`.
- Если к `p1` не было доступа в течение 24 часов с момента последнего обновления, Selena отбрасывает кэшированные метаданные `p1`. Метаданные будут кэшированы при следующем запросе.
- Если к `p1` не было доступа в течение 36 часов с момента последнего обновления, Selena отбрасывает кэшированные метаданные базовых файлов данных `p1`. Метаданные будут кэшированы при следующем запросе.

5.7.6 Delta Lake catalog

Delta Lake catalog — это разновидность внешнего каталога, который позволяет выполнять запросы к данным Delta Lake без их загрузки.

Также вы можете напрямую трансформировать и загружать данные из Delta Lake с помощью INSERT INTO на основе каталогов Delta Lake. Selena поддерживает каталоги Delta Lake начиная с версии v1.5.2.

Чтобы обеспечить успешное выполнение SQL-запросов в вашем cluster Delta Lake, ваш cluster Selena должен иметь доступ к системе хранения и metastore вашего cluster Delta Lake. Selena поддерживает следующие системы хранения и metastore:

- Распределённая файловая система (HDFS) или объектное хранилище, такое как AWS S3, Microsoft Azure Storage, Google GCS или другие S3-совместимые системы хранения (например, MinIO)
- Metastore, такой как Hive metastore или AWS Glue

> **ПРИМЕЧАНИЕ**
>
> Если вы выбрали AWS S3 в качестве хранилища, вы можете использовать HMS или AWS Glue в качестве metastore. Если вы выбрали любую другую систему хранения, вы можете использовать только HMS в качестве metastore.

Примечания по использованию

- Формат файлов Delta Lake, который поддерживает Selena — это Parquet. Файлы Parquet поддерживают следующие форматы сжатия: SNAPPY, LZ4, ZSTD, GZIP и NO_COMPRESSION.
- Типы данных Delta Lake, которые Selena не поддерживает — это MAP и STRUCT.

Подготовка к интеграции

Перед созданием Delta Lake catalog убедитесь, что ваш cluster Selena может интегрироваться с системой хранения и metastore вашего cluster Delta Lake.

AWS IAM

Если ваш cluster Delta Lake использует AWS S3 в качестве хранилища или AWS Glue в качестве metastore, выберите подходящий метод аутентификации и выполните необходимую подготовку, чтобы убедиться, что ваш cluster Selena может получить доступ к соответствующим облачным ресурсам AWS.

Рекомендуются следующие методы аутентификации:

- Instance profile
- Assumed role
- IAM user

Из трёх вышеупомянутых методов аутентификации наиболее широко используется instance profile.

Для получения дополнительной информации см. Подготовка к аутентификации в AWS IAM.

HDFS

Если вы выбираете HDFS в качестве хранилища, настройте ваш cluster Selena следующим образом:

- (Опционально) Установите имя пользователя, которое используется для доступа к вашему cluster HDFS и Hive metastore. По умолчанию Selena использует имя пользователя процессов FE и BE или CN для доступа к вашему cluster HDFS и Hive metastore. Вы также можете установить имя пользователя, добавив `export HADOOP_USER_NAME="<user_name>"` в начало файла **fe/conf/hadoop_env.sh** каждого FE и в начало файла **be/conf/hadoop_env.sh** каждого BE или файла **cn/conf/hadoop_env.sh** каждого CN. После установки имени пользователя в этих файлах перезапустите каждый FE и каждый BE или CN, чтобы настройки параметров вступили в силу. Вы можете установить только одно имя пользователя для каждого cluster Selena.

- При запросе данных Delta Lake узлы FE и BE или CN вашего cluster Selena используют HDFS-клиент для доступа к вашему cluster HDFS. В большинстве случаев вам не нужно настраивать ваш cluster Selena для достижения этой цели, и Selena

запускает HDFS-клиент с использованием настроек по умолчанию. Вам нужно настроить ваш cluster Selena только в следующих ситуациях:

- Для вашего cluster HDFS включена высокая доступность (HA): Добавьте файл **hdfs-site.xml** вашего cluster HDFS в путь **$FE_HOME/conf** каждого FE и в путь **$BE_HOME/conf** каждого BE или путь **$CN_HOME/conf** каждого CN.
- Для вашего cluster HDFS включена View File System (ViewFs): Добавьте файл **core-site.xml** вашего cluster HDFS в путь **$FE_HOME/conf** каждого FE и в путь **$BE_HOME/conf** каждого BE или путь **$CN_HOME/conf** каждого CN.

> **ПРИМЕЧАНИЕ**
>
> Если при отправке запроса возвращается ошибка, указывающая на неизвестный хост, вы должны добавить сопоставление между именами хостов и IP-адресами узлов вашего cluster HDFS в путь **/etc/hosts**.

Аутентификация Kerberos

Если для вашего cluster HDFS или Hive metastore включена аутентификация Kerberos, настройте ваш cluster Selena следующим образом:

- Выполните команду `kinit -kt keytab_path principal` на каждом FE и каждом BE или CN, чтобы получить Ticket Granting Ticket (TGT) от Key Distribution Center (KDC). Для выполнения этой команды у вас должны быть разрешения на доступ к вашему cluster HDFS и Hive metastore. Обратите внимание, что доступ к KDC с помощью этой команды зависит от времени. Поэтому вам нужно использовать cron для периодического выполнения этой команды.
- Добавьте `JAVA_OPTS="-Djava.security.krb5.conf=/etc/krb5.conf"` в файл **$FE_HOME/conf/fe.conf** каждого FE и в файл **$BE_HOME/conf/be.conf** каждого BE или файл **$CN_HOME/conf/cn.conf** каждого CN. В этом примере `/etc/krb5.conf` является путём сохранения файла **krb5.conf**. Вы можете изменить путь в соответствии с вашими потребностями.

Создание Delta Lake catalog

Синтаксис

```
CREATE EXTERNAL CATALOG <catalog_name>
[COMMENT <comment>]
PROPERTIES
(
    "type" = "deltalake",
    MetastoreParams,
    StorageCredentialParams,
    MetadataUpdateParams
)
```

Параметры

catalog_name

Имя Delta Lake catalog. Соглашения об именовании следующие:

- Имя может содержать буквы, цифры (0-9) и символы подчёркивания (_). Оно должно начинаться с буквы.
- Имя чувствительно к регистру и не может превышать 1023 символа в длину.

comment

Описание Delta Lake catalog. Этот параметр является опциональным.

type

Тип вашего источника данных. Установите значение `deltalake`.

MetastoreParams

Набор параметров о том, как Selena интегрируется с metastore вашего источника данных.

Hive metastore

Если вы выбираете Hive metastore в качестве metastore для вашего источника данных, настройте `MetastoreParams` следующим образом:

```
"hive.metastore.type" = "hive",
"hive.metastore.uris" = "<hive_metastore_uri>"
```

> **ПРИМЕЧАНИЕ**
>
> Перед запросом данных Delta Lake вы должны добавить сопоставление между именами хостов и IP-адресами узлов вашего Hive metastore в путь `/etc/hosts`. В противном случае Selena может не получить доступ к вашему Hive metastore при запуске запроса.

В следующей таблице описаны параметры, которые вам нужно настроить в `MetastoreParams`.

Параметр	Обязательный	Описание
hive.metastore.type	Да	Тип metastore, который вы используете для вашего cluster Delta Lake. Установите значение `hive`.
hive.metastore.uris	Да	URI вашего Hive metastore. Формат: `thrift://<metastore_IP_address>:<metastore_port>`. Если для вашего Hive metastore включена высокая доступность (HA), вы можете указать несколько URI metastore и разделить их запятыми (`,`), например, `"thrift://<metastore_IP_address_1>:<metastore_port_1>,thrift://<metastore_IP_address_2>:<metastore_port_2>,thrift://<metastore_IP_address_3>:<metastore_port_3>"`.

AWS Glue

Если вы выбираете AWS Glue в качестве metastore для вашего источника данных, что поддерживается только при выборе AWS S3 в качестве хранилища, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации на основе instance profile, настройте `MetastoreParams` следующим образом:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "true",
"aws.glue.region" = "<aws_glue_region>"
```

- Чтобы выбрать метод аутентификации на основе assumed role, настройте `MetastoreParams` следующим образом:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "true",
"aws.glue.iam_role_arn" = "<iam_role_arn>",
"aws.glue.region" = "<aws_glue_region>"
```

- Чтобы выбрать метод аутентификации на основе IAM user, настройте `MetastoreParams` следующим образом:

```
"hive.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "false",
"aws.glue.access_key" = "<iam_user_access_key>",
"aws.glue.secret_key" = "<iam_user_secret_key>",
"aws.glue.region" = "<aws_s3_region>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `MetastoreParams`.

Параметр	Обязательный	Описание
hive.metastore.type	Да	Тип metastore, который вы используете для вашего cluster Delta Lake. Установите значение `glue`.
aws.glue.use_instance_profile	Да	Указывает, включать ли метод аутентификации на основе instance profile и метод аутентификации на основе assumed role. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`.
aws.glue.iam_role_arn	Нет	ARN роли IAM, которая имеет привилегии на ваш AWS Glue Data Catalog. Если вы используете метод аутентификации на основе assumed role для доступа к AWS Glue, вы должны указать этот параметр.
aws.glue.region	Да	Регион, в котором находится ваш AWS Glue Data Catalog. Пример: `us-west-1`.
aws.glue.access_key	Нет	Ключ доступа вашего пользователя AWS IAM. Если вы используете метод аутентификации на основе IAM user для доступа к AWS Glue, вы должны указать этот параметр.
aws.glue.secret_key	Нет	Секретный ключ вашего пользователя AWS IAM. Если вы используете метод аутентификации на основе IAM user для доступа к AWS Glue, вы должны указать этот параметр.

Для получения информации о том, как выбрать метод аутентификации для доступа к AWS Glue и как настроить политику контроля доступа в консоли AWS IAM, см. Параметры

аутентификации для доступа к AWS Glue.

StorageCredentialParams

Набор параметров о том, как Selena интегрируется с вашей системой хранения. Этот набор параметров является опциональным.

Если вы используете HDFS в качестве хранилища, вам не нужно настраивать `StorageCredentialParams`.

Если вы используете AWS S3, другую S3-совместимую систему хранения, Microsoft Azure Storage или Google GCS в качестве хранилища, вы должны настроить `StorageCredentialParams`.

AWS S3

Если вы выбираете AWS S3 в качестве хранилища для вашего cluster Delta Lake, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации на основе instance profile, настройте `StorageCredentialParams` следующим образом:

  ```
  "aws.s3.use_instance_profile" = "true",
  "aws.s3.region" = "<aws_s3_region>"
  ```

- Чтобы выбрать метод аутентификации на основе assumed role, настройте `StorageCredentialParams` следующим образом:

  ```
  "aws.s3.use_instance_profile" = "true",
  "aws.s3.iam_role_arn" = "<iam_role_arn>",
  "aws.s3.region" = "<aws_s3_region>"
  ```

- Чтобы выбрать метод аутентификации на основе IAM user, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "false",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>",
"aws.s3.region" = "<aws_s3_region>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
aws.s3.use_instance_profile	Да	Указывает, включать ли метод аутентификации на основе instance profile и метод аутентификации на основе assumed role. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`.
aws.s3.iam_role_arn	Нет	ARN роли IAM, которая имеет привилегии на ваш bucket AWS S3. Если вы используете метод аутентификации на основе assumed role для доступа к AWS S3, вы должны указать этот параметр.
aws.s3.region	Да	Регион, в котором находится ваш bucket AWS S3. Пример: `us-west-1`.
aws.s3.access_key	Нет	Ключ доступа вашего пользователя IAM. Если вы используете метод аутентификации на основе IAM user для доступа к AWS S3, вы должны указать этот параметр.
aws.s3.secret_key	Нет	Секретный ключ вашего пользователя IAM. Если вы используете метод аутентификации на основе IAM user для доступа к AWS S3, вы должны указать этот параметр.

Для получения информации о том, как выбрать метод аутентификации для доступа к AWS S3 и как настроить политику контроля доступа в консоли AWS IAM, см. Параметры аутентификации для доступа к AWS S3.

S3-совместимая система хранения

Delta Lake catalogs поддерживают S3-совместимые системы хранения начиная с версии v1.5.2.

Если вы выбираете S3-совместимую систему хранения, такую как MinIO, в качестве хранилища для вашего cluster Delta Lake, настройте `StorageCredentialParams` следующим образом для обеспечения успешной интеграции:

```
"aws.s3.enable_ssl" = "false",
"aws.s3.enable_path_style_access" = "true",
"aws.s3.endpoint" = "<s3_endpoint>",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
aws.s3.enable_ssl	Да	Указывает, включать ли SSL-соединение. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`.
aws.s3.enable_path_style_access	Да	Указывает, включать ли доступ в стиле path. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`. Для MinIO вы должны установить значение `true`. URL в стиле path используют следующий формат: `https://s3.<region_code>.amazonaws.com/<bucket_name>/<key_name>`. Например, если вы создаёте bucket с именем `DOC-EXAMPLE-BUCKET1` в регионе US West (Oregon) и хотите получить доступ к объекту `alice.jpg` в этом bucket, вы можете использовать следующий URL в стиле path: `https://s3.us-west-2.amazonaws.com/DOC-EXAMPLE-BUCKET1/alice.jpg`.
aws.s3.endpoint	Да	Конечная точка, которая используется для подключения к вашей S3-совместимой системе хранения вместо AWS S3.
aws.s3.access_key	Да	Ключ доступа вашего пользователя IAM.
aws.s3.secret_key	Да	Секретный ключ вашего пользователя IAM.

Microsoft Azure Storage

Delta Lake catalogs поддерживают Microsoft Azure Storage начиная с версии v1.5.2.

Azure Blob Storage

Если вы выбираете Blob Storage в качестве хранилища для вашего cluster Delta Lake, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Shared Key, настройте `StorageCredentialParams` следующим образом:

```
"azure.blob.storage_account" = "<storage_account_name>",
"azure.blob.shared_key" = "<storage_account_shared_key>"
```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.blob.storage_account	Да	Имя пользователя вашей учётной записи Blob Storage.
azure.blob.shared_key	Да	Общий ключ вашей учётной записи Blob Storage.

- Чтобы выбрать метод аутентификации SAS Token, настройте `StorageCredentialParams` следующим образом:

```
"azure.blob.storage_account" = "<storage_account_name>",
"azure.blob.container" = "<container_name>",
"azure.blob.sas_token" = "<storage_account_SAS_token>"
```

  В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.blob.storage_account	Да	Имя пользователя вашей учётной записи Blob Storage.
azure.blob.container	Да	Имя blob-контейнера, в котором хранятся ваши данные.
azure.blob.sas_token	Да	SAS-токен, который используется для доступа к вашей учётной записи Blob Storage.

Azure Data Lake Storage Gen2

Если вы выбираете Data Lake Storage Gen2 в качестве хранилища для вашего cluster Delta Lake, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Managed Identity, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.oauth2_use_managed_identity" = "true",
"azure.adls2.oauth2_tenant_id" = "<service_principal_tenant_id>",
"azure.adls2.oauth2_client_id" = "<service_client_id>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls2.oauth2_use_managed_identity	Да	Указывает, включать ли метод аутентификации Managed Identity. Установите значение `true`.
azure.adls2.oauth2_tenant_id	Да	Идентификатор арендатора, к данным которого вы хотите получить доступ.
azure.adls2.oauth2_client_id	Да	Идентификатор клиента (приложения) управляемого удостоверения.

- Чтобы выбрать метод аутентификации Shared Key, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.storage_account" = "<storage_account_name>",
"azure.adls2.shared_key" = "<storage_account_shared_key>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls2.storage_account	Да	Имя пользователя вашей учётной записи хранения Data Lake Storage Gen2.
azure.adls2.shared_key	Да	Общий ключ вашей учётной записи хранения Data Lake Storage Gen2.

- Чтобы выбрать метод аутентификации Service Principal, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.oauth2_client_id" = "<service_client_id>",
"azure.adls2.oauth2_client_secret" = "
<service_principal_client_secret>",
"azure.adls2.oauth2_client_endpoint" = "
<service_principal_client_endpoint>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls2.oauth2_client_id	Да	Идентификатор клиента (приложения) service principal.
azure.adls2.oauth2_client_secret	Да	Значение созданного нового секрета клиента (приложения).
azure.adls2.oauth2_client_endpoint	Да	Конечная точка токена OAuth 2.0 (v1) service principal или приложения.

Azure Data Lake Storage Gen1

Если вы выбираете Data Lake Storage Gen1 в качестве хранилища для вашего cluster Delta Lake, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Managed Service Identity, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls1.use_managed_service_identity" = "true"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls1.use_managed_service_identity	Да	Указывает, включать ли метод аутентификации Managed Service Identity. Установите значение `true`.

- Чтобы выбрать метод аутентификации Service Principal, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls1.oauth2_client_id" = "<application_client_id>",
"azure.adls1.oauth2_credential" = "<application_client_credential>",
"azure.adls1.oauth2_endpoint" = "<OAuth_2.0_authorization_endpoint_v2>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls1.oauth2_client_id	Да	Идентификатор клиента (приложения) service principal.
azure.adls1.oauth2_credential	Да	Значение созданного нового секрета клиента (приложения).
azure.adls1.oauth2_endpoint	Да	Конечная точка токена OAuth 2.0 (v1) service principal или приложения.

Google GCS

Delta Lake catalogs поддерживают Google GCS начиная с версии v1.5.2.

Если вы выбираете Google GCS в качестве хранилища для вашего cluster Delta Lake, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации на основе VM, настройте `StorageCredentialParams` следующим образом:

```
"gcp.gcs.use_compute_engine_service_account" = "true"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.use_compute_engine_service_account	false	true	Указывает, использовать ли напрямую учётную запись службы, привязанную к вашему Compute Engine.

- Чтобы выбрать метод аутентификации на основе service account, настройте `StorageCredentialParams` следующим образом:

```
"gcp.gcs.service_account_email" = "<google_service_account_email>",
"gcp.gcs.service_account_private_key_id" = "
<google_service_private_key_id>",
"gcp.gcs.service_account_private_key" = "<google_service_private_key>",
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.service_account_email	""	`"user@hello.iam.gserviceaccount.com"`	Адрес электронной почты в JSON-файле, созданном при создании учётной записи службы.
gcp.gcs.service_account_private_key_id	""	"61d257bd8479547cb3e04f0b9b6b9ca07af3b7ea"	Идентификатор приватного ключа в JSON-файле, созданном при создании учётной записи службы.
gcp.gcs.service_account_private_key	""	"-----BEGIN PRIVATE KEY-----xxxx-----END PRIVATE KEY-----\n"	Приватный ключ в JSON-файле, созданном при создании учётной записи службы.

- Чтобы выбрать метод аутентификации на основе impersonation, настройте `StorageCredentialParams` следующим образом:
  - Сделать так, чтобы экземпляр VM выдавал себя за учётную запись службы:

    ```
    "gcp.gcs.use_compute_engine_service_account" = "true",
    "gcp.gcs.impersonation_service_account" = "
    <assumed_google_service_account_email>"
    ```

    В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.use_compute_engine_service_account	false	true	Указывает, использовать ли напрямую учётную запись службы, привязанную к вашему Compute Engine.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.impersonation_service_account	""	"hello"	Учётная запись службы, за которую вы хотите выдать себя.

- Сделать так, чтобы учётная запись службы (временно названная meta service account) выдавала себя за другую учётную запись службы (временно названную data service account):

```
"gcp.gcs.service_account_email" = "<google_service_account_email>",
"gcp.gcs.service_account_private_key_id" = "
<meta_google_service_account_email>",
"gcp.gcs.service_account_private_key" = "
<meta_google_service_account_email>",
"gcp.gcs.impersonation_service_account" = "
<data_google_service_account_email>"
```

В следующей таблице описаны параметры, которые вам нужно настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.service_account_email	""	`"user@hello.iam.gserviceaccount.com"`	Адрес электронной почты в JSON-файле, созданном при создании meta service account.
gcp.gcs.service_account_private_key_id	""	"61d257bd8479547cb3e04f0b9b6b9ca07af3b7ea"	Идентификатор приватного ключа в JSON-файле, созданном при создании meta service account.
gcp.gcs.service_account_private_key	""	"-----BEGIN PRIVATE KEY-----xxxx-----END PRIVATE KEY-----\n"	Приватный ключ в JSON-файле, созданном при создании meta service account.
gcp.gcs.impersonation_service_account	""	"hello"	Data service account, за которую вы хотите выдать себя.

MetadataUpdateParams

Набор параметров о том, как Selena обновляет кешированные метаданные Delta Lake. Этот набор параметров является опциональным.

Начиная с версии v1.5.2, Delta Lake Catalog поддерживает Локальное кеширование и получение метаданных. В большинстве случаев вы можете игнорировать `MetadataUpdateParams` и вам не нужно настраивать параметры политики в нём, потому что значения этих параметров по умолчанию уже обеспечивают производительность "из коробки".

Однако, если частота обновления данных в Delta Lake высока, вы можете настроить эти параметры для дальнейшей оптимизации производительности автоматических асинхронных обновлений.

> **ПРИМЕЧАНИЕ**
>
> В большинстве случаев, если ваши данные Delta Lake обновляются с детализацией 1 час или меньше, частота обновления данных считается высокой.

Параметр	Единица измерения	По умолчанию	Описание
enable_deltalake_table_cache	-	true	Включать ли Table Cache в кеше метаданных для Delta Lake.
enable_deltalake_json_meta_cache	-	true	Включать ли кеш для JSON-файлов Delta Log.
deltalake_json_meta_cache_ttl_sec	Секунда	48 * 60 * 60	Время жизни (TTL) для кеша JSON-файлов Delta Log.
deltalake_json_meta_cache_memory_usage_ratio	-	0.1	Максимальное соотношение размера JVM Heap, занимаемого кешем JSON-файлов Delta Log.
enable_deltalake_checkpoint_meta_cache	-	true	Включать ли кеш для файлов Checkpoint Delta Log.
deltalake_checkpoint_meta_cache_ttl_sec	Секунда	48 * 60 * 60	Время жизни (TTL) для кеша файлов Checkpoint Delta Log.
deltalake_checkpoint_meta_cache_memory_usage_ratio	-	0.1	Максимальное соотношение размера JVM Heap, занимаемого кешем файлов Checkpoint Delta Log.

Примеры

Следующие примеры создают Delta Lake catalog с именем `deltalake_catalog_hms` или `deltalake_catalog_glue`, в зависимости от типа metastore, который вы используете, для запроса данных из вашего cluster Delta Lake.

HDFS

Если вы используете HDFS в качестве хранилища, выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
);
```

AWS S3

Если вы выбираете учётные данные на основе instance profile

- Если вы используете Hive metastore в вашем cluster Delta Lake, выполните команду, подобную приведённой ниже:

  ```
  CREATE EXTERNAL CATALOG deltalake_catalog_hms
  PROPERTIES
  (
      "type" = "deltalake",
      "hive.metastore.type" = "hive",
      "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
      "aws.s3.use_instance_profile" = "true",
      "aws.s3.region" = "us-west-2"
  );
  ```

- Если вы используете AWS Glue в вашем cluster Delta Lake Amazon EMR, выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_glue
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2"
);
```

If you choose assumed role-based credential

- Если вы используете Hive metastore в вашем cluster Delta Lake, выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

- Если вы используете AWS Glue в вашем cluster Delta Lake Amazon EMR, выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_glue
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_glue_role",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

Если вы выбираете учётные данные на основе IAM user

- Если вы используете Hive metastore в вашем cluster Delta Lake, выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_access_key>",
    "aws.s3.region" = "us-west-2"
);
```

- Если вы используете AWS Glue в вашем cluster Delta Lake Amazon EMR, выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_glue
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "false",
    "aws.glue.access_key" = "<iam_user_access_key>",
    "aws.glue.secret_key" = "<iam_user_secret_key>",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_secret_key>",
    "aws.s3.region" = "us-west-2"
);
```

S3-совместимая система хранения

Используем MinIO в качестве примера. Выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.enable_ssl" = "true",
    "aws.s3.enable_path_style_access" = "true",
    "aws.s3.endpoint" = "<s3_endpoint>",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_secret_key>"
);
```

Microsoft Azure Storage

Azure Blob Storage

- Если вы выбираете метод аутентификации Shared Key, выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.blob.storage_account" = "<blob_storage_account_name>",
    "azure.blob.shared_key" = "<blob_storage_account_shared_key>"
);
```

- Если вы выбираете метод аутентификации SAS Token, выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.blob.storage_account" = "<blob_storage_account_name>",
    "azure.blob.container" = "<blob_container_name>",
    "azure.blob.sas_token" = "<blob_storage_account_SAS_token>"
);
```

Azure Data Lake Storage Gen1

- Если вы выбираете метод аутентификации Managed Service Identity, выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls1.use_managed_service_identity" = "true"
);
```

- Если вы выбираете метод аутентификации Service Principal, выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls1.oauth2_client_id" = "<application_client_id>",
    "azure.adls1.oauth2_credential" = "
<application_client_credential>",
    "azure.adls1.oauth2_endpoint" = "
<OAuth_2.0_authorization_endpoint_v2>"
);
```

Azure Data Lake Storage Gen2

- Если вы выбираете метод аутентификации Managed Identity, выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls2.oauth2_use_managed_identity" = "true",
    "azure.adls2.oauth2_tenant_id" = "<service_principal_tenant_id>",
    "azure.adls2.oauth2_client_id" = "<service_client_id>"
);
```

- Если вы выбираете метод аутентификации Shared Key, выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls2.storage_account" = "<storage_account_name>",
    "azure.adls2.shared_key" = "<shared_key>"
);
```

- Если вы выбираете метод аутентификации Service Principal, выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls2.oauth2_client_id" = "<service_client_id>",
    "azure.adls2.oauth2_client_secret" = "
<service_principal_client_secret>",
    "azure.adls2.oauth2_client_endpoint" = "
<service_principal_client_endpoint>"
);
```

Google GCS

- Если вы выбираете метод аутентификации на основе VM, выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "gcp.gcs.use_compute_engine_service_account" = "true"
);
```

- Если вы выбираете метод аутентификации на основе service account, выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "gcp.gcs.service_account_email" = "<google_service_account_email>",
    "gcp.gcs.service_account_private_key_id" = "
<google_service_private_key_id>",
    "gcp.gcs.service_account_private_key" = "
<google_service_private_key>"
);
```

- Если вы выбираете метод аутентификации на основе impersonation:
  - Если вы делаете так, чтобы экземпляр VM выдавал себя за учётную запись службы, выполните команду, подобную приведённой ниже:

    ```
    CREATE EXTERNAL CATALOG deltalake_catalog_hms
    PROPERTIES
    (
        "type" = "deltalake",
        "hive.metastore.type" = "hive",
        "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
        "gcp.gcs.use_compute_engine_service_account" = "true",
        "gcp.gcs.impersonation_service_account" = "
    <assumed_google_service_account_email>"
    );
    ```

  - Если вы делаете так, чтобы учётная запись службы выдавала себя за другую учётную запись службы, выполните команду, подобную приведённой ниже:

```
CREATE EXTERNAL CATALOG deltalake_catalog_hms
PROPERTIES
(
    "type" = "deltalake",
    "hive.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "gcp.gcs.service_account_email" = "
<google_service_account_email>",
    "gcp.gcs.service_account_private_key_id" = "
<meta_google_service_account_email>",
    "gcp.gcs.service_account_private_key" = "
<meta_google_service_account_email>",
    "gcp.gcs.impersonation_service_account" = "
<data_google_service_account_email>"
);
```

Просмотр Delta Lake catalogs

Вы можете использовать SHOW CATALOGS для запроса всех catalogs в текущем cluster Selena:

```
SHOW CATALOGS;
```

Вы также можете использовать SHOW CREATE CATALOG для запроса оператора создания внешнего catalog. Следующий пример запрашивает оператор создания Delta Lake catalog с именем `deltalake_catalog_glue`:

```
SHOW CREATE CATALOG deltalake_catalog_glue;
```

Переключение на Delta Lake Catalog и базу данных в нём

Вы можете использовать один из следующих методов для переключения на Delta Lake catalog и базу данных в нём:

- Используйте SET CATALOG для указания Delta Lake catalog в текущей сессии, а затем используйте USE для указания активной базы данных:

```
-- Переключиться на указанный catalog в текущей сессии:
SET CATALOG <catalog_name>
-- Указать активную базу данных в текущей сессии:
USE <db_name>
```

- Напрямую используйте USE для переключения на Delta Lake catalog и базу данных в нём:

```
USE <catalog_name>.<db_name>
```

Удаление Delta Lake catalog

Вы можете использовать DROP CATALOG для удаления внешнего catalog.

Следующий пример удаляет Delta Lake catalog с именем `deltalake_catalog_glue`:

```
DROP Catalog deltalake_catalog_glue;
```

Просмотр схемы таблицы Delta Lake

Вы можете использовать один из следующих синтаксисов для просмотра схемы таблицы Delta Lake:

- Просмотр схемы

```
DESC[RIBE] <catalog_name>.<database_name>.<table_name>
```

- Просмотр схемы и расположения из оператора CREATE

```
SHOW CREATE TABLE <catalog_name>.<database_name>.<table_name>
```

Запрос таблицы Delta Lake

1. Используйте SHOW DATABASES для просмотра баз данных в вашем cluster Delta Lake:

```
SHOW DATABASES FROM <catalog_name>
```

2. Переключитесь на Delta Lake Catalog и базу данных в нём.

3. Используйте SELECT для запроса целевой таблицы в указанной базе данных:

```
SELECT count(*) FROM <table_name> LIMIT 10
```

Загрузка данных из Delta Lake

Предположим, у вас есть OLAP-таблица с именем `olap_tbl`, вы можете преобразовать и загрузить данные следующим образом:

```
INSERT INTO default_catalog.olap_db.olap_tbl SELECT * FROM deltalake_table
```

Настройка кеша метаданных и стратегии обновления

Начиная с версии v1.5.2, Delta Lake Catalog поддерживает Локальное кеширование и получение метаданных.

Вы можете настроить обновление кеша метаданных Delta Lake с помощью следующих параметров FE:

Элемент конфигурации	По умолчанию	Описание
enable_background_refresh_connector_metadata	`true`	Включать ли периодическое обновление кеша метаданных Delta Lake. После включения Selena опрашивает metastore (Hive Metastore или AWS Glue) вашего cluster Delta Lake и обновляет кешированные метаданные часто используемых Delta Lake catalogs для восприятия изменений данных. `true` указывает на включение обновления кеша метаданных Delta Lake, а `false` указывает на отключение.
background_refresh_metadata_interval_millis	`600000`	Интервал между двумя последовательными обновлениями кеша метаданных Delta Lake. Единица измерения: миллисекунда.
background_refresh_metadata_time_secs_sin	`86400`	Время истечения задачи обновления кеша метаданных Delta Lake. Для Delta Lake catalog, к которому был осуществлён доступ, если к нему не было обращений более указанного времени, Selena прекращает обновление его кешированных метаданных.

Элемент конфигурации	По умолчанию	Описание
ce_last_access_secs		Для Delta Lake catalog, к которому не было обращений, Selena не будет обновлять его кешированные метаданные. Единица измерения: секунда.

Приложение: Локальное кеширование и получение метаданных

Поскольку повторная распаковка и разбор файлов метаданных могут вызывать ненужные задержки, Selena использует новую стратегию кеширования метаданных — кеширование десериализованных объектов в памяти. Сохраняя эти десериализованные файлы в памяти, система может обойти этапы распаковки и разбора для последующих запросов. Этот механизм кеширования позволяет напрямую получать доступ к требуемым метаданным, значительно сокращая время получения. В результате система становится более отзывчивой и лучше приспособленной для удовлетворения высоких требований к запросам и потребностей в перезаписи материализованных представлений.

Вы можете настроить это поведение через свойство Catalog MetadataUpdateParams и соответствующие элементы конфигурации.

Поддержка функций

В настоящее время Delta Lake catalogs поддерживают следующие возможности таблиц:

- V2 Checkpoint (начиная с v1.5.2)
- Timestamp without Timezone (начиная с v1.5.2)
- Column mapping (начиная с v1.5.2)
- Deletion Vector (начиная с v1.5.2)

5.7.7 JDBC catalog

> *Beta* **BETA FEATURE**
>
> Рекомендации по использованию бета-функций

Selena поддерживает JDBC catalogs начиная с версии v1.5.2.

JDBC catalog — это тип внешнего catalog, который позволяет запрашивать данные из источников данных, доступных через JDBC, без загрузки.

Кроме того, вы можете напрямую преобразовывать и загружать данные из источников данных JDBC с помощью INSERT INTO на основе JDBC catalogs.

JDBC catalogs поддерживают MySQL и PostgreSQL начиная с v1.5.2.

Предварительные требования

- FE и BE или CN в вашем Selena cluster могут загружать JDBC driver по URL загрузки, указанному параметром `driver_url`.
- `JAVA_HOME` в файле **$BE_HOME/bin/start_be.sh** на каждом BE или CN node правильно настроен как путь в окружении JDK, а не как путь в окружении JRE. Например, вы можете настроить `export JAVA_HOME = <JDK_absolute_path>`. Вы должны добавить эту конфигурацию в начало скрипта и перезапустить BE или CN, чтобы конфигурация вступила в силу.

Создание JDBC catalog

Синтаксис

```
CREATE EXTERNAL CATALOG <catalog_name>
[COMMENT <comment>]
PROPERTIES ("key"="value", ...)
```

Параметры

`catalog_name`

Название JDBC catalog. Соглашения об именовании следующие:

- Название может содержать буквы, цифры (0-9) и подчеркивания (_). Оно должно начинаться с буквы.
- Название чувствительно к регистру и не может превышать 1023 символа.

`comment`

Описание JDBC catalog. Этот параметр необязателен.

`PROPERTIES`

Свойства JDBC Catalog. `PROPERTIES` должен включать следующие параметры:

Параметр	Описание
type	Тип ресурса. Установите значение `jdbc`.
user	Имя пользователя, которое используется для подключения к целевой базе данных.
password	Пароль, который используется для подключения к целевой базе данных.
jdbc_uri	URI, который JDBC driver использует для подключения к целевой базе данных. Для MySQL URI имеет формат `"jdbc:mysql://ip:port"`. Для PostgreSQL URI имеет формат `"jdbc:postgresql://ip:port/db_name"`. Для получения дополнительной информации: PostgreSQL.
driver_url	URL загрузки JAR-пакета JDBC driver. Поддерживаются HTTP URL или file URL, например, `https://repo1.maven.org/maven2/org/postgresql/postgresql/42.3.3/postgresql-42.3.3.jar` и `file:///home/disk1/postgresql-42.3.3.jar`. **ПРИМЕЧАНИЕ** Вы также можете поместить JDBC driver по одному и тому же пути на FE и BE или CN nodes и установить `driver_url` на этот путь, который должен быть в формате `file:///<path>/to/the/driver`.
driver_class	Имя класса JDBC driver. Имена классов JDBC driver распространенных движков баз данных следующие: • MySQL: `com.mysql.jdbc.Driver` (MySQL v5.x и ранее) и `com.mysql.cj.jdbc.Driver` (MySQL v6.x и позже) • PostgreSQL: `org.postgresql.Driver`

> **ПРИМЕЧАНИЕ**
>
> FE загружают JAR-пакет JDBC driver во время создания JDBC catalog, а BE или CN загружают JAR-пакет JDBC driver во время первого запроса. Время, затрачиваемое на загрузку, зависит от условий сети.

Примеры

Следующий пример создает пять различных JDBC catalogs.

```
-- PostgresSQL
CREATE EXTERNAL CATALOG jdbc0
PROPERTIES
(
    "type"="jdbc",
    "user"="postgres",
    "password"="changeme",
    "jdbc_uri"="jdbc:postgresql://127.0.0.1:5432/jdbc_test",
    "driver_url"="https://repo1.maven.org/maven2/org/postgresql/postgresql/42
42.3.3.jar",
    "driver_class"="org.postgresql.Driver"
);
-- MySQL
CREATE EXTERNAL CATALOG jdbc1
PROPERTIES
(
    "type"="jdbc",
    "user"="root",
    "password"="changeme",
    "jdbc_uri"="jdbc:mysql://127.0.0.1:3306",
    "driver_url"="https://repo1.maven.org/maven2/mysql/mysql-connector-java/8
connector-java-8.0.28.jar",
    "driver_class"="com.mysql.cj.jdbc.Driver"
);
-- Oracle
CREATE EXTERNAL CATALOG jdbc2
PROPERTIES
(
    "type"="jdbc",
    "user"="root",
    "password"="changeme",
    "jdbc_uri"="jdbc:oracle:thin:@127.0.0.1:1521:ORCL",

"driver_url"="https://repo1.maven.org/maven2/com/oracle/database/jdbc/ojdbc10
19.18.0.0.jar",
    "driver_class"="oracle.jdbc.driver.OracleDriver"
);
-- SQL Server
CREATE EXTERNAL CATALOG jdbc3
PROPERTIES
(
    "type"="jdbc",
    "user"="root",
    "password"="changeme",
    "jdbc_uri"="jdbc:sqlserver://127.0.0.1:1433;databaseName=MyDatabase;",
```

```
    "driver_url"="https://repo1.maven.org/maven2/com/microsoft/sqlserver/mssq
jdbc/12.4.2.jre11/mssql-jdbc-12.4.2.jre11.jar",
    "driver_class"="com.microsoft.sqlserver.jdbc.SQLServerDriver"
);
-- ClickHouse
CREATE EXTERNAL CATALOG jdbc4
PROPERTIES
(
    "type"="jdbc",
    "user"="default",
    "jdbc_uri"="jdbc:clickhouse://127.0.0.1:8443",
    "driver_url"="file:///path/to/clickhouse-jdbc-0.4.6.jar",
    "driver_class"="com.clickhouse.jdbc.ClickHouseDriver"
);
```

Просмотр JDBC catalogs

Вы можете использовать SHOW CATALOGS для запроса всех catalogs в текущем Selena cluster:

```
SHOW CATALOGS;
```

Вы также можете использовать SHOW CREATE CATALOG для запроса оператора создания внешнего catalog. Следующий пример запрашивает оператор создания JDBC catalog с именем `jdbc0`:

```
SHOW CREATE CATALOG jdbc0;
```

Удаление JDBC catalog

Вы можете использовать DROP CATALOG для удаления JDBC catalog.

Следующий пример удаляет JDBC catalog с именем `jdbc0`:

```
DROP Catalog jdbc0;
```

Запрос таблицы в JDBC catalog

1. Используйте SHOW DATABASES для просмотра баз данных в вашем JDBC-совместимом cluster:

```
SHOW DATABASES FROM <catalog_name>;
```

2. Используйте SET CATALOG для переключения на целевой catalog в текущей сессии:

```
SET CATALOG <catalog_name>;
```

   Затем используйте USE для указания активной базы данных в текущей сессии:

```
USE <db_name>;
```

   Или вы можете использовать USE для непосредственного указания активной базы данных в целевом catalog:

```
USE <catalog_name>.<db_name>;
```

3. Используйте SELECT для запроса целевой таблицы в указанной базе данных:

```
SELECT * FROM <table_name>;
```

FAQ

Что делать, если появляется ошибка «Malformed database URL, failed to parse the main URL sections»?

Если вы столкнулись с такой ошибкой, URI, который вы передали в `jdbc_uri`, недействителен. Проверьте URI, который вы передаете, и убедитесь, что он действителен. Для получения дополнительной информации см. описания параметров в разделе «PROPERTIES» этой темы.

5.7.8 Elasticsearch catalog

Selena поддерживает Elasticsearch catalogs начиная с версии v1.5.2.

Selena и Elasticsearch — это обе популярные аналитические системы с различными сильными сторонами. Selena превосходна в крупномасштабных распределённых вычислениях и поддерживает запрос данных из Elasticsearch через внешние таблицы. Elasticsearch известен своими возможностями полнотекстового поиска. Комбинация Selena и Elasticsearch обеспечивает более комплексное OLAP-решение. С помощью Elasticsearch catalogs вы можете напрямую анализировать все индексированные данные в вашем cluster Elasticsearch, используя SQL-операторы в Selena без необходимости миграции данных.

В отличие от catalogs для других источников данных, Elasticsearch catalog имеет только одну базу данных с именем `default_db` при создании. Каждый индекс Elasticsearch автоматически сопоставляется с таблицей данных и монтируется в базу данных `default_db`.

Создание Elasticsearch catalog

Синтаксис

```
CREATE EXTERNAL CATALOG <catalog_name>
[COMMENT <comment>]
PROPERTIES ("key"="value", ...)
```

Параметры

`catalog_name`

Имя Elasticsearch catalog. Соглашения об именовании следующие:

- Имя может содержать буквы, цифры (0-9) и символы подчёркивания (_). Оно должно начинаться с буквы.
- Имя чувствительно к регистру и не может превышать 1023 символа в длину.

`comment`

Описание Elasticsearch catalog. Этот параметр является опциональным.

PROPERTIES

Свойства Elasticsearch catalog. В следующей таблице описаны свойства, поддерживаемые для Elasticsearch catalogs.

Параметр	Обязательный	Значение по умолчанию	Описание
hosts	Да	Нет	Адрес подключения к cluster Elasticsearch. Вы можете указать один или несколько адресов. Selena может разобрать версию Elasticsearch и распределение shard индекса по этому адресу. Selena взаимодействует с вашим cluster Elasticsearch на основе адреса, возвращённого операцией API `GET /_nodes/http`. Поэтому значение параметра `hosts` должно совпадать с адресом, возвращённым операцией API `GET /_nodes/http`. В противном случае BE или CN могут не иметь возможности взаимодействовать с вашим cluster Elasticsearch.
type	Да	Нет	Тип источника данных. Установите этот параметр в `es` при создании Elasticsearch catalog.
user	Нет	Пусто	Имя пользователя, которое используется для входа в cluster Elasticsearch с включённой базовой HTTP-аутентификацией. Убедитесь, что у вас есть разрешения на доступ к путям, таким как `/cluster/state/ nodes/http`, и разрешения на чтение индекса.
password	Нет	Пусто	Пароль, который используется для входа в cluster Elasticsearch.
es.type	Нет	_doc	Тип индекса. Если вы хотите запросить данные в Elasticsearch версии 8 и более поздних, вам не нужно настраивать этот параметр, потому что типы сопоставления были удалены в Elasticsearch версии 8 и более поздних.
es.nodes.wan.only	Нет	FALSE	Указывает, использует ли Selena только адреса, указанные в `hosts`, для доступа к cluster Elasticsearch и получения данных. • `true`: Selena использует только адреса, указанные в `hosts`, для доступа к cluster Elasticsearch и получения данных и не отслеживает узлы данных, на которых находятся shard индекса Elasticsearch. Если Selena не может получить доступ к адресам узлов данных внутри cluster Elasticsearch, вам нужно установить этот параметр в `true`. • `false`: Selena использует адреса, указанные в `hosts`, для отслеживания узлов данных, на которых находятся shard индексов cluster Elasticsearch. После того как Selena сгенерирует план выполнения запроса, BE или CN напрямую обращаются к узлам данных внутри cluster Elasticsearch для получения данных из shard индексов. Если Selena может получить доступ к адресам узлов данных внутри cluster Elasticsearch, мы рекомендуем сохранить значение по умолчанию `false`.

Параметр	Обязательный	Значение по умолчанию	Описание
es.net.ssl	Нет	FALSE	Указывает, можно ли использовать протокол HTTPS для доступа к cluster Elasticsearch. Только Selena v1.5.2 и более поздние версии поддерживают настройку этого параметра. • `true`: Оба протокола HTTPS и HTTP могут использоваться для доступа к вашему cluster Elasticsearch. • `false`: Только протокол HTTP может использоваться для доступа к вашему cluster Elasticsearch.
enable_docvalue_scan	Нет	TRUE	Указывает, получать ли значения целевых полей из столбцового хранилища Elasticsearch. В большинстве случаев чтение данных из столбцового хранилища превосходит чтение данных из строкового хранилища.
enable_keyword_sniff	Нет	TRUE	Указывает, отслеживать ли поля типа TEXT в Elasticsearch на основе полей типа KEYWORD. Если этот параметр установлен в `false`, Selena выполняет сопоставление после токенизации.

Примеры

Следующий пример создаёт Elasticsearch catalog с именем `es_test`:

```
CREATE EXTERNAL CATALOG es_test
COMMENT 'test123'
PROPERTIES
(
    "type" = "es",
    "es.type" = "_doc",
    "hosts" = "https://xxx:9200",
    "es.net.ssl" = "true",
    "user" = "admin",
    "password" = "xxx",
    "es.nodes.wan.only" = "true"
);
```

Проталкивание предикатов

Selena поддерживает проталкивание предикатов, указанных в запросах к таблицам Elasticsearch, в Elasticsearch для выполнения. Это минимизирует расстояние между движком запросов и источником хранения и улучшает производительность запросов. В следующей таблице перечислены операторы, которые могут быть протолкнуты в Elasticsearch.

SQL-синтаксис	Синтаксис Elasticsearch
`=`	term query
`in`	terms query
`>=, <=, >, <`	range
`and`	bool.filter
`or`	bool.should
`not`	bool.must_not
`not in`	bool.must_not + terms
`esquery`	ES Query DSL

Примеры запросов

Функция `esquery()` может использоваться для проталкивания запросов Elasticsearch, таких как запросы match и geoshape, которые не могут быть выражены в SQL, в Elasticsearch для фильтрации и обработки. В функции `esquery()` первый параметр, который указывает имя столбца, используется для связи с индексом, а второй параметр — это JSON-представление запроса Elasticsearch на основе Elasticsearch Query DSL, заключённое в фигурные скобки (`{}`). JSON-представление может и должно иметь только один корневой ключ, такой как `match`, `geo_shape` или `bool`.

- Запрос Match

```
SELECT * FROM es_table WHERE esquery(k4, '{
   "match": {
      "k4": "Selena on elasticsearch"
   }
}');
```

- Запрос Geoshape

```
SELECT * FROM es_table WHERE esquery(k4, '{
"geo_shape": {
   "location": {
      "shape": {
         "type": "envelope",
         "coordinates": [
            [
               13,
               53
            ],
            [
               14,
               52
            ]
         ]
      },
      "relation": "within"
   }
}
}');
```

- Запрос Boolean

```
SELECT * FROM es_table WHERE esquery(k4, ' {
   "bool": {
      "must": [
         {
            "terms": {
               "k1": [
                  11,
                  12
               ]
            }
         },
         {
            "terms": {
               "k2": [
                  100
               ]
            }
         }
      ]
   }
}');
```

Примечания по использованию

- Начиная с версии v5.x, Elasticsearch использует другой метод сканирования данных. Selena поддерживает только запрос данных из Elasticsearch v5.x и более поздних версий.
- Selena поддерживает только запрос данных из clusters Elasticsearch, для которых включена базовая HTTP-аутентификация.
- Некоторые запросы, такие как запросы с использованием `count()`, выполняются намного медленнее в Selena, чем в Elasticsearch, потому что Elasticsearch может напрямую читать метаданные, связанные с указанным количеством документов, соответствующих условиям запроса, без необходимости фильтровать запрошенные данные.

5.7.9 Paimon catalog

> *Beta* **BETA FEATURE**
>
> Рекомендации по использованию бета-функций

Selena поддерживает Paimon catalog начиная с версии v1.5.2.

Paimon catalog — это разновидность внешнего catalog, которая позволяет выполнять запросы к данным из Apache Paimon без необходимости их загрузки.

Кроме того, вы можете напрямую трансформировать и загружать данные из Paimon с помощью INSERT INTO на основе Paimon catalog.

Для успешного выполнения SQL-запросов к вашему Paimon cluster, ваш Selena cluster должен иметь доступ к системе хранения и metastore вашего Paimon cluster. Selena поддерживает следующие системы хранения и metastore:

- Распределённая файловая система (HDFS) или объектное хранилище, такое как AWS S3, Microsoft Azure Storage, Google GCS, или другие S3-совместимые системы хранения (например, MinIO)
- Metastore, такие как файловая система или Hive metastore

Примечания по использованию

Вы можете использовать Paimon catalog только для запроса данных. Нельзя использовать Paimon catalog для удаления или вставки данных в ваш Paimon cluster.

Соответствие типов данных Paimon и Selena

Тип Paimon	Тип Selena
`BINARY`	`VARBINARY`
`VARBINARY`	`VARBINARY`
`CHAR`	`CHAR(length)`
`VARCHAR`	`VARCHAR`
`BOOLEAN`	`BOOLEAN`
`DECIMAL`	`DECIMAL(precision, scale)`
`TINYINT`	`TINYINT`
`SMALLINT`	`SMALLINT`
`INT`	`INT`
`BIGINT`	`BIGINT`
`FLOAT`	`FLOAT`
`DOUBLE`	`DOUBLE`
`DATE`	`DATE`

Тип Paimon	Тип Selena
`TIME`	`TIME`
`TIMESTAMP`	`DATETIME`
`LocalZonedTimestamp`	`DATETIME`
`ARRAY`	`ARRAY<element_type>`
`MAP`	`MAP<key_type, value_type>`
`ROW/STRUCT`	`STRUCT<field1:type1, ...>`

Подготовка к интеграции

Перед созданием Paimon catalog убедитесь, что ваш Selena cluster может интегрироваться с системой хранения и metastore вашего Paimon cluster.

AWS IAM

Если ваш Paimon cluster использует AWS S3 в качестве хранилища, выберите подходящий метод аутентификации и выполните необходимую подготовку, чтобы ваш Selena cluster мог получить доступ к соответствующим облачным ресурсам AWS.

Рекомендуются следующие методы аутентификации:

- Instance profile (рекомендуется)
- Assumed role
- IAM user

Из вышеупомянутых трёх методов аутентификации instance profile используется наиболее широко.

Подробнее см. в разделе Подготовка к аутентификации в AWS IAM.

HDFS

Если вы выбрали HDFS в качестве хранилища, настройте ваш Selena cluster следующим образом:

- (Опционально) Установите имя пользователя, которое используется для доступа к вашему HDFS cluster и Hive metastore. По умолчанию Selena использует имя пользователя процессов FE и BE или CN для доступа к вашему HDFS cluster и Hive metastore. Вы также можете задать имя пользователя, добавив `export HADOOP_USER_NAME="<user_name>"` в начало файла **fe/conf/hadoop_env.sh** каждого

FE и в начало файла **be/conf/hadoop_env.sh** каждого BE или файла **cn/conf/hadoop_env.sh** каждого CN. После установки имени пользователя в этих файлах перезапустите каждый FE и каждый BE или CN, чтобы настройки параметров вступили в силу. Для каждого Selena cluster можно задать только одно имя пользователя.

- При запросе данных Paimon, FE и BE или CN вашего Selena cluster используют HDFS-клиент для доступа к вашему HDFS cluster. В большинстве случаев вам не нужно настраивать ваш Selena cluster для этого, и Selena запускает HDFS-клиент с настройками по умолчанию. Настройка Selena cluster необходима только в следующих ситуациях:
  - Для вашего HDFS cluster включена высокая доступность (HA): Добавьте файл **hdfs-site.xml** вашего HDFS cluster в путь **$FE_HOME/conf** каждого FE и в путь **$BE_HOME/conf** каждого BE или путь **$CN_HOME/conf** каждого CN.
  - Для вашего HDFS cluster включена View File System (ViewFs): Добавьте файл **core-site.xml** вашего HDFS cluster в путь **$FE_HOME/conf** каждого FE и в путь **$BE_HOME/conf** каждого BE или путь **$CN_HOME/conf** каждого CN.

> **ПРИМЕЧАНИЕ**
>
> Если при отправке запроса возвращается ошибка о неизвестном хосте, необходимо добавить соответствие между именами хостов и IP-адресами узлов вашего HDFS cluster в путь **/etc/hosts**.

Аутентификация Kerberos

Если для вашего HDFS cluster или Hive metastore включена аутентификация Kerberos, настройте ваш Selena cluster следующим образом:

- Выполните команду `kinit -kt keytab_path principal` на каждом FE и каждом BE или CN, чтобы получить Ticket Granting Ticket (TGT) от Key Distribution Center (KDC). Для выполнения этой команды у вас должны быть разрешения на доступ к вашему HDFS cluster и Hive metastore. Обратите внимание, что доступ к KDC с помощью этой команды ограничен по времени. Поэтому необходимо использовать cron для периодического выполнения этой команды.
- Добавьте `JAVA_OPTS="-Djava.security.krb5.conf=/etc/krb5.conf"` в файл **$FE_HOME/conf/fe.conf** каждого FE и в файл **$BE_HOME/conf/be.conf** каждого BE или файл **$CN_HOME/conf/cn.conf** каждого CN. В этом примере `/etc/krb5.conf` —

это путь сохранения файла **krb5.conf**. Вы можете изменить путь в соответствии с вашими потребностями.

Создание Paimon catalog

Синтаксис

```
CREATE EXTERNAL CATALOG <catalog_name>
[COMMENT <comment>]
PROPERTIES
(
    "type" = "paimon",
    CatalogParams,
    StorageCredentialParams,
)
```

Параметры

catalog_name

Имя Paimon catalog. Соглашения об именовании следующие:

- Имя может содержать буквы, цифры (0-9) и символы подчёркивания (_). Оно должно начинаться с буквы.
- Имя чувствительно к регистру и не может превышать 1023 символа в длину.

comment

Описание Paimon catalog. Этот параметр является опциональным.

type

Тип вашего источника данных. Установите значение `paimon`.

CatalogParams

Набор параметров о том, как Selena получает доступ к метаданным вашего Paimon cluster.

В следующей таблице описан параметр, который необходимо настроить в `CatalogParams`.

Параметр	Обязательный	Описание
paimon.catalog.type	Да	Тип metastore, который вы используете для вашего Paimon cluster. Установите этот параметр на `filesystem` или `hive`.
paimon.catalog.warehouse	Да	Путь к хранилищу warehouse ваших данных Paimon.
hive.metastore.uris	Нет	URI вашего Hive metastore. Формат: `thrift://<metastore_IP_address>:<metastore_port>`. Если для вашего Hive metastore включена высокая доступность (HA), вы можете указать несколько URI metastore и разделить их запятыми (`,`), например, `"thrift://<metastore_IP_address_1>:<metastore_port_1>,thrift://<metastore_IP_address_2>:<metastore_port_2>,thrift://<metastore_IP_address_3>:<metastore_port_3>"`.

> **ПРИМЕЧАНИЕ**
>
> Если вы используете Hive metastore, вы должны добавить соответствие между именами хостов и IP-адресами узлов вашего Hive metastore в путь `/etc/hosts` перед запросом данных Paimon. В противном случае Selena может не получить доступ к вашему Hive metastore при запуске запроса.

StorageCredentialParams

Набор параметров о том, как Selena интегрируется с вашей системой хранения. Этот набор параметров является опциональным.

Если вы используете HDFS в качестве хранилища, вам не нужно настраивать `StorageCredentialParams`.

Если вы используете AWS S3, другую S3-совместимую систему хранения, Microsoft Azure Storage или Google GCS в качестве хранилища, вы должны настроить `StorageCredentialParams`.

AWS S3

Если вы выбрали AWS S3 в качестве хранилища для вашего Paimon cluster, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации на основе instance profile, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.endpoint" = "<aws_s3_endpoint>"
```

- Чтобы выбрать метод аутентификации на основе assumed role, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.iam_role_arn" = "<iam_role_arn>",
"aws.s3.endpoint" = "<aws_s3_endpoint>"
```

- Чтобы выбрать метод аутентификации на основе IAM user, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "false",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>",
"aws.s3.endpoint" = "<aws_s3_endpoint>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
aws.s3.use_instance_profile	Да	Указывает, следует ли включить метод аутентификации на основе instance profile и метод аутентификации на основе assumed role. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`.
aws.s3.iam_role_arn	Нет	ARN IAM role, которая имеет привилегии на ваш AWS S3 bucket. Если вы используете метод аутентификации на основе assumed role для доступа к AWS S3, вы должны указать этот параметр.
aws.s3.endpoint	Да	Endpoint, который используется для подключения к вашему AWS S3 bucket. Например, `https://s3.us-west-2.amazonaws.com`.
aws.s3.access_key	Нет	Access key вашего IAM user. Если вы используете метод аутентификации на основе IAM user для доступа к AWS S3, вы должны указать этот параметр.

Параметр	Обязательный	Описание
aws.s3.secret_key	Нет	Secret key вашего IAM user. Если вы используете метод аутентификации на основе IAM user для доступа к AWS S3, вы должны указать этот параметр.

Информацию о том, как выбрать метод аутентификации для доступа к AWS S3 и как настроить политику управления доступом в AWS IAM Console, см. в разделе Параметры аутентификации для доступа к AWS S3.

S3-совместимая система хранения

Если вы выбрали S3-совместимую систему хранения, такую как MinIO, в качестве хранилища для вашего Paimon cluster, настройте `StorageCredentialParams` следующим образом для обеспечения успешной интеграции:

```
"aws.s3.enable_ssl" = "false",
"aws.s3.enable_path_style_access" = "true",
"aws.s3.endpoint" = "<s3_endpoint>",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
aws.s3.enable_ssl	Да	Указывает, следует ли включить SSL-соединение. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`.
aws.s3.enable_path_style_access	Да	Указывает, следует ли включить доступ в стиле пути. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`. Для MinIO вы должны установить значение `true`. URL в стиле пути используют следующий формат: `https://s3.<region_code>.amazonaws.com/<bucket_name>/<key_name>`. Например, если вы создали bucket с именем `DOC-EXAMPLE-BUCKET1` в регионе US West (Oregon) и хотите получить доступ к объекту `alice.jpg` в этом bucket, вы можете использовать следующий URL в стиле пути: `https://s3.us-west-2.amazonaws.com/DOC-EXAMPLE-BUCKET1/alice.jpg`.

Параметр	Обязательный	Описание
aws.s3.endpoint	Да	Endpoint, который используется для подключения к вашей S3-совместимой системе хранения вместо AWS S3.
aws.s3.access_key	Да	Access key вашего IAM user.
aws.s3.secret_key	Да	Secret key вашего IAM user.

Microsoft Azure Storage

Azure Blob Storage

Если вы выбрали Blob Storage в качестве хранилища для вашего Paimon cluster, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Shared Key, настройте `StorageCredentialParams` следующим образом:

  ```
  "azure.blob.storage_account" = "<storage_account_name>",
  "azure.blob.shared_key" = "<storage_account_shared_key>"
  ```

  В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.blob.storage_account	Да	Имя пользователя вашей учётной записи Blob Storage.
azure.blob.shared_key	Да	Shared key вашей учётной записи Blob Storage.

- Чтобы выбрать метод аутентификации SAS Token, настройте `StorageCredentialParams` следующим образом:

  ```
  "azure.blob.storage_account" = "<storage_account_name>",
  "azure.blob.container" = "<container_name>",
  "azure.blob.sas_token" = "<storage_account_SAS_token>"
  ```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.blob.storage_account	Да	Имя пользователя вашей учётной записи Blob Storage.
azure.blob.container	Да	Имя blob-контейнера, в котором хранятся ваши данные.
azure.blob.sas_token	Да	SAS token, который используется для доступа к вашей учётной записи Blob Storage.

Azure Data Lake Storage Gen2

Если вы выбрали Data Lake Storage Gen2 в качестве хранилища для вашего Paimon cluster, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Managed Identity, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.oauth2_use_managed_identity" = "true",
"azure.adls2.oauth2_tenant_id" = "<service_principal_tenant_id>",
"azure.adls2.oauth2_client_id" = "<service_client_id>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls2.oauth2_use_managed_identity	Да	Указывает, следует ли включить метод аутентификации Managed Identity. Установите значение `true`.
azure.adls2.oauth2_tenant_id	Да	ID tenant, к данным которого вы хотите получить доступ.
azure.adls2.oauth2_client_id	Да	Client (application) ID управляемого удостоверения.

- Чтобы выбрать метод аутентификации Shared Key, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.storage_account" = "<storage_account_name>",
"azure.adls2.shared_key" = "<storage_account_shared_key>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls2.storage_account	Да	Имя пользователя вашей учётной записи хранения Data Lake Storage Gen2.
azure.adls2.shared_key	Да	Shared key вашей учётной записи хранения Data Lake Storage Gen2.

- Чтобы выбрать метод аутентификации Service Principal, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.oauth2_client_id" = "<service_client_id>",
"azure.adls2.oauth2_client_secret" = "
<service_principal_client_secret>",
"azure.adls2.oauth2_client_endpoint" = "
<service_principal_client_endpoint>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls2.oauth2_client_id	Да	Client (application) ID service principal.
azure.adls2.oauth2_client_secret	Да	Значение нового секрета клиента (приложения), который был создан.
azure.adls2.oauth2_client_endpoint	Да	OAuth 2.0 token endpoint (v1) service principal или приложения.

Azure Data Lake Storage Gen1

Если вы выбрали Data Lake Storage Gen1 в качестве хранилища для вашего Paimon cluster, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Managed Service Identity, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls1.use_managed_service_identity" = "true"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls1.use_managed_service_identity	Да	Указывает, следует ли включить метод аутентификации Managed Service Identity. Установите значение `true`.

- Чтобы выбрать метод аутентификации Service Principal, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls1.oauth2_client_id" = "<application_client_id>",
"azure.adls1.oauth2_credential" = "<application_client_credential>",
"azure.adls1.oauth2_endpoint" = "<OAuth_2.0_authorization_endpoint_v2>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls1.oauth2_client_id	Да	Client (application) ID service principal.
azure.adls1.oauth2_credential	Да	Значение нового секрета клиента (приложения), который был создан.
azure.adls1.oauth2_endpoint	Да	OAuth 2.0 token endpoint (v1) service principal или приложения.

Google GCS

Если вы выбрали Google GCS в качестве хранилища для вашего Paimon cluster, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации на основе VM, настройте `StorageCredentialParams` следующим образом:

```
"gcp.gcs.use_compute_engine_service_account" = "true"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.use_compute_engine_service_account	FALSE	TRUE	Указывает, следует ли напрямую использовать сервисную учётную запись, привязанную к вашему Compute Engine.

- Чтобы выбрать метод аутентификации на основе сервисной учётной записи, настройте `StorageCredentialParams` следующим образом:

```
"gcp.gcs.service_account_email" = "<google_service_account_email>",
"gcp.gcs.service_account_private_key_id" = "
<google_service_private_key_id>",
"gcp.gcs.service_account_private_key" = "<google_service_private_key>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.service_account_email	""	"user@hello.iam.gserviceaccount.com"	Адрес электронной почты в JSON-файле, сгенерированном при создании сервисной учётной записи.
gcp.gcs.service_account_private_key_id	""	"61d257bd8479547cb3e04f0b9b6b9ca07af3b7ea"	ID приватного ключа в JSON-файле, сгенерированном при создании сервисной учётной записи.
gcp.gcs.service_account_private_key	""	"-----BEGIN PRIVATE KEY-----xxxx-----END PRIVATE KEY-----\n"	Приватный ключ в JSON-файле, сгенерированном при создании сервисной учётной записи.

- Чтобы выбрать метод аутентификации на основе имперсонации, настройте `StorageCredentialParams` следующим образом:

- Чтобы VM-инстанс имперсонировал сервисную учётную запись:

```
"gcp.gcs.use_compute_engine_service_account" = "true",
"gcp.gcs.impersonation_service_account" = "
<assumed_google_service_account_email>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.use_compute_engine_service_account	FALSE	TRUE	Указывает, следует ли напрямую использовать сервисную учётную запись, привязанную к вашему Compute Engine.
gcp.gcs.impersonation_service_account	""	"hello"	Сервисная учётная запись, которую вы хотите имперсонировать.

- Чтобы сервисная учётная запись (временно названная meta service account) имперсонировала другую сервисную учётную запись (временно названную data service account):

```
"gcp.gcs.service_account_email" = "<google_service_account_email>",
"gcp.gcs.service_account_private_key_id" = "
<meta_google_service_account_email>",
"gcp.gcs.service_account_private_key" = "
<meta_google_service_account_email>",
"gcp.gcs.impersonation_service_account" = "
<data_google_service_account_email>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.service_account_email	""	"user@hello.iam.gserviceaccount.com"	Адрес электронной почты в JSON-файле, сгенерированном при создании meta service account.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.service_account_private_key_id	""	"61d257bd8479547cb3e04f0b9b6b9ca07af3b7ea"	ID приватного ключа в JSON-файле, сгенерированном при создании meta service account.
gcp.gcs.service_account_private_key	""	"-----BEGIN PRIVATE KEY-----xxxx-----END PRIVATE KEY-----\n"	Приватный ключ в JSON-файле, сгенерированном при создании meta service account.
gcp.gcs.impersonation_service_account	""	"hello"	Data service account, которую вы хотите имперсонировать.

Примеры

Следующие примеры создают Paimon catalog с именем `paimon_catalog_fs`, тип metastore которого `paimon.catalog.type` установлен на `filesystem`, для запроса данных из вашего Paimon cluster.

AWS S3

- Если вы выбираете метод аутентификации на основе instance profile, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG paimon_catalog_fs
PROPERTIES
(
    "type" = "paimon",
    "paimon.catalog.type" = "filesystem",
    "paimon.catalog.warehouse" = "<s3_paimon_warehouse_path>",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.endpoint" = "<s3_endpoint>"
);
```

- Если вы выбираете метод аутентификации на основе assumed role, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG paimon_catalog_fs
PROPERTIES
(
    "type" = "paimon",
    "paimon.catalog.type" = "filesystem",
    "paimon.catalog.warehouse" = "<s3_paimon_warehouse_path>",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.endpoint" = "<s3_endpoint>"
);
```

- Если вы выбираете метод аутентификации на основе IAM user, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG paimon_catalog_fs
PROPERTIES
(
    "type" = "paimon",
    "paimon.catalog.type" = "filesystem",
    "paimon.catalog.warehouse" = "<s3_paimon_warehouse_path>",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_secret_key>",
    "aws.s3.endpoint" = "<s3_endpoint>"
);
```

S3-совместимая система хранения

На примере MinIO. Выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG paimon_catalog_fs
PROPERTIES
(
    "type" = "paimon",
    "paimon.catalog.type" = "filesystem",
    "paimon.catalog.warehouse" = "<paimon_warehouse_path>",
    "aws.s3.enable_ssl" = "true",
    "aws.s3.enable_path_style_access" = "true",
    "aws.s3.endpoint" = "<s3_endpoint>",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_secret_key>"
);
```

Microsoft Azure Storage

Azure Blob Storage

- Если вы выбираете метод аутентификации Shared Key, выполните команду, подобную следующей:

  ```
  CREATE EXTERNAL CATALOG paimon_catalog_fs
  PROPERTIES
  (
      "type" = "paimon",
      "paimon.catalog.type" = "filesystem",
      "paimon.catalog.warehouse" = "<blob_paimon_warehouse_path>",
      "azure.blob.storage_account" = "<blob_storage_account_name>",
      "azure.blob.shared_key" = "<blob_storage_account_shared_key>"
  );
  ```

- Если вы выбираете метод аутентификации SAS Token, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG paimon_catalog_fs
PROPERTIES
(
    "type" = "paimon",
    "paimon.catalog.type" = "filesystem",
    "paimon.catalog.warehouse" = "<blob_paimon_warehouse_path>",
    "azure.blob.storage_account" = "<blob_storage_account_name>",
    "azure.blob.container" = "<blob_container_name>",
    "azure.blob.sas_token" = "<blob_storage_account_SAS_token>"
);
```

Azure Data Lake Storage Gen1

- Если вы выбираете метод аутентификации Managed Service Identity, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG paimon_catalog_fs
PROPERTIES
(
    "type" = "paimon",
    "paimon.catalog.type" = "filesystem",
    "paimon.catalog.warehouse" = "<adls1_paimon_warehouse_path>",
    "azure.adls1.use_managed_service_identity" = "true"
);
```

- Если вы выбираете метод аутентификации Service Principal, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG paimon_catalog_fs
PROPERTIES
(
    "type" = "paimon",
    "paimon.catalog.type" = "filesystem",
    "paimon.catalog.warehouse" = "<adls1_paimon_warehouse_path>",
    "azure.adls1.oauth2_client_id" = "<application_client_id>",
    "azure.adls1.oauth2_credential" = "
<application_client_credential>",
    "azure.adls1.oauth2_endpoint" = "
<OAuth_2.0_authorization_endpoint_v2>"
);
```

Azure Data Lake Storage Gen2

- Если вы выбираете метод аутентификации Managed Identity, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG paimon_catalog_fs
PROPERTIES
(
    "type" = "paimon",
    "paimon.catalog.type" = "filesystem",
    "paimon.catalog.warehouse" = "<adls2_paimon_warehouse_path>",
    "azure.adls2.oauth2_use_managed_identity" = "true",
    "azure.adls2.oauth2_tenant_id" = "<service_principal_tenant_id>",
    "azure.adls2.oauth2_client_id" = "<service_client_id>"
);
```

- Если вы выбираете метод аутентификации Shared Key, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG paimon_catalog_fs
PROPERTIES
(
    "type" = "paimon",
    "paimon.catalog.type" = "filesystem",
    "paimon.catalog.warehouse" = "<adls2_paimon_warehouse_path>",
    "azure.adls2.storage_account" = "<storage_account_name>",
    "azure.adls2.shared_key" = "<shared_key>"
);
```

- Если вы выбираете метод аутентификации Service Principal, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG paimon_catalog_fs
PROPERTIES
(
    "type" = "paimon",
    "paimon.catalog.type" = "filesystem",
    "paimon.catalog.warehouse" = "<adls2_paimon_warehouse_path>",
    "azure.adls2.oauth2_client_id" = "<service_client_id>",
    "azure.adls2.oauth2_client_secret" = "
<service_principal_client_secret>",
    "azure.adls2.oauth2_client_endpoint" = "
<service_principal_client_endpoint>"
);
```

Google GCS

- Если вы выбираете метод аутентификации на основе VM, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG paimon_catalog_fs
PROPERTIES
(
    "type" = "paimon",
    "paimon.catalog.type" = "filesystem",
    "paimon.catalog.warehouse" = "<gcs_paimon_warehouse_path>",
    "gcp.gcs.use_compute_engine_service_account" = "true"
);
```

- Если вы выбираете метод аутентификации на основе сервисной учётной записи, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG paimon_catalog_fs
PROPERTIES
(
    "type" = "paimon",
    "paimon.catalog.type" = "filesystem",
    "paimon.catalog.warehouse" = "<gcs_paimon_warehouse_path>",
    "gcp.gcs.service_account_email" = "<google_service_account_email>",
    "gcp.gcs.service_account_private_key_id" = "
<google_service_private_key_id>",
    "gcp.gcs.service_account_private_key" = "
<google_service_private_key>"
);
```

- Если вы выбираете метод аутентификации на основе имперсонации:
  - Если вы настраиваете VM-инстанс на имперсонацию сервисной учётной записи, выполните команду, подобную следующей:

    ```
    CREATE EXTERNAL CATALOG paimon_catalog_fs
    PROPERTIES
    (
        "type" = "paimon",
        "paimon.catalog.type" = "filesystem",
        "paimon.catalog.warehouse" = "<gcs_paimon_warehouse_path>",
        "gcp.gcs.use_compute_engine_service_account" = "true",
        "gcp.gcs.impersonation_service_account" = "
    <assumed_google_service_account_email>"
    );
    ```

  - Если вы настраиваете сервисную учётную запись на имперсонацию другой сервисной учётной записи, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG paimon_catalog_fs
PROPERTIES
(
    "type" = "paimon",
    "paimon.catalog.type" = "filesystem",
    "paimon.catalog.warehouse" = "<gcs_paimon_warehouse_path>",
    "gcp.gcs.service_account_email" = "
<google_service_account_email>",
    "gcp.gcs.service_account_private_key_id" = "
<meta_google_service_account_email>",
    "gcp.gcs.service_account_private_key" = "
<meta_google_service_account_email>",
    "gcp.gcs.impersonation_service_account" = "
<data_google_service_account_email>"
);
```

Просмотр Paimon catalog

Вы можете использовать SHOW CATALOGS для запроса всех catalog в текущем Selena cluster:

```
SHOW CATALOGS;
```

Вы также можете использовать SHOW CREATE CATALOG для запроса оператора создания внешнего catalog. Следующий пример запрашивает оператор создания Paimon catalog с именем `paimon_catalog_fs`:

```
SHOW CREATE CATALOG paimon_catalog_fs;
```

Удаление Paimon catalog

Вы можете использовать DROP CATALOG для удаления внешнего catalog.

Следующий пример удаляет Paimon catalog с именем `paimon_catalog_fs`:

```
DROP Catalog paimon_catalog_fs;
```

Просмотр схемы таблицы Paimon

Вы можете использовать один из следующих синтаксисов для просмотра схемы таблицы Paimon:

- Просмотр схемы

```
DESC[RIBE] <catalog_name>.<database_name>.<table_name>;
```

- Просмотр схемы и расположения из оператора CREATE

```
SHOW CREATE TABLE <catalog_name>.<database_name>.<table_name>;
```

Запрос таблицы Paimon

1. Используйте SHOW DATABASES для просмотра баз данных в вашем Paimon cluster:

```
SHOW DATABASES FROM <catalog_name>;
```

2. Используйте SET CATALOG для переключения на целевой catalog в текущей сессии:

```
SET CATALOG <catalog_name>;
```

   Затем используйте USE для указания активной базы данных в текущей сессии:

```
USE <db_name>;
```

   Или вы можете использовать USE для прямого указания активной базы данных в целевом catalog:

```
USE <catalog_name>.<db_name>;
```

3. Используйте SELECT для запроса целевой таблицы в указанной базе данных:

```
SELECT count(*) FROM <table_name> LIMIT 10;
```

Загрузка данных из Paimon

Предположим, у вас есть OLAP-таблица с именем `olap_tbl`, вы можете трансформировать и загрузить данные следующим образом:

```
INSERT INTO default_catalog.olap_db.olap_tbl SELECT * FROM paimon_table;
```

5.7.10 Unified catalog

> *Beta* **BETA FEATURE**
>
> Рекомендации по использованию бета-функций

Unified catalog — это тип внешнего catalog, предоставляемый Selena начиная с версии v1.5.2 для обработки таблиц из источников данных Apache Hive™, Apache Iceberg, Apache Hudi, Delta Lake и Apache Kudu как единого источника данных без загрузки. С помощью unified catalog вы можете:

- Напрямую запрашивать данные, хранящиеся в Hive, Iceberg, Hudi, Delta Lake, Paimon и Kudu, без необходимости создания таблиц вручную.
- Использовать INSERT INTO или асинхронные материализованные представления (поддерживаются с версии v1.5.2) для обработки данных, хранящихся в Hive, Iceberg, Hudi, Delta Lake, Paimon и Kudu, и загрузки данных в Selena.
- Выполнять операции в Selena для создания или удаления баз данных и таблиц Hive и Iceberg.

Для успешного выполнения SQL-запросов к вашему единому источнику данных ваш Selena cluster должен иметь доступ к системе хранения и metastore вашего единого источника данных. Selena поддерживает следующие системы хранения и metastore:

- Распределённая файловая система (HDFS) или объектное хранилище, такое как AWS S3, Microsoft Azure Storage, Google GCS, или другие S3-совместимые системы хранения (например, MinIO)

- Metastore, такие как Hive metastore или AWS Glue

> **ПРИМЕЧАНИЕ**
>
> Если вы выбираете AWS S3 в качестве хранилища, вы можете использовать HMS или AWS Glue в качестве metastore. Если вы выбираете любую другую систему хранения, вы можете использовать только HMS в качестве metastore.

Ограничения

Один unified catalog поддерживает интеграцию только с одной системой хранения и одним сервисом metastore. Поэтому убедитесь, что все источники данных, которые вы хотите интегрировать как единый источник данных с Selena, используют одну и ту же систему хранения и сервис metastore.

Примечания по использованию

- См. раздел «Примечания по использованию» в Hive catalog, Iceberg catalog, Hudi catalog, Delta Lake catalog, Paimon catalog и Kudu catalog, чтобы понять поддерживаемые форматы файлов и типы данных.

- Операции, специфичные для формата, поддерживаются только для определённых форматов таблиц. Например, CREATE TABLE и DROP TABLE поддерживаются только для Hive и Iceberg, а REFRESH EXTERNAL TABLE поддерживается только для Hive и Hudi.

  При создании таблицы в unified catalog с помощью оператора CREATE TABLE используйте параметр `ENGINE` для указания формата таблицы (Hive или Iceberg).

Подготовка к интеграции

Перед созданием unified catalog убедитесь, что ваш Selena cluster может интегрироваться с системой хранения и metastore вашего единого источника данных.

AWS IAM

Если вы используете AWS S3 в качестве хранилища или AWS Glue в качестве metastore, выберите подходящий метод аутентификации и выполните необходимую подготовку, чтобы ваш Selena cluster мог получить доступ к соответствующим облачным ресурсам AWS. Подробнее см. в разделе Аутентификация к ресурсам AWS - Подготовка.

HDFS

Если вы выбрали HDFS в качестве хранилища, настройте ваш Selena cluster следующим образом:

- (Опционально) Установите имя пользователя, которое используется для доступа к вашему HDFS cluster и Hive metastore. По умолчанию Selena использует имя пользователя процессов FE и BE или CN для доступа к вашему HDFS cluster и Hive metastore. Вы также можете задать имя пользователя, добавив `export HADOOP_USER_NAME="<user_name>"` в начало файла **fe/conf/hadoop_env.sh** каждого FE и в начало файла **be/conf/hadoop_env.sh** каждого BE или файла **cn/conf/hadoop_env.sh** каждого CN. После установки имени пользователя в этих файлах перезапустите каждый FE и каждый BE или CN, чтобы настройки параметров вступили в силу. Для каждого Selena cluster можно задать только одно имя пользователя.
- При запросе данных FE и BE или CN вашего Selena cluster используют HDFS-клиент для доступа к вашему HDFS cluster. В большинстве случаев вам не нужно настраивать ваш Selena cluster для этого, и Selena запускает HDFS-клиент с настройками по умолчанию. Настройка Selena cluster необходима только в следующих ситуациях:
  - Для вашего HDFS cluster включена высокая доступность (HA): Добавьте файл **hdfs-site.xml** вашего HDFS cluster в путь **$FE_HOME/conf** каждого FE и в путь **$BE_HOME/conf** каждого BE или путь **$CN_HOME/conf** каждого CN.
  - Для вашего HDFS cluster включена View File System (ViewFs): Добавьте файл **core-site.xml** вашего HDFS cluster в путь **$FE_HOME/conf** каждого FE и в путь **$BE_HOME/conf** каждого BE или путь **$CN_HOME/conf** каждого CN.

> **ПРИМЕЧАНИЕ**
>
> Если при отправке запроса возвращается ошибка о неизвестном хосте, необходимо добавить соответствие между именами хостов и IP-адресами узлов вашего HDFS cluster в путь **/etc/hosts**.

Аутентификация Kerberos

Если для вашего HDFS cluster или Hive metastore включена аутентификация Kerberos, настройте ваш Selena cluster следующим образом:

- Выполните команду `kinit -kt keytab_path principal` на каждом FE и каждом BE или CN, чтобы получить Ticket Granting Ticket (TGT) от Key Distribution Center (KDC). Для выполнения этой команды у вас должны быть разрешения на доступ к вашему HDFS cluster и Hive metastore. Обратите внимание, что доступ к KDC с помощью этой команды ограничен по времени. Поэтому необходимо использовать cron для периодического выполнения этой команды.
- Добавьте `JAVA_OPTS="-Djava.security.krb5.conf=/etc/krb5.conf"` в файл **$FE_HOME/conf/fe.conf** каждого FE и в файл **$BE_HOME/conf/be.conf** каждого BE или файл **$CN_HOME/conf/cn.conf** каждого CN. В этом примере `/etc/krb5.conf` — это путь сохранения файла krb5.conf. Вы можете изменить путь в соответствии с вашими потребностями.

Создание unified catalog

Синтаксис

```
CREATE EXTERNAL CATALOG <catalog_name>
[COMMENT <comment>]
PROPERTIES
(
    "type" = "unified",
    MetastoreParams,
    StorageCredentialParams,
    MetadataUpdateParams,
    PaimonCatalogParams,
    KuduCatalogParams
)
```

Параметры

catalog_name

Имя unified catalog. Соглашения об именовании следующие:

- Имя может содержать буквы, цифры (0-9) и символы подчёркивания (_). Оно должно начинаться с буквы.
- Имя чувствительно к регистру и не может превышать 1023 символа в длину.

comment

Описание unified catalog. Этот параметр является опциональным.

type

Тип вашего источника данных. Установите значение `unified`.

MetastoreParams

Набор параметров о том, как Selena интегрируется с вашим metastore.

Hive metastore

Если вы выбрали Hive metastore в качестве metastore вашего единого источника данных, настройте `MetastoreParams` следующим образом:

```
"unified.metastore.type" = "hive",
"hive.metastore.uris" = "<hive_metastore_uri>"
```

> **ПРИМЕЧАНИЕ**
>
> Перед запросом данных необходимо добавить соответствие между именами хостов и IP-адресами узлов вашего Hive metastore в путь **/etc/hosts**. В противном случае Selena может не получить доступ к вашему Hive metastore при запуске запроса.

В следующей таблице описаны параметры, которые необходимо настроить в `MetastoreParams`.

Параметр	Обязательный	Описание
unified.metastore.type	Да	Тип metastore, который вы используете для вашего единого источника данных. Установите значение `hive`.
hive.metastore.uris	Да	URI вашего Hive metastore. Формат: `thrift://<metastore_IP_address>:<metastore_port>`. Если для вашего Hive metastore включена высокая доступность (HA), вы можете указать несколько URI metastore и разделить их запятыми (`,`), например, `"thrift://<metastore_IP_address_1>:<metastore_port_1>,thrift://<metastore_IP_address_2>:<metastore_port_2>,thrift://<metastore_IP_address_3>:<metastore_port_3>"`.

AWS Glue

Если вы выбрали AWS Glue в качестве metastore вашего источника данных, что поддерживается только при выборе AWS S3 в качестве хранилища, выполните одно из

следующих действий:

- Чтобы выбрать метод аутентификации на основе instance profile, настройте `MetastoreParams` следующим образом:

```
"unified.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "true",
"aws.glue.region" = "<aws_glue_region>"
```

- Чтобы выбрать метод аутентификации на основе assumed role, настройте `MetastoreParams` следующим образом:

```
"unified.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "true",
"aws.glue.iam_role_arn" = "<iam_role_arn>",
"aws.glue.region" = "<aws_glue_region>"
```

- Чтобы выбрать метод аутентификации на основе IAM user, настройте `MetastoreParams` следующим образом:

```
"unified.metastore.type" = "glue",
"aws.glue.use_instance_profile" = "false",
"aws.glue.access_key" = "<iam_user_access_key>",
"aws.glue.secret_key" = "<iam_user_secret_key>",
"aws.glue.region" = "<aws_s3_region>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `MetastoreParams`.

Параметр	Обязательный	Описание
unified.metastore.type	Да	Тип metastore, который вы используете для вашего единого источника данных. Установите значение `glue`.
aws.glue.use_instance_profile	Да	Указывает, следует ли включить метод аутентификации на основе instance profile и метод аутентификации на основе assumed role. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`.

Параметр	Обязательный	Описание
aws.glue.iam_role_arn	Нет	ARN IAM role, которая имеет привилегии на ваш AWS Glue Data Catalog. Если вы используете метод аутентификации на основе assumed role для доступа к AWS Glue, вы должны указать этот параметр.
aws.glue.region	Да	Регион, в котором находится ваш AWS Glue Data Catalog. Пример: `us-west-1`.
aws.glue.access_key	Нет	Access key вашего AWS IAM user. Если вы используете метод аутентификации на основе IAM user для доступа к AWS Glue, вы должны указать этот параметр.
aws.glue.secret_key	Нет	Secret key вашего AWS IAM user. Если вы используете метод аутентификации на основе IAM user для доступа к AWS Glue, вы должны указать этот параметр.

Информацию о том, как выбрать метод аутентификации для доступа к AWS Glue и как настроить политику управления доступом в AWS IAM Console, см. в разделе Параметры аутентификации для доступа к AWS Glue.

StorageCredentialParams

Набор параметров о том, как Selena интегрируется с вашей системой хранения. Этот набор параметров является опциональным.

Если вы используете HDFS в качестве хранилища, вам не нужно настраивать `StorageCredentialParams`.

Если вы используете AWS S3, другую S3-совместимую систему хранения, Microsoft Azure Storage или Google GCS в качестве хранилища, вы должны настроить `StorageCredentialParams`.

AWS S3

Если вы выбрали AWS S3 в качестве хранилища, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации на основе instance profile, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.region" = "<aws_s3_region>"
```

- Чтобы выбрать метод аутентификации на основе assumed role, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.iam_role_arn" = "<iam_role_arn>",
"aws.s3.region" = "<aws_s3_region>"
```

- Чтобы выбрать метод аутентификации на основе IAM user, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "false",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>",
"aws.s3.region" = "<aws_s3_region>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
aws.s3.use_instance_profile	Да	Указывает, следует ли включить метод аутентификации на основе instance profile и метод аутентификации на основе assumed role. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`.
aws.s3.iam_role_arn	Нет	ARN IAM role, которая имеет привилегии на ваш AWS S3 bucket. Если вы используете метод аутентификации на основе assumed role для доступа к AWS S3, вы должны указать этот параметр.
aws.s3.region	Да	Регион, в котором находится ваш AWS S3 bucket. Пример: `us-west-1`.
aws.s3.access_key	Нет	Access key вашего IAM user. Если вы используете метод аутентификации на основе IAM user для доступа к AWS S3, вы должны указать этот параметр.
aws.s3.secret_key	Нет	Secret key вашего IAM user. Если вы используете метод аутентификации на основе IAM user для доступа к AWS S3, вы должны указать этот параметр.

Информацию о том, как выбрать метод аутентификации для доступа к AWS S3 и как настроить политику управления доступом в AWS IAM Console, см. в разделе Параметры аутентификации для доступа к AWS S3.

S3-совместимая система хранения

Если вы выбрали S3-совместимую систему хранения, такую как MinIO, в качестве хранилища, настройте `StorageCredentialParams` следующим образом для обеспечения успешной интеграции:

```
"aws.s3.enable_ssl" = "false",
"aws.s3.enable_path_style_access" = "true",
"aws.s3.endpoint" = "<s3_endpoint>",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
aws.s3.enable_ssl	Да	Указывает, следует ли включить SSL-соединение. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`.
aws.s3.enable_path_style_access	Да	Указывает, следует ли включить доступ в стиле пути. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`. Для MinIO вы должны установить значение `true`. URL в стиле пути используют следующий формат: `https://s3.<region_code>.amazonaws.com/<bucket_name>/<key_name>`. Например, если вы создали bucket с именем `DOC-EXAMPLE-BUCKET1` в регионе US West (Oregon) и хотите получить доступ к объекту `alice.jpg` в этом bucket, вы можете использовать следующий URL в стиле пути: `https://s3.us-west-2.amazonaws.com/DOC-EXAMPLE-BUCKET1/alice.jpg`.
aws.s3.endpoint	Да	Endpoint, который используется для подключения к вашей S3-совместимой системе хранения вместо AWS S3.
aws.s3.access_key	Да	Access key вашего IAM user.
aws.s3.secret_key	Да	Secret key вашего IAM user.

Microsoft Azure Storage

Azure Blob Storage

Если вы выбрали Blob Storage в качестве хранилища, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Shared Key, настройте `StorageCredentialParams` следующим образом:

```
"azure.blob.storage_account" = "<storage_account_name>",
"azure.blob.shared_key" = "<storage_account_shared_key>"
```

  В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.blob.storage_account	Да	Имя пользователя вашей учётной записи Blob Storage.
azure.blob.shared_key	Да	Shared key вашей учётной записи Blob Storage.

- Чтобы выбрать метод аутентификации SAS Token, настройте `StorageCredentialParams` следующим образом:

```
"azure.blob.storage_account" = "<storage_account_name>",
"azure.blob.container" = "<container_name>",
"azure.blob.sas_token" = "<storage_account_SAS_token>"
```

  В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.blob.storage_account	Да	Имя пользователя вашей учётной записи Blob Storage.
azure.blob.container	Да	Имя blob-контейнера, в котором хранятся ваши данные.
azure.blob.sas_token	Да	SAS token, который используется для доступа к вашей учётной записи Blob Storage.

Azure Data Lake Storage Gen2

Если вы выбрали Data Lake Storage Gen2 в качестве хранилища, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Managed Identity, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.oauth2_use_managed_identity" = "true",
"azure.adls2.oauth2_tenant_id" = "<service_principal_tenant_id>",
"azure.adls2.oauth2_client_id" = "<service_client_id>"
```

  В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls2.oauth2_use_managed_identity	Да	Указывает, следует ли включить метод аутентификации Managed Identity. Установите значение `true`.
azure.adls2.oauth2_tenant_id	Да	ID tenant, к данным которого вы хотите получить доступ.
azure.adls2.oauth2_client_id	Да	Client (application) ID управляемого удостоверения.

- Чтобы выбрать метод аутентификации Shared Key, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.storage_account" = "<storage_account_name>",
"azure.adls2.shared_key" = "<storage_account_shared_key>"
```

  В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls2.storage_account	Да	Имя пользователя вашей учётной записи хранения Data Lake Storage Gen2.
azure.adls2.shared_key	Да	Shared key вашей учётной записи хранения Data Lake Storage Gen2.

- Чтобы выбрать метод аутентификации Service Principal, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls2.oauth2_client_id" = "<service_client_id>",
"azure.adls2.oauth2_client_secret" = "
<service_principal_client_secret>",
"azure.adls2.oauth2_client_endpoint" = "
<service_principal_client_endpoint>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls2.oauth2_client_id	Да	Client (application) ID service principal.
azure.adls2.oauth2_client_secret	Да	Значение нового секрета клиента (приложения), который был создан.
azure.adls2.oauth2_client_endpoint	Да	OAuth 2.0 token endpoint (v1) service principal или приложения.

Azure Data Lake Storage Gen1

Если вы выбрали Data Lake Storage Gen1 в качестве хранилища, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации Managed Service Identity, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls1.use_managed_service_identity" = "true"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls1.use_managed_service_identity	Да	Указывает, следует ли включить метод аутентификации Managed Service Identity. Установите значение `true`.

- Чтобы выбрать метод аутентификации Service Principal, настройте `StorageCredentialParams` следующим образом:

```
"azure.adls1.oauth2_client_id" = "<application_client_id>",
"azure.adls1.oauth2_credential" = "<application_client_credential>",
"azure.adls1.oauth2_endpoint" = "<OAuth_2.0_authorization_endpoint_v2>"
```

  В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
azure.adls1.oauth2_client_id	Да	Client (application) ID service principal.
azure.adls1.oauth2_credential	Да	Значение нового секрета клиента (приложения), который был создан.
azure.adls1.oauth2_endpoint	Да	OAuth 2.0 token endpoint (v1) service principal или приложения.

Google GCS

Если вы выбрали Google GCS в качестве хранилища, выполните одно из следующих действий:

- Чтобы выбрать метод аутентификации на основе VM, настройте `StorageCredentialParams` следующим образом:

```
"gcp.gcs.use_compute_engine_service_account" = "true"
```

  В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.use_compute_engine_service_account	false	true	Указывает, следует ли напрямую использовать сервисную учётную запись, привязанную к вашему Compute Engine.

- Чтобы выбрать метод аутентификации на основе сервисной учётной записи, настройте `StorageCredentialParams` следующим образом:

```
"gcp.gcs.service_account_email" = "<google_service_account_email>",
"gcp.gcs.service_account_private_key_id" = "
<google_service_private_key_id>",
"gcp.gcs.service_account_private_key" = "<google_service_private_key>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.service_account_email	""	"user@hello.iam.gserviceaccount.com"	Адрес электронной почты в JSON-файле, сгенерированном при создании сервисной учётной записи.
gcp.gcs.service_account_private_key_id	""	"61d257bd8479547cb3e04f0b9b6b9ca07af3b7ea"	ID приватного ключа в JSON-файле, сгенерированном при создании сервисной учётной записи.
gcp.gcs.service_account_private_key	""	"-----BEGIN PRIVATE KEY-----xxxx-----END PRIVATE KEY-----\n"	Приватный ключ в JSON-файле, сгенерированном при создании сервисной учётной записи.

- Чтобы выбрать метод аутентификации на основе имперсонации, настройте `StorageCredentialParams` следующим образом:

  - Чтобы VM-инстанс имперсонировал сервисную учётную запись:

```
"gcp.gcs.use_compute_engine_service_account" = "true",
"gcp.gcs.impersonation_service_account" = "
<assumed_google_service_account_email>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.use_compute_engine_service_account	false	true	Указывает, следует ли напрямую использовать сервисную учётную запись, привязанную к вашему Compute Engine.
gcp.gcs.impersonation_service_account	""	"hello"	Сервисная учётная запись, которую вы хотите имперсонировать.

- Чтобы сервисная учётная запись (временно названная meta service account) имперсонировала другую сервисную учётную запись (временно названную data service account):

```
"gcp.gcs.service_account_email" = "<google_service_account_email>",
"gcp.gcs.service_account_private_key_id" = "
<meta_google_service_account_email>",
"gcp.gcs.service_account_private_key" = "
<meta_google_service_account_email>",
"gcp.gcs.impersonation_service_account" = "
<data_google_service_account_email>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Значение по умолчанию	Пример значения	Описание
gcp.gcs.service_account_email	""	"user@hello.iam.gserviceaccount.com"	Адрес электронной почты в JSON-файле, сгенерированном при создании meta service account.
gcp.gcs.service_account_private_key_id	""	"61d257bd8479547cb3e04f0b9b6b9ca07af3b7ea"	ID приватного ключа в JSON-файле, сгенерированном при создании meta service account.
gcp.gcs.service_account_private_key	""	"-----BEGIN PRIVATE KEY-----xxxx-----END PRIVATE KEY-----\n"	Приватный ключ в JSON-файле, сгенерированном при создании meta service account.
gcp.gcs.impersonation_service_account	""	"hello"	Data service account, которую вы хотите имперсонировать.

MetadataUpdateParams

Набор параметров о том, как Selena обновляет кэшированные метаданные Hive, Hudi и Delta Lake. Этот набор параметров является опциональным. Подробнее о политиках обновления кэшированных метаданных из Hive, Hudi и Delta Lake см. в разделах Hive catalog, Hudi catalog и Delta Lake catalog.

В большинстве случаев вы можете игнорировать `MetadataUpdateParams` и не нуждаетесь в настройке параметров политики в нём, поскольку значения этих параметров по умолчанию уже обеспечивают готовую к использованию производительность.

Однако, если частота обновления данных в Hive, Hudi или Delta Lake высока, вы можете настроить эти параметры для дополнительной оптимизации производительности автоматических асинхронных обновлений.

Параметр	Обязательный	Описание
enable_metastore_cache	Нет	Указывает, кэширует ли Selena метаданные таблиц Hive, Hudi или Delta Lake. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`. Значение `true` включает кэш, значение `false` отключает кэш.
enable_remote_file_cache	Нет	Указывает, кэширует ли Selena метаданные базовых файлов данных таблиц или partition Hive, Hudi или Delta Lake. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`. Значение `true` включает кэш, значение `false` отключает кэш.
metastore_cache_refresh_interval_sec	Нет	Интервал времени, с которым Selena асинхронно обновляет метаданные таблиц или partition Hive, Hudi или Delta Lake, кэшированных в ней. Единица измерения: секунды. Значение по умолчанию: `7200`, то есть 2 часа.
remote_file_cache_refresh_interval_sec	Нет	Интервал времени, с которым Selena асинхронно обновляет метаданные базовых файлов данных таблиц или partition Hive, Hudi или Delta Lake, кэшированных в ней. Единица измерения: секунды. Значение по умолчанию: `60`.
metastore_cache_ttl_sec	Нет	Интервал времени, по истечении которого Selena автоматически удаляет метаданные таблиц или partition Hive, Hudi или Delta Lake, кэшированных в ней. Единица измерения: секунды. Значение по умолчанию: `86400`, то есть 24 часа.
remote_file_cache_ttl_sec	Нет	Интервал времени, по истечении которого Selena автоматически удаляет метаданные базовых файлов данных таблиц или partition Hive, Hudi или Delta Lake, кэшированных в ней. Единица измерения: секунды. Значение по умолчанию: `129600`, то есть 36 часов.

PaimonCatalogParams

Набор параметров о том, как подключиться к Paimon Catalog. Этот набор параметров является опциональным.

Параметр	Обязательный	Описание
paimon.catalog.warehouse	Нет	Путь к хранилищу warehouse ваших данных Paimon.

KuduCatalogParams

Набор параметров о том, как подключиться к Kudu Catalog. Этот набор параметров является опциональным.

Параметр	Обязательный	Описание
kudu.master	Нет	Указывает адрес Kudu Master, по умолчанию `localhost:7051`.
kudu.schema-emulation.enabled	Нет	Опция для включения или отключения эмуляции `schema`. По умолчанию выключена (false), что означает, что все таблицы принадлежат `schema` `default`.
kudu.schema-emulation.prefix	Нет	Префикс для эмуляции `schema` должен устанавливаться только когда `kudu.schema-emulation.enabled` = `true`. Префикс по умолчанию — пустая строка: `''`.

Примеры

Следующие примеры создают unified catalog с именем `unified_catalog_hms` или `unified_catalog_glue`, в зависимости от типа используемого вами metastore, для запроса данных из вашего единого источника данных.

HDFS

Если вы используете HDFS в качестве хранилища, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_hms
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
);
```

AWS S3

Аутентификация на основе instance profile

- Если вы используете Hive metastore, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_hms
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2"
);
```

- Если вы используете AWS Glue с Amazon EMR, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_glue
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2"
);
```

Аутентификация на основе assumed role

- Если вы используете Hive metastore, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_hms
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

- Если вы используете AWS Glue с Amazon EMR, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_glue
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "true",
    "aws.glue.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_glue_role",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::081976408565:role/test_s3_role",
    "aws.s3.region" = "us-west-2"
);
```

Аутентификация на основе IAM user

- Если вы используете Hive metastore, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_hms
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_access_key>",
    "aws.s3.region" = "us-west-2"
);
```

- Если вы используете AWS Glue с Amazon EMR, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_glue
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "glue",
    "aws.glue.use_instance_profile" = "false",
    "aws.glue.access_key" = "<iam_user_access_key>",
    "aws.glue.secret_key" = "<iam_user_secret_key>",
    "aws.glue.region" = "us-west-2",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_secret_key>",
    "aws.s3.region" = "us-west-2"
);
```

S3-совместимая система хранения

На примере MinIO. Выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_hms
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "aws.s3.enable_ssl" = "true",
    "aws.s3.enable_path_style_access" = "true",
    "aws.s3.endpoint" = "<s3_endpoint>",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_secret_key>"
);
```

Microsoft Azure Storage

Azure Blob Storage

- Если вы выбираете метод аутентификации Shared Key, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_hms
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.blob.storage_account" = "<blob_storage_account_name>",
    "azure.blob.shared_key" = "<blob_storage_account_shared_key>"
);
```

- Если вы выбираете метод аутентификации SAS Token, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_hms
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.blob.storage_account" = "<blob_storage_account_name>",
    "azure.blob.container" = "<blob_container_name>",
    "azure.blob.sas_token" = "<blob_storage_account_SAS_token>"
);
```

Azure Data Lake Storage Gen1

- Если вы выбираете метод аутентификации Managed Service Identity, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_hms
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls1.use_managed_service_identity" = "true"
);
```

- Если вы выбираете метод аутентификации Service Principal, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_hms
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls1.oauth2_client_id" = "<application_client_id>",
    "azure.adls1.oauth2_credential" = "
<application_client_credential>",
    "azure.adls1.oauth2_endpoint" = "
<OAuth_2.0_authorization_endpoint_v2>"
);
```

Azure Data Lake Storage Gen2

- Если вы выбираете метод аутентификации Managed Identity, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_hms
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls2.oauth2_use_managed_identity" = "true",
    "azure.adls2.oauth2_tenant_id" = "<service_principal_tenant_id>",
    "azure.adls2.oauth2_client_id" = "<service_client_id>"
);
```

- Если вы выбираете метод аутентификации Shared Key, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_hms
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls2.storage_account" = "<storage_account_name>",
    "azure.adls2.shared_key" = "<shared_key>"
);
```

- Если вы выбираете метод аутентификации Service Principal, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_hms
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "azure.adls2.oauth2_client_id" = "<service_client_id>",
    "azure.adls2.oauth2_client_secret" = "
<service_principal_client_secret>",
    "azure.adls2.oauth2_client_endpoint" = "
<service_principal_client_endpoint>"
);
```

Google GCS

- Если вы выбираете метод аутентификации на основе VM, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_hms
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "gcp.gcs.use_compute_engine_service_account" = "true"
);
```

- Если вы выбираете метод аутентификации на основе сервисной учётной записи, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_hms
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "gcp.gcs.service_account_email" = "<google_service_account_email>",
    "gcp.gcs.service_account_private_key_id" = "
<google_service_private_key_id>",
    "gcp.gcs.service_account_private_key" = "
<google_service_private_key>"
);
```

- Если вы выбираете метод аутентификации на основе имперсонации:
  - Если вы настраиваете VM-инстанс на имперсонацию сервисной учётной записи, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_hms
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "gcp.gcs.use_compute_engine_service_account" = "true",
    "gcp.gcs.impersonation_service_account" = "
<assumed_google_service_account_email>"
);
```

  - Если вы настраиваете сервисную учётную запись на имперсонацию другой сервисной учётной записи, выполните команду, подобную следующей:

```
CREATE EXTERNAL CATALOG unified_catalog_hms
PROPERTIES
(
    "type" = "unified",
    "unified.metastore.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "gcp.gcs.service_account_email" = "
<google_service_account_email>",
    "gcp.gcs.service_account_private_key_id" = "
<meta_google_service_account_email>",
    "gcp.gcs.service_account_private_key" = "
<meta_google_service_account_email>",
    "gcp.gcs.impersonation_service_account" = "
<data_google_service_account_email>"
);
```

Просмотр unified catalog

Вы можете использовать SHOW CATALOGS для запроса всех catalog в текущем Selena cluster:

```
SHOW CATALOGS;
```

Вы также можете использовать SHOW CREATE CATALOG для запроса оператора создания внешнего catalog. Следующий пример запрашивает оператор создания unified catalog с именем `unified_catalog_glue`:

```
SHOW CREATE CATALOG unified_catalog_glue;
```

Переключение на Unified Catalog и базу данных в нём

Вы можете использовать один из следующих методов для переключения на unified catalog и базу данных в нём:

- Используйте SET CATALOG для указания unified catalog в текущей сессии, а затем используйте USE для указания активной базы данных:

```
-- Переключение на указанный catalog в текущей сессии:
SET CATALOG <catalog_name>
-- Указание активной базы данных в текущей сессии:
USE <db_name>
```

- Напрямую используйте USE для переключения на unified catalog и базу данных в нём:

```
USE <catalog_name>.<db_name>
```

Удаление unified catalog

Вы можете использовать DROP CATALOG для удаления внешнего catalog.

Следующий пример удаляет unified catalog с именем `unified_catalog_glue`:

```
DROP CATALOG unified_catalog_glue;
```

Просмотр схемы таблицы из unified catalog

Вы можете использовать один из следующих синтаксисов для просмотра схемы таблицы из unified catalog:

- Просмотр схемы

```
DESC[RIBE] <catalog_name>.<database_name>.<table_name>
```

- Просмотр схемы и расположения из оператора CREATE

```
SHOW CREATE TABLE <catalog_name>.<database_name>.<table_name>
```

Запрос данных из unified catalog

Для запроса данных из unified catalog выполните следующие шаги:

1. Используйте SHOW DATABASES для просмотра баз данных в вашем едином источнике данных, с которым связан unified catalog:

```
SHOW DATABASES FROM <catalog_name>
```

2. Переключитесь на Hive Catalog и базу данных в нём.

3. Используйте SELECT для запроса целевой таблицы в указанной базе данных:

```
SELECT count(*) FROM <table_name> LIMIT 10
```

Загрузка данных из Hive, Iceberg, Hudi, Delta Lake или Kudu

Вы можете использовать INSERT INTO для загрузки данных таблицы Hive, Iceberg, Hudi, Delta Lake или Kudu в таблицу Selena, созданную в unified catalog.

Следующий пример загружает данные таблицы Hive `hive_table` в таблицу Selena `test_tbl`, созданную в базе данных `test_database`, которая принадлежит unified catalog `unified_catalog`:

```
INSERT INTO unified_catalog.test_database.test_table SELECT * FROM
hive_table
```

Создание базы данных в unified catalog

Аналогично внутреннему catalog Selena, если у вас есть привилегия CREATE DATABASE на unified catalog, вы можете использовать оператор CREATE DATABASE для создания базы данных в этом catalog.

> **ПРИМЕЧАНИЕ**
>
> Вы можете предоставлять и отзывать привилегии с помощью GRANT и REVOKE.

Selena поддерживает создание только баз данных Hive и Iceberg в unified catalog.

Переключитесь на unified catalog, а затем используйте следующий оператор для создания базы данных в этом catalog:

```
CREATE DATABASE <database_name>
[properties ("location" = "
<prefix>://<path_to_database>/<database_name.db>")]
```

Параметр `location` указывает путь к файлу, в котором вы хотите создать базу данных, который может быть в HDFS или облачном хранилище.

- Когда вы используете Hive metastore в качестве metastore вашего источника данных, параметр `location` по умолчанию имеет значение `<warehouse_location>/<database_name.db>`, которое поддерживается Hive metastore, если вы не указываете этот параметр при создании базы данных.
- Когда вы используете AWS Glue в качестве metastore вашего источника данных, параметр `location` не имеет значения по умолчанию, и поэтому вы должны указать этот параметр при создании базы данных.

`prefix` варьируется в зависимости от используемой вами системы хранения:

Система хранения	Значение `prefix`
HDFS	`hdfs`
Google GCS	`gs`
Azure Blob Storage	• Если ваша учётная запись хранения разрешает доступ через HTTP, `prefix` — `wasb`. • Если ваша учётная запись хранения разрешает доступ через HTTPS, `prefix` — `wasbs`.
Azure Data Lake Storage Gen1	`adl`
Azure Data Lake Storage Gen2	• Если ваша учётная запись хранения разрешает доступ через HTTP, `prefix` — `abfs`. • Если ваша учётная запись хранения разрешает доступ через HTTPS, `prefix` — `abfss`.
AWS S3 или другое S3-совместимое хранилище (например, MinIO)	`s3`

Удаление базы данных из unified catalog

Аналогично внутренним базам данных Selena, если у вас есть привилегия DROP на базу данных, созданную в unified catalog, вы можете использовать оператор DROP DATABASE для удаления этой базы данных. Удалять можно только пустые базы данных.

> **ПРИМЕЧАНИЕ**
>
> Вы можете предоставлять и отзывать привилегии с помощью GRANT и REVOKE.

Selena поддерживает удаление только баз данных Hive и Iceberg из unified catalog.

При удалении базы данных из unified catalog путь к файлам базы данных в вашем HDFS cluster или облачном хранилище не удаляется вместе с базой данных.

Переключитесь на unified catalog, а затем используйте следующий оператор для удаления базы данных в этом catalog:

```
DROP DATABASE <database_name>
```

Создание таблицы в unified catalog

Аналогично внутренним базам данных Selena, если у вас есть привилегия CREATE TABLE на базу данных, созданную в unified catalog, вы можете использовать оператор CREATE TABLE или CREATE TABLE AS SELECT для создания таблицы в этой базе данных.

> **ПРИМЕЧАНИЕ**
>
> Вы можете предоставлять и отзывать привилегии с помощью GRANT и REVOKE.

Selena поддерживает создание только таблиц Hive и Iceberg в unified catalog.

Переключитесь на Hive Catalog и базу данных в нём. Затем используйте CREATE TABLE для создания таблицы Hive или Iceberg в этой базе данных:

```
CREATE TABLE <table_name>
(column_definition1[, column_definition2, ...]
ENGINE = {|hive|iceberg}
[partition_desc]
```

Подробнее см. в разделах Создание таблицы Hive и Создание таблицы Iceberg.

Следующий пример создаёт таблицу Hive с именем `hive_table`. Таблица состоит из трёх столбцов `action`, `id` и `dt`, из которых `id` и `dt` являются столбцами partition.

```
CREATE TABLE hive_table
(
    action varchar(65533),
    id int,
    dt date
)
ENGINE = hive
PARTITION BY (id,dt);
```

Загрузка данных в таблицу в unified catalog

Аналогично внутренним таблицам Selena, если у вас есть привилегия INSERT на таблицу, созданную в unified catalog, вы можете использовать оператор INSERT для загрузки данных таблицы Selena в эту таблицу Unified Catalog (в настоящее время поддерживаются только таблицы Unified Catalog в формате Parquet).

> **ПРИМЕЧАНИЕ**
>
> Вы можете предоставлять и отзывать привилегии с помощью GRANT и REVOKE.

Selena поддерживает загрузку данных только в таблицы Hive и Iceberg в unified catalog.

Переключитесь на Hive Catalog и базу данных в нём. Затем используйте INSERT INTO для вставки данных в таблицу Hive или Iceberg в этой базе данных:

```
INSERT {INTO | OVERWRITE} <table_name>
[ (column_name [, ...]) ]
{ VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }

-- Если вы хотите загрузить данные в указанные partition, используйте
следующий синтаксис:
INSERT {INTO | OVERWRITE} <table_name>
PARTITION (par_col1=<value> [, par_col2=<value>...])
{ VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }
```

Подробнее см. в разделах Загрузка данных в таблицу Hive и Загрузка данных в таблицу Iceberg.

Следующий пример вставляет три строки данных в таблицу Hive с именем `hive_table`:

```
INSERT INTO hive_table
VALUES
    ("buy", 1, "2023-09-01"),
    ("sell", 2, "2023-09-02"),
    ("buy", 3, "2023-09-03");
```

Удаление таблицы из unified catalog

Аналогично внутренним таблицам Selena, если у вас есть привилегия DROP на таблицу, созданную в unified catalog, вы можете использовать оператор DROP TABLE для удаления этой таблицы.

> **ПРИМЕЧАНИЕ**
>
> Вы можете предоставлять и отзывать привилегии с помощью GRANT и REVOKE.

Selena поддерживает удаление только таблиц Hive и Iceberg из unified catalog.

Переключитесь на Hive Catalog и базу данных в нём. Затем используйте DROP TABLE для удаления таблицы Hive или Iceberg в этой базе данных:

```
DROP TABLE <table_name>
```

Подробнее см. в разделах Удаление таблицы Hive и Удаление таблицы Iceberg.

Следующий пример удаляет таблицу Hive с именем `hive_table`:

```
DROP TABLE hive_table FORCE
```

5.7.11 MaxCompute catalog

> *Beta* **BETA FEATURE**
>
> Рекомендации по использованию бета-функций

Selena поддерживает catalogs Alibaba Cloud MaxCompute (ранее известные как ODPS) начиная с версии v1.5.2.

MaxCompute catalog — это тип внешнего catalog, который позволяет запрашивать данные из MaxCompute без загрузки.

С помощью MaxCompute catalogs вы также можете напрямую преобразовывать и загружать данные из MaxCompute с помощью INSERT INTO.

Примечания по использованию

Вы можете использовать MaxCompute catalogs только для запроса данных из MaxCompute. Вы не можете использовать MaxCompute catalogs для удаления, уничтожения или вставки данных в ваш MaxCompute cluster.

Подготовка к интеграции

Перед созданием MaxCompute catalog убедитесь, что ваш Selena cluster может правильно получить доступ к вашему сервису MaxCompute.

Создание MaxCompute catalog

Синтаксис

```
CREATE EXTERNAL CATALOG <catalog_name>
[COMMENT <comment>]
PROPERTIES
(
    "type" = "odps",
    CatalogParams,
    ScanParams,
    CachingMetaParams
)
```

Параметры

catalog_name

Название MaxCompute catalog. Соглашения об именовании следующие:

- Название может содержать буквы, цифры (0-9) и подчеркивания (_). Оно должно начинаться с буквы.
- Название чувствительно к регистру и не может превышать 1023 символа.

comment

Описание MaxCompute catalog. Этот параметр необязателен.

type

Тип вашего источника данных. Установите значение `odps`.

CatalogParams

Набор параметров о том, как Selena обращается к метаданным MaxCompute cluster.

Следующая таблица описывает параметр, который вам нужно настроить в `CatalogParams`.

Параметр	Обязательный	Описание
odps.endpoint	Да	Адрес подключения (а именно, endpoint) для сервиса MaxCompute. Вам нужно настроить endpoint в соответствии с регионом, выбранным при создании проекта MaxCompute, а также режимом сетевого подключения. Для получения подробной информации о endpoints, используемых в различных регионах и режимах сетевого подключения, см. Endpoint. Обратите внимание, что в настоящее время поддерживаются только два режима сетевого подключения Alibaba Cloud для обеспечения лучшего опыта: VPC и классическая сеть.
odps.project	Да	Название проекта MaxCompute, к которому вы хотите получить доступ. Если вы создали рабочее пространство в стандартном режиме, обратите внимание на различия между именами проектов для производственной среды и среды разработки (_dev) при настройке этого параметра. Вы можете войти в MaxCompute Console и получить имя проекта MaxCompute на странице **Workspace** > **Project Management**.
odps.access.id	Да	AccessKey ID учетной записи Alibaba Cloud или пользователя RAM. Вы можете перейти на страницу AccessKey Management, чтобы получить AccessKey ID.
odps.access.key	Да	AccessKey Secret, соответствующий AccessKey ID. Вы можете перейти на страницу AccessKey Management, чтобы получить AccessKey Secret.

Параметр	Обязательный	Описание
odps.tunnel.endpoint	Нет	Ссылка на публичный сетевой доступ для сервиса Tunnel. Если вы не настроили Tunnel endpoint, Tunnel автоматически направит на Tunnel endpoint, соответствующий сети, в которой находится сервис MaxCompute. Если вы настроили Tunnel endpoint, он будет использоваться в соответствии с настройками и не будет автоматически направлен.
odps.tunnel.quota	Да	Название квоты, которая используется для доступа к MaxCompute. MaxCompute предоставляет два типа ресурсов для передачи данных: эксклюзивная группа ресурсов для MaxCompute Tunnel (подписка) и storage API (оплата по мере использования). Вы можете выполнить следующие операции для получения имени квоты в зависимости от типа ресурса. **Эксклюзивная группа ресурсов для MaxCompute Tunnel**: Войдите в MaxCompute console. В верхней панели навигации выберите регион. В боковой панели навигации выберите Workspace > Quotas, чтобы просмотреть доступные квоты. Для получения дополнительной информации см. Manage quotas for computing resources in the MaxCompute console. **Storage API**: Войдите в MaxCompute console. В боковой панели навигации выберите Tenants > Tenant Property. На странице Tenants включите Storage API Switch. Для получения дополнительной информации см. Use storage API (pay-as-you-go). Название по умолчанию storage API — **"pay-as-you-go"**.

ScanParams

Набор параметров о том, как Selena обращается к файлам, хранящимся в MaxCompute cluster. Этот набор параметров необязателен.

Следующая таблица описывает параметр, который вам нужно настроить в `ScanParams`.

Параметр	Обязательный	Описание
odps.split.policy	Нет	Политика разделения, используемая при сканировании данных. Допустимые значения: `size` (разделение по размеру данных) и `row_offset` (разделение по количеству строк). Значение по умолчанию: `size`.
odps.split.row.count	Нет	Максимальное количество строк на раздел, когда `odps.split.policy` установлен на `row_offset`. Значение по умолчанию: `4 * 1024 * 1024 = 4194304`.

CachingMetaParams

Набор параметров о том, как Selena кэширует метаданные Hive. Этот набор параметров необязателен.

Следующая таблица описывает параметр, который вам нужно настроить в `CachingMetaParams`.

Параметр	Обязательный	Описание
odps.cache.table.enable	Нет	Указывает, кэширует ли Selena метаданные таблиц MaxCompute. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`. Значение `true` включает кэш, а значение `false` отключает кэш.
odps.cache.table.expire	Нет	Интервал времени в секундах, по истечении которого Selena автоматически удаляет кэшированные метаданные таблиц или разделов MaxCompute. Значение по умолчанию: `86400` (24 часа).
odps.cache.table.size	Нет	Количество записей метаданных таблиц MaxCompute, которые Selena кэширует. Значение по умолчанию: `1000`.
odps.cache.partition.enable	Нет	Указывает, кэширует ли Selena метаданные всех разделов для таблицы MaxCompute. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`. Значение `true` включает кэш, а значение `false` отключает кэш.
odps.cache.partition.expire	Нет	Интервал времени в секундах, по истечении которого Selena автоматически удаляет кэшированные метаданные всех разделов для таблицы MaxCompute. Значение по умолчанию: `86400` (24 часа).
odps.cache.partition.size	Нет	Количество таблиц MaxCompute, для которых Selena кэширует метаданные всех разделов. Значение по умолчанию: `1000`.
odps.cache.table-name.enable	Нет	Указывает, кэширует ли Selena информацию о таблицах из проекта MaxCompute. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`. Значение `true` включает кэш, а значение `false` отключает кэш.
odps.cache.table-name.expire	Нет	Интервал времени в секундах, по истечении которого Selena автоматически удаляет кэшированную информацию о таблицах из проекта MaxCompute. Значение по умолчанию: `86400` (24 часа).
odps.cache.table-name.size	Нет	Количество проектов MaxCompute, которые Selena кэширует. Значение по умолчанию: `1000`.

Примеры

Следующий пример создает MaxCompute catalog с именем `odps_catalog`, который использует `odps_project` в качестве проекта хранилища.

```
CREATE EXTERNAL CATALOG odps_catalog
PROPERTIES (
   "type"="odps",
   "odps.access.id"="<maxcompute_user_access_id>",
   "odps.access.key"="<maxcompute_user_access_key>",
   "odps.endpoint"="<maxcompute_server_endpoint>",
   "odps.project"="odps_project"
);
```

Просмотр MaxCompute catalogs

Вы можете использовать SHOW CATALOGS для запроса всех catalogs в текущем Selena cluster:

```
SHOW CATALOGS;
```

Вы также можете использовать SHOW CREATE CATALOG для запроса оператора создания внешнего catalog. Следующий пример запрашивает оператор создания MaxCompute catalog с именем `odps_catalog`:

```
SHOW CREATE CATALOG odps_catalog;
```

Удаление MaxCompute catalog

Вы можете использовать DROP CATALOG для удаления внешнего catalog.

Следующий пример удаляет MaxCompute catalog с именем `odps_catalog`:

```
DROP CATALOG odps_catalog;
```

Просмотр схемы MaxCompute таблицы

Вы можете использовать один из следующих синтаксисов для просмотра схемы MaxCompute таблицы:

- Просмотр схемы

```
DESC[RIBE] <catalog_name>.<database_name>.<table_name>;
```

- Просмотр схемы и расположения из оператора CREATE

```
SHOW CREATE TABLE <catalog_name>.<database_name>.<table_name>;
```

Запрос MaxCompute таблицы

1. Используйте SHOW DATABASES для просмотра баз данных в вашем MaxCompute cluster:

```
SHOW DATABASES FROM <catalog_name>;
```

2. Используйте SET CATALOG для переключения на целевой catalog в текущей сессии:

```
SET CATALOG <catalog_name>;
```

Затем используйте USE для указания активной базы данных в текущей сессии:

```
USE <db_name>;
```

Или вы можете использовать USE для непосредственного указания активной базы данных в целевом catalog:

```
USE <catalog_name>.<db_name>;
```

3. Используйте SELECT для запроса целевой таблицы в указанной базе данных:

```
SELECT count(*) FROM <table_name> LIMIT 10;
```

Загрузка данных из MaxCompute

Предположим, что в вашем Selena cluster есть OLAP таблица с именем `olap_tbl` и в вашем MaxCompute cluster есть таблица с именем `mc_table`. Вы можете преобразовать и загрузить данные из таблицы MaxCompute `mc_table` в таблицу Selena `olap_tbl` следующим образом:

```
INSERT INTO default_catalog.olap_db.olap_tbl SELECT * FROM mc_table;
```

Сопоставление типов данных

MaxCompute catalogs сопоставляют типы данных MaxCompute с типами данных Selena. Следующая таблица показывает сопоставление между типами данных MaxCompute и типами данных Selena.

Тип данных MaxCompute	Тип данных Selena
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
DECIMAL(p, s)	DECIMAL(p, s)
STRING	VARCHAR(1073741824)
VARCHAR(n)	VARCHAR(n)
CHAR(n)	CHAR(n)
JSON	VARCHAR(1073741824)
BINARY	VARBINARY
DATE	DATE
DATETIME	DATETIME
TIMESTAMP	DATETIME
ARRAY	ARRAY
MAP	MAP
STRUCT	STRUCT

© ООО «Платформа Данных Селена», 2026

> **ПРИМЕЧАНИЕ**
>
> Тип TIMESTAMP потеряет точность из-за преобразования типа в Selena.

Сбор статистики CBO

В текущей версии MaxCompute catalogs не могут автоматически собирать статистику CBO для таблиц MaxCompute, и, следовательно, оптимизатор может не иметь возможности генерировать оптимальные планы запросов. Таким образом, ручное сканирование статистики CBO для таблиц MaxCompute и импорт их в Selena может эффективно ускорить запросы.

Предположим, что в вашем MaxCompute cluster есть таблица MaxCompute с именем `mc_table`. Вы можете создать задачу ручного сбора для сбора статистики CBO с помощью ANALYZE TABLE:

```
ANALYZE TABLE mc_table;
```

Ручное обновление кэша метаданных

По умолчанию Selena кэширует метаданные MaxCompute для повышения производительности запросов. Поэтому после внесения изменений в схему или других обновлений в таблицу MaxCompute вы можете использовать REFRESH EXTERNAL TABLE для ручного обновления метаданных таблицы, тем самым гарантируя, что Selena может своевременно получить самые последние метаданные:

```
REFRESH EXTERNAL TABLE <table_name> [PARTITION ('partition_name', ...)]
```

5.7.12 Kudu catalog

> **EXPERIMENTAL FEATURE**
>
> Рекомендации по использованию экспериментальных функций

Selena поддерживает Kudu catalogs начиная с версии v1.5.2.

Kudu catalog — это тип внешнего catalog, который позволяет запрашивать данные из Apache Kudu без загрузки.

Кроме того, вы можете напрямую преобразовывать и загружать данные из Kudu с помощью INSERT INTO на основе Kudu catalogs.

Чтобы обеспечить успешное выполнение SQL-нагрузок на вашем Kudu cluster, ваш Selena cluster должен интегрироваться со следующими важными компонентами:

- Metastore, такой как ваша файловая система Kudu или Hive metastore

Примечания по использованию

Вы можете использовать Kudu catalogs только для запроса данных. Вы не можете использовать Kudu catalogs для удаления, уничтожения или вставки данных в ваш Kudu cluster.

Подготовка к интеграции

Перед созданием Kudu catalog убедитесь, что ваш Selena cluster может интегрироваться с системой хранения и metastore вашего Kudu cluster.

> **ПРИМЕЧАНИЕ**
>
> Если при отправке запроса возвращается ошибка, указывающая на неизвестный хост, вы должны добавить сопоставление между именами хостов и IP-адресами nodes вашего KUDU cluster в путь **/etc/hosts**.

Kerberos-аутентификация

Если Kerberos-аутентификация включена для вашего KUDU cluster или Hive metastore, настройте ваш Selena cluster следующим образом:

- Выполните команду `kinit -kt keytab_path principal` на каждом FE и каждом BE для получения Ticket Granting Ticket (TGT) от Key Distribution Center (KDC). Для выполнения этой команды вы должны иметь разрешения на доступ к вашему KUDU cluster и Hive metastore. Обратите внимание, что доступ к KDC с помощью этой команды зависит от времени. Поэтому вам нужно использовать cron для периодического выполнения этой команды.

- Добавьте `JAVA_OPTS="-Djava.security.krb5.conf=/etc/krb5.conf"` в файл **$FE_HOME/conf/fe.conf** каждого FE и в файл **$BE_HOME/conf/be.conf** каждого BE. В этом примере `/etc/krb5.conf` — это путь сохранения файла **krb5.conf**. Вы можете изменить путь в соответствии с вашими потребностями.

Создание Kudu catalog

Синтаксис

```
CREATE EXTERNAL CATALOG <catalog_name>
[COMMENT <comment>]
PROPERTIES
(
    "type" = "kudu",
    CatalogParams
)
```

Параметры

catalog_name

Название Kudu catalog. Соглашения об именовании следующие:

- Название может содержать буквы, цифры (0-9) и подчеркивания (_). Оно должно начинаться с буквы.
- Название чувствительно к регистру и не может превышать 1023 символа.

comment

Описание Kudu catalog. Этот параметр необязателен.

type

Тип вашего источника данных. Установите значение `kudu`.

CatalogParams

Набор параметров о том, как Selena обращается к метаданным вашего Kudu cluster.

Следующая таблица описывает параметр, который вам нужно настроить в `CatalogParams`.

Параметр	Обязательный	Описание
kudu.catalog.type	Да	Тип metastore, который вы используете для вашего Kudu cluster. Установите этот параметр на `kudu` или `hive`.
kudu.master	Нет	Указывает адрес Kudu Master, который по умолчанию равен `localhost:7051`.
hive.metastore.uris	Нет	URI вашего Hive metastore. Формат: `thrift://<metastore_IP_address>:<metastore_port>`. Если высокая доступность (HA) включена для вашего Hive metastore, вы можете указать несколько URI metastore и разделить их запятыми (`,`), например, `"thrift://<metastore_IP_address_1>:<metastore_port_1>,thrift://<metastore_IP_address_2>:<metastore_port_2>,thrift://<metastore_IP_address_3>:<metastore_port_3>"`.
kudu.schema-emulation.enabled	Нет	Опция для включения или отключения эмуляции `schema`. По умолчанию она отключена (false), что означает, что все таблицы принадлежат `schema` по умолчанию `default`.
kudu.schema-emulation.prefix	Нет	Префикс для эмуляции `schema` должен быть установлен только когда `kudu.schema-emulation.enabled` = `true`. Префикс по умолчанию используется пустая строка: `''`.

> **ПРИМЕЧАНИЕ**
>
> Если вы используете Hive metastore, вы должны добавить сопоставление между именами хостов и IP-адресами nodes вашего Hive metastore в путь `/etc/hosts` перед запросом данных Kudu. В противном случае Selena может не получить доступ к вашему Hive metastore при запуске запроса.

Примеры

- Следующие примеры создают Kudu catalog с именем `kudu_catalog`, чей тип metastore `kudu.catalog.type` установлен на `kudu` для запроса данных из вашего Kudu cluster.

```
CREATE EXTERNAL CATALOG kudu_catalog
PROPERTIES
(
    "type" = "kudu",
    "kudu.master" = "localhost:7051",
    "kudu.catalog.type" = "kudu",
    "kudu.schema-emulation.enabled" = "true",
    "kudu.schema-emulation.prefix" = "impala::"
);
```

- Следующие примеры создают Kudu catalog с именем `kudu_catalog`, чей тип metastore `kudu.catalog.type` установлен на `hive` для запроса данных из вашего Kudu cluster.

```
CREATE EXTERNAL CATALOG kudu_catalog
PROPERTIES
(
    "type" = "kudu",
    "kudu.master" = "localhost:7051",
    "kudu.catalog.type" = "hive",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083",
    "kudu.schema-emulation.enabled" = "true",
    "kudu.schema-emulation.prefix" = "impala::"
);
```

Просмотр Kudu catalogs

Вы можете использовать SHOW CATALOGS для запроса всех catalogs в текущем Selena cluster:

```
SHOW CATALOGS;
```

Вы также можете использовать SHOW CREATE CATALOG для запроса оператора создания внешнего catalog. Следующий пример запрашивает оператор создания Kudu catalog с именем `kudu_catalog`:

```
SHOW CREATE CATALOG kudu_catalog;
```

Удаление Kudu catalog

Вы можете использовать DROP CATALOG для удаления внешнего catalog.

Следующий пример удаляет Kudu catalog с именем `kudu_catalog`:

```
DROP Catalog kudu_catalog;
```

Просмотр схемы Kudu таблицы

Вы можете использовать один из следующих синтаксисов для просмотра схемы Kudu таблицы:

- Просмотр схемы

  ```
  DESC[RIBE] <catalog_name>.<database_name>.<table_name>;
  ```

- Просмотр схемы и расположения из оператора CREATE

  ```
  SHOW CREATE TABLE <catalog_name>.<database_name>.<table_name>;
  ```

Запрос Kudu таблицы

1. Используйте SHOW DATABASES для просмотра баз данных в вашем Kudu cluster:

   ```
   SHOW DATABASES FROM <catalog_name>;
   ```

2. Используйте SET CATALOG для переключения на целевой catalog в текущей сессии:

   ```
   SET CATALOG <catalog_name>;
   ```

   Затем используйте USE для указания активной базы данных в текущей сессии:

   ```
   USE <db_name>;
   ```

Или вы можете использовать USE для непосредственного указания активной базы данных в целевом catalog:

```
USE <catalog_name>.<db_name>;
```

3. Используйте SELECT для запроса целевой таблицы в указанной базе данных:

```
SELECT count(*) FROM <table_name> LIMIT 10;
```

Загрузка данных из Kudu

Предположим, у вас есть OLAP таблица с именем `olap_tbl`, вы можете преобразовывать и загружать данные следующим образом:

```
INSERT INTO default_catalog.olap_db.olap_tbl SELECT * FROM kudu_table;
```

5.7.13 Руководство по Iceberg Lakehouse

5.7.14 Apache Iceberg Lakehouse

Это руководство поможет вам начать работу с Apache Iceberg™ используя Selena™, включая примеры кода для демонстрации некоторых мощных возможностей.

Docker-Compose

Самый быстрый способ начать — использовать файл docker-compose, который использует образы `selena/fe-ubuntu` и `selena/be-ubuntu`, содержащие локальный cluster Selena с настроенным каталогом Iceberg. Для этого вам нужно установить Docker CLI.

После установки Docker сохраните приведенный ниже yaml в файл с именем docker-compose.yml:

```yaml
services:

  selena-fe:
    image: selena/fe-ubuntu:4.0-latest
    hostname: selena-fe
    container_name: selena-fe
    user: root
    command: |
      bash /opt/selena/fe/bin/start_fe.sh --host_type FQDN
    ports:
      - 8030:8030
      - 9020:9020
      - 9030:9030
    networks:
      iceberg_net:
    environment:
      - AWS_ACCESS_KEY_ID=admin
      - AWS_SECRET_ACCESS_KEY=password
      - AWS_REGION=us-east-1
    healthcheck:
      test: 'mysql -u root -h selena-fe -P 9030 -e "SHOW FRONTENDS\G" |grep
"Alive: true"'
      interval: 10s
      timeout: 5s
      retries: 3

  selena-be:
    image: selena/be-ubuntu:4.0-latest
    command:
      - /bin/bash
      - -c
      - |
        ulimit -n 65535;
        echo "# Enable data cache"  >> /opt/selena/be/conf/be.conf
        echo "block_cache_enable = true"  >> /opt/selena/be/conf/be.conf
        echo "block_cache_mem_size = 536870912" >>
/opt/selena/be/conf/be.conf
        echo "block_cache_disk_size = 1073741824" >>
/opt/selena/be/conf/be.conf
        sleep 15s
        mysql --connect-timeout 2 -h selena-fe -P 9030 -u root -e "ALTER
SYSTEM ADD BACKEND \"selena-be:9050\";"
        bash /opt/selena/be/bin/start_be.sh
    ports:
      - 8040:8040
```

```
    hostname: selena-be
    container_name: selena-be
    user: root
    depends_on:
      selena-fe:
        condition: service_healthy
    healthcheck:
      test: 'mysql -u root -h selena-fe -P 9030 -e "SHOW BACKENDS\G" |grep
"Alive: true"'
      interval: 10s
      timeout: 5s
      retries: 3
    networks:
      iceberg_net:
    environment:
      - HOST_TYPE=FQDN

  rest:
    image: apache/iceberg-rest-fixture
    container_name: iceberg-rest
    networks:
      iceberg_net:
        aliases:
          - iceberg-rest.minio
    ports:
      - 8181:8181
    environment:
      - AWS_ACCESS_KEY_ID=admin
      - AWS_SECRET_ACCESS_KEY=password
      - AWS_REGION=us-east-1
      - CATALOG_WAREHOUSE=s3://warehouse/
      - CATALOG_IO__IMPL=org.apache.iceberg.aws.s3.S3FileIO
      - CATALOG_S3_ENDPOINT=http://minio:9000

  minio:
    image: minio/minio:RELEASE.2024-10-29T16-01-48Z
    container_name: minio
    environment:
      - MINIO_ROOT_USER=admin
      - MINIO_ROOT_PASSWORD=password
      - MINIO_DOMAIN=minio
    networks:
      iceberg_net:
        aliases:
          - warehouse.minio
    ports:
```

```
      - 9001:9001
      - 9000:9000
    command: ["server", "/data", "--console-address", ":9001"]
  mc:
    depends_on:
      - minio
    image: minio/mc:RELEASE.2024-10-29T15-34-59Z
    container_name: mc
    networks:
      iceberg_net:
    environment:
      - AWS_ACCESS_KEY_ID=admin
      - AWS_SECRET_ACCESS_KEY=password
      - AWS_REGION=us-east-1
    entrypoint: >
      /bin/sh -c "
      until (/usr/bin/mc config host add minio http://minio:9000 admin
password) do echo '...waiting...' && sleep 1; done;
      /usr/bin/mc rm -r --force minio/warehouse;
      /usr/bin/mc mb minio/warehouse;
      /usr/bin/mc policy set public minio/warehouse;
      tail -f /dev/null
      "
networks:
  iceberg_net:
```

Затем запустите контейнеры docker с помощью этой команды:

```
docker compose up --detach --wait --wait-timeout 400
```

После этого вы можете выполнить любую из следующих команд, чтобы начать сессию Selena.

```
docker exec -it selena-fe \
mysql -P 9030 -h 127.0.0.1 -u root --prompt="Selena > "
```

Добавление и использование каталога

```
CREATE EXTERNAL CATALOG 'demo'
COMMENT "External catalog to Apache Iceberg on MinIO"
PROPERTIES
(
  "type"="iceberg",
  "iceberg.catalog.type"="rest",
  "iceberg.catalog.uri"="http://iceberg-rest:8181",
  "iceberg.catalog.warehouse"="warehouse",
  "aws.s3.access_key"="admin",
  "aws.s3.secret_key"="password",
  "aws.s3.endpoint"="http://minio:9000",
  "aws.s3.enable_path_style_access"="true"
);
```

```
SHOW CATALOGS\G
```

```
*************************** 1. row ***************************
Catalog: default_catalog
   Type: Internal
Comment: An internal catalog contains this cluster's self-managed tables.
*************************** 2. row ***************************
Catalog: demo
   Type: Iceberg
Comment: External catalog to Apache Iceberg on MinIO
2 rows in set (0.00 sec)
```

```
SET CATALOG demo;
```

Создание и использование базы данных

```
CREATE DATABASE nyc;
```

```
USE nyc;
```

Создание таблицы

```
CREATE TABLE demo.nyc.taxis
(
    trip_id            bigint,
    trip_distance      float,
    fare_amount double,
    store_and_fwd_flag string,
    vendor_id          bigint
) PARTITION BY (vendor_id);
```

Запись данных в таблицу

```
INSERT INTO demo.nyc.taxis
VALUES (1000371, 1.8, 15.32, 'N', 1),
       (1000372, 2.5, 22.15, 'N', 2),
       (1000373, 0.9, 9.01, 'N', 2),
       (1000374, 8.4, 42.13, 'Y', 1);
```

Чтение данных из таблицы

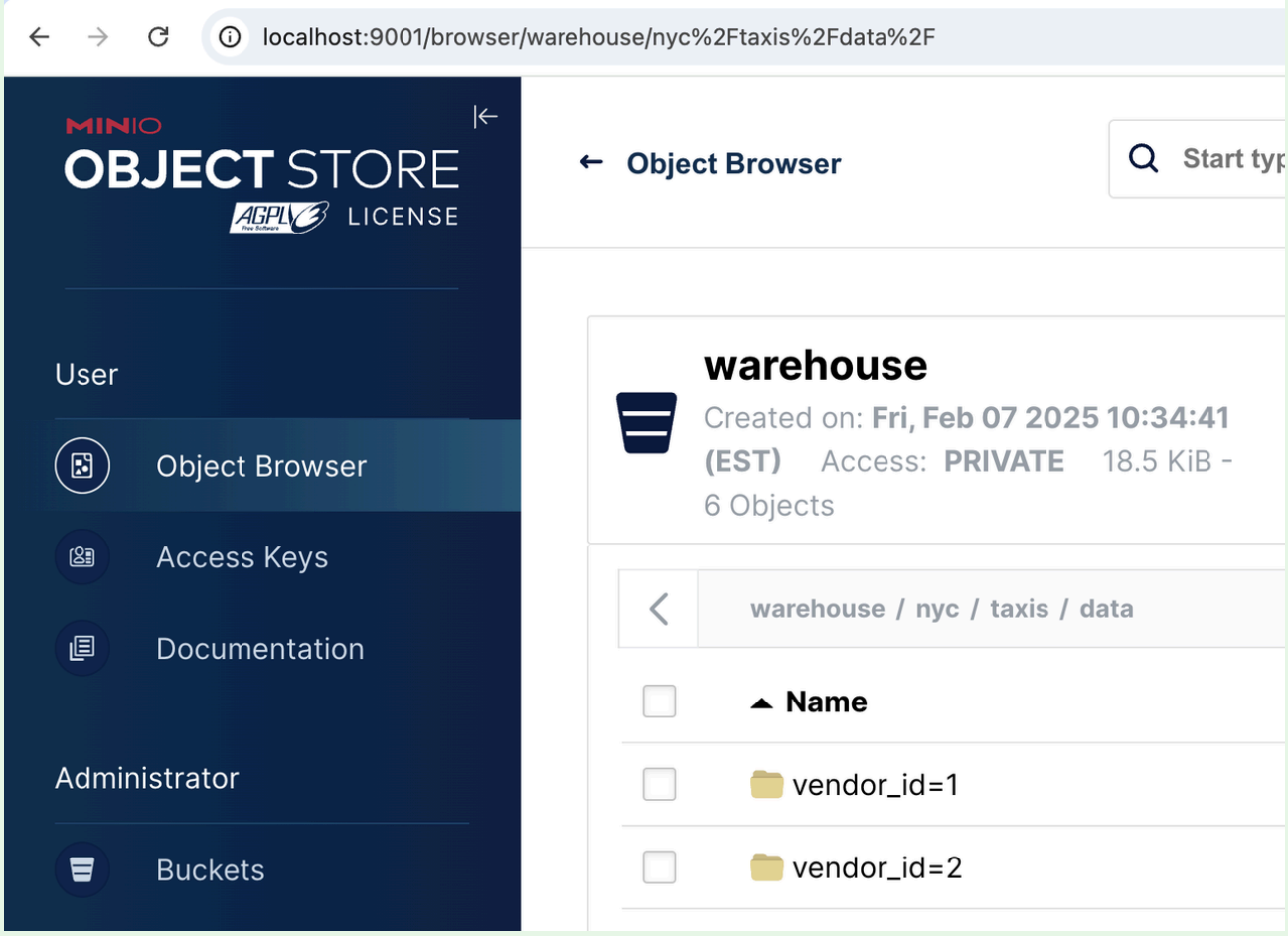
```
SELECT *
FROM demo.nyc.taxis;
```

Проверка хранения данных в объектном хранилище

Когда вы добавили и использовали внешний каталог, Selena начала использовать MinIO в качестве объектного хранилища для таблицы `demo.nyc.taxis`. Если вы перейдете по адресу http://localhost:9001 и затем перейдете через меню Object Browser к `warehouse/nyc/taxis/`, вы можете подтвердить, что Selena использует MinIO для хранения.

ПОДСКАЗКА

Имя пользователя и пароль для MinIO находятся в файле docker-compose.yml. Вам будет предложено изменить пароль на что-то более надежное, просто проигнорируйте этот совет для данного руководства.



Следующие шаги

Добавление Iceberg в Selena

Если у вас уже есть среда Selena версии 3.2.0 или новее, она уже включает Iceberg 1.6.0. Дополнительные загрузки или jar-файлы не требуются.

Узнать больше

Теперь, когда вы запустили Iceberg и Selena, ознакомьтесь с документацией Selena-Iceberg, чтобы узнать больше!

5.7.15 Запрос внешних данных

Этот раздел описывает процесс запроса данных из внешних источников данных с использованием внешних catalog.

Предварительные требования

Внешние catalog создаются на основе внешних источников данных. Информацию о поддерживаемых типах внешних catalog см. в разделе Catalog.

Процедура

1. Подключитесь к вашему Selena cluster.

   - Если вы используете MySQL-клиент для подключения к Selena cluster, вы по умолчанию переходите в `default_catalog` после подключения.
   - Если вы используете JDBC для подключения к Selena cluster, вы можете сразу перейти в целевую базу данных в default catalog, указав `default_catalog.db_name` при подключении.
2. (Опционально) Выполните следующий оператор для просмотра всех catalog и поиска созданного вами внешнего catalog. См. SHOW CATALOGS для проверки вывода этого оператора.

   ```
   SHOW CATALOGS;
   ```

3. (Опционально) Выполните следующий оператор для просмотра всех баз данных во внешнем catalog. См. SHOW DATABASES для проверки вывода этого оператора.

   ```
   SHOW DATABASES FROM catalog_name;
   ```

4. (Опционально) Выполните следующий оператор для перехода в целевую базу данных во внешнем catalog.

   ```
   USE catalog_name.db_name;
   ```

5. Выполните запрос к внешним данным. Дополнительные способы использования оператора SELECT см. в разделе SELECT.

```
SELECT * FROM table_name;
```

Если вы не указали внешний catalog и базу данных на предыдущих шагах, вы можете указать их непосредственно в запросе select.

```
SELECT * FROM catalog_name.db_name.table_name;
```

Примеры

Если вы уже создали Hive catalog с именем `hive1` и хотите использовать `hive1` для запроса данных из `hive_db.hive_table` cluster Apache Hive™, вы можете выполнить одну из следующих операций:

```
USE hive1.hive_db;
SELECT * FROM hive_table limit 1;
```

Или

```
SELECT * FROM hive1.hive_db.hive_table limit 1;
```

См. также

Для запроса данных из вашего Selena cluster см. раздел Default catalog.

5.8 Кэш данных

5.8.1 Data Cache

Понимание основных принципов Data Cache и способов ускорения запросов данных с помощью Data Cache.

Data Cache используется для кэширования данных нативных таблиц и внешних таблиц. Эта функция включена по умолчанию начиная с версии v1.5.2. Начиная с версии v2.0.0, кэш в памяти и дисковый кэш были объединены в единую систему Data Cache для удобства управления.

Data Cache состоит из двух компонентов: Page Cache (кэш в памяти) и Block Cache (дисковый кэш).

Принципы работы Page Cache

Как кэш в памяти, Page Cache отвечает за хранение страниц данных нативных и внешних таблиц после декомпрессии. Размер этих страниц не фиксирован. В настоящее время Page Cache поддерживает кэширование следующих типов данных:

- Страницы данных и индексные страницы нативных таблиц
- Информация footer файлов данных внешних таблиц
- Частично декомпрессированные страницы данных внешних таблиц

Page Cache в настоящее время использует стратегию LRU (Least Recently Used) для вытеснения данных.

Принципы работы Block Cache

Block Cache — это дисковый кэш, основная функция которого заключается в кэшировании файлов данных (из внешних таблиц, а также из cloud-native таблиц в shared-data cluster) на локальные диски. Это снижает задержку доступа к удалённым данным и повышает эффективность запросов. Размер каждого блока данных фиксирован.

Предпосылки и ценность

В сценариях аналитики Data Lake и cloud-native таблиц Selena, выступая в роли OLAP-движка запросов, должна сканировать файлы данных, хранящиеся в HDFS или объектном хранилище (далее — «удалённая система хранения»). Этот процесс сталкивается с двумя основными узкими местами производительности:

- Чем больше файлов нужно прочитать запросу, тем больше накладные расходы на удалённый I/O.
- В сценариях ad-hoc запросов частый доступ к одним и тем же данным приводит к избыточному потреблению удалённого I/O.

Для решения этих проблем в версии v1.5.2 была введена функция Block Cache. Она разделяет исходные данные из удалённой системы хранения на несколько блоков по определённой стратегии и кэширует эти блоки на локальных дисках узлов BE или CN. За счёт избежания повторного получения удалённых данных значительно повышается производительность запросов к горячим данным.

Сценарии использования

- Запросы к данным из удалённых систем хранения с использованием внешних catalog (кроме JDBC Catalog).
- Запросы к cloud-native таблицам в shared-data cluster.

Основной механизм

Разбиение данных и единица кэширования

Когда система кэширует удалённые файлы, она разбивает исходные файлы на блоки равного размера в соответствии с настроенной стратегией. Блок — это минимальная единица кэширования, и его размер настраивается.

Пример:

Если размер блока настроен как 1 МБ, при запросе файла Parquet размером 128 МБ на Amazon S3 файл будет разбит на 128 последовательных блоков (то есть `[0, 1 МБ)`, `[1 МБ, 2 МБ)`, ..., `[127 МБ, 128 МБ)`).

Каждому блоку присваивается глобально уникальный идентификатор кэша (cache key), который состоит из следующих трёх частей:

```
hash(filename) + fileModificationTime + blockId
```

Компонент	Описание
filename	Имя файла данных.
fileModificationTime	Время последнего изменения файла данных.
blockId	ID, присвоенный каждому блоку при разбиении файла данных. Этот ID уникален в пределах одного файла, но не глобально уникален.

Попадание в кэш и процесс чтения

Предположим, запрос попадает в блок в диапазоне `[1 МБ, 2 МБ)`, Block Cache действует следующим образом:

1. Система сначала проверяет, существует ли блок в Block Cache локального узла BE (путём сопоставления cache key).
2. Если найден (попадание в кэш), блок читается напрямую с локального диска.

3. Если не найден (промах кэша), блок извлекается из удалённого хранилища и синхронизируется в Block Cache локального узла BE для повторного использования в последующих запросах.

Среда хранения кэша

Block Cache использует локальные диски узлов BE или CN в качестве среды хранения, и эффект ускорения кэша напрямую связан с производительностью диска:

- Рекомендуется использовать высокопроизводительные локальные диски (например, NVMe-диски) для минимизации задержки чтения/записи кэша.
- Если производительность диска неоптимальна, вы можете увеличить количество дисков для достижения балансировки нагрузки и снижения нагрузки I/O на отдельные диски.

Политики замены кэша

Block Cache поддерживает две стратегии кэширования и вытеснения данных: LRU и SLRU (Segmented LRU).

LRU

Стратегия LRU основана на принципе «Наименее недавно использованный» — вытесняются блоки, к которым не обращались дольше всего. Она проста в реализации и подходит для сценариев со стабильными паттернами доступа.

SLRU

Стратегия SLRU делит пространство кэша на сегмент вытеснения и сегмент защиты, оба следуют правилам LRU:

1. Данные попадают в сегмент вытеснения при первом доступе.
2. Данные в сегменте вытеснения перемещаются в сегмент защиты при повторном доступе.
3. Данные, вытесненные из сегмента защиты, возвращаются в сегмент вытеснения, в то время как данные, вытесненные из сегмента вытеснения, полностью удаляются из кэша.

Стратегия SLRU может эффективно противостоять внезапному разреженному трафику, предотвращая прямое вытеснение горячих данных в сегменте защиты «временными

данными, к которым обращались только один раз». Она обеспечивает лучшую стабильность, чем LRU.

Включение и настройка Data Cache

Data Cache включён по умолчанию, управляется параметром конфигурации BE `datacache_enable` (по умолчанию: `true`). Page Cache и Block Cache, как два независимых компонента, также включены по умолчанию. Установка `datacache_enable` в `false` отключит Data Cache целиком, то есть и Page Cache, и Block Cache.

Вы также можете активировать или деактивировать Page Cache и Block Cache по отдельности, используя различные параметры конфигурации BE.

- Page Cache (включён по умолчанию) управляется параметром `disable_storage_page_cache` (по умолчанию: `false`).
- Block Cache (включён по умолчанию) управляется параметром `block_cache_enable` (по умолчанию: `true`).

Вы можете дополнительно использовать следующие конфигурации BE для установки максимальных лимитов использования памяти и диска для Data Cache, предотвращая чрезмерное потребление ресурсов:

- `datacache_mem_size`: Устанавливает максимальный лимит использования памяти для Data Cache (используется для хранения данных в Page Cache).
- `datacache_disk_size`: Устанавливает максимальный лимит использования диска для Data Cache (используется для хранения данных в Block Cache).

Заполнение Block Cache

Правила заполнения

Начиная с версии v1.5.2, для повышения коэффициента попадания в Block Cache система заполняет Block Cache в соответствии со следующими правилами:

- Кэш не будет заполняться для операторов, отличных от `SELECT`, например, `ANALYZE TABLE` и `INSERT INTO SELECT`.
- Запросы, сканирующие все partition таблицы, не будут заполнять кэш. Однако, если таблица имеет только один partition, заполнение выполняется по умолчанию.
- Запросы, сканирующие все столбцы таблицы, не будут заполнять кэш. Однако, если таблица имеет только один столбец, заполнение выполняется по умолчанию.

- Кэш не будет заполняться для таблиц, которые не являются Hive, Paimon, Delta Lake, Hudi или Iceberg.

Просмотр поведения заполнения кэша

Вы можете просмотреть поведение заполнения для конкретного запроса с помощью команды `EXPLAIN VERBOSE`.

Пример:

```
mysql> EXPLAIN VERBOSE SELECT col1 FROM hudi_table;
...
|   0:HudiScanNode                        |
|      TABLE: hudi_table                  |
|      partitions=3/3                     |
|      cardinality=9084                   |
|      avgRowSize=2.0                     |
|      dataCacheOptions={populate: false} |
|      cardinality: 9084                  |
+-----------------------------------------+
```

`dataCacheOptions={populate: false}` указывает, что кэш не будет заполняться, поскольку запрос будет сканировать все partition.

Вы также можете точно настроить поведение заполнения Block Cache через сессионную переменную `populate_datacache_mode`.

Режим заполнения

Block Cache поддерживает два режима: синхронное заполнение и асинхронное заполнение. Вы можете выбирать между ними в зависимости от бизнес-требований к «производительности первого запроса» и «эффективности кэша».

Синхронное заполнение

- Основная логика: Когда удалённые данные читаются для первого запроса, данные немедленно кэшируются локально. Последующие запросы могут напрямую использовать кэш.
- Преимущества: Высокая эффективность кэша, поскольку кэширование данных завершается за один запрос.

- Недостатки: Операции кэширования выполняются синхронно с операциями чтения, что может увеличить задержку для первого запроса.

Асинхронное заполнение

- Основная логика: Для первого запроса чтение данных имеет приоритет и выполняется первым. Запись в кэш выполняется асинхронно в фоновом режиме, не блокируя текущий процесс запроса.
- Преимущества: Не влияет на производительность первого запроса и предотвращает задержку операций чтения из-за кэширования.
- Недостатки: Более низкая эффективность кэша, поскольку один запрос может не полностью кэшировать все данные, к которым обращались. Требуется несколько запросов для постепенного улучшения покрытия кэша.

Начиная с версии v1.5.2, асинхронное заполнение кэша включено по умолчанию. Вы можете изменить режим заполнения, установив сессионную переменную `enable_datacache_async_populate_mode`.

Персистентность

Кэшированные данные на дисках по умолчанию являются персистентными, и эти данные могут быть повторно использованы после перезапуска BE или CN.

Проверка попадания запроса в Data Cache

Вы можете проверить, попал ли запрос в Data Cache, анализируя следующие метрики в профиле запроса:

- `DataCacheReadBytes`: размер данных, которые система читает напрямую из памяти и с дисков.
- `DataCacheWriteBytes`: размер данных, загруженных из удалённой системы хранения в память и на диски.
- `BytesRead`: общий объём прочитанных данных, включая данные, которые система читает из удалённой системы хранения, а также из памяти и с дисков.

Пример 1: В этом примере система читает большой объём данных (7,65 ГБ) из удалённой системы хранения и лишь немного данных (518,73 МБ) с дисков. Это означает, что попадание в Block Cache было минимальным.

```
- Table: lineorder
- DataCacheReadBytes: 518.73 MB
  - __MAX_OF_DataCacheReadBytes: 4.73 MB
  - __MIN_OF_DataCacheReadBytes: 16.00 KB
- DataCacheReadCounter: 684
  - __MAX_OF_DataCacheReadCounter: 4
  - __MIN_OF_DataCacheReadCounter: 0
- DataCacheReadTimer: 737.357us
- DataCacheWriteBytes: 7.65 GB
  - __MAX_OF_DataCacheWriteBytes: 64.39 MB
  - __MIN_OF_DataCacheWriteBytes: 0.00
- DataCacheWriteCounter: 7.887K (7887)
  - __MAX_OF_DataCacheWriteCounter: 65
  - __MIN_OF_DataCacheWriteCounter: 0
- DataCacheWriteTimer: 23.467ms
  - __MAX_OF_DataCacheWriteTimer: 62.280ms
  - __MIN_OF_DataCacheWriteTimer: 0ns
- BufferUnplugCount: 15
  - __MAX_OF_BufferUnplugCount: 2
  - __MIN_OF_BufferUnplugCount: 0
- BytesRead: 7.65 GB
  - __MAX_OF_BytesRead: 64.39 MB
  - __MIN_OF_BytesRead: 0.00
```

Пример 2: В этом примере система читает большой объём данных (46,08 ГБ) из Data Cache и не читает данные из удалённой системы хранения, что означает, что данные читаются только из Block Cache.

```
Table: lineitem
- DataCacheReadBytes: 46.08 GB
 - __MAX_OF_DataCacheReadBytes: 194.99 MB
 - __MIN_OF_DataCacheReadBytes: 81.25 MB
- DataCacheReadCounter: 72.237K (72237)
 - __MAX_OF_DataCacheReadCounter: 299
 - __MIN_OF_DataCacheReadCounter: 118
- DataCacheReadTimer: 856.481ms
 - __MAX_OF_DataCacheReadTimer: 1s547ms
 - __MIN_OF_DataCacheReadTimer: 261.824ms
- DataCacheWriteBytes: 0.00
- DataCacheWriteCounter: 0
- DataCacheWriteTimer: 0ns
- BufferUnplugCount: 1.231K (1231)
 - __MAX_OF_BufferUnplugCount: 81
 - __MIN_OF_BufferUnplugCount: 35
- BytesRead: 46.08 GB
 - __MAX_OF_BytesRead: 194.99 MB
 - __MIN_OF_BytesRead: 81.25 MB
```

I/O Adaptor

Для предотвращения значительной хвостовой задержки при доступе к диску из-за высокой нагрузки I/O на диск кэша, что может привести к негативной оптимизации системы кэширования, Data Cache предоставляет функцию I/O Adaptor. Эта функция направляет некоторые запросы кэша в удалённое хранилище при высокой нагрузке на диск, используя как локальный кэш, так и удалённое хранилище для повышения пропускной способности I/O. Эта функция включена по умолчанию.

Вы можете включить I/O Adaptor, установив следующую системную переменную:

```
SET GLOBAL enable_datacache_io_adaptor=true;
```

Динамическое масштабирование

Data Cache поддерживает ручную настройку ёмкости кэша без перезапуска процесса BE, а также поддерживает автоматическую настройку ёмкости кэша.

Ручное масштабирование

Вы можете изменить лимит памяти или дисковую ёмкость Data Cache, динамически настраивая параметры конфигурации BE.

Примеры:

```
-- Настройка лимита памяти Data Cache для конкретного экземпляра BE.
UPDATE be_configs SET VALUE="10G" WHERE NAME="datacache_mem_size" and
BE_ID=10005;

-- Настройка лимита соотношения памяти Data Cache для всех экземпляров BE.
UPDATE be_configs SET VALUE="10%" WHERE NAME="datacache_mem_size";

-- Настройка дискового лимита Data Cache для всех экземпляров BE.
UPDATE be_configs SET VALUE="2T" WHERE NAME="datacache_disk_size";
```

ПРИМЕЧАНИЕ

- Будьте осторожны при настройке ёмкостей таким способом. Убедитесь, что вы не пропустили предложение WHERE, чтобы избежать изменения несвязанных параметров конфигурации.
- Настройки ёмкости кэша, сделанные таким образом, не будут сохранены и будут потеряны после перезапуска процесса BE или CN. Поэтому вы можете сначала динамически настроить параметры, как описано выше, а затем вручную изменить файл конфигурации BE или CN, чтобы гарантировать, что изменения вступят в силу после следующего перезапуска.

Автоматическое масштабирование

Selena в настоящее время поддерживает автоматическое масштабирование дисковой ёмкости. Если вы не указываете путь к диску кэша и лимит ёмкости в конфигурации BE, автоматическое масштабирование включается по умолчанию.

Вы также можете включить автоматическое масштабирование, добавив следующий параметр конфигурации в файл конфигурации и перезапустив процесс BE или CN:

```
datacache_auto_adjust_enable=true
```

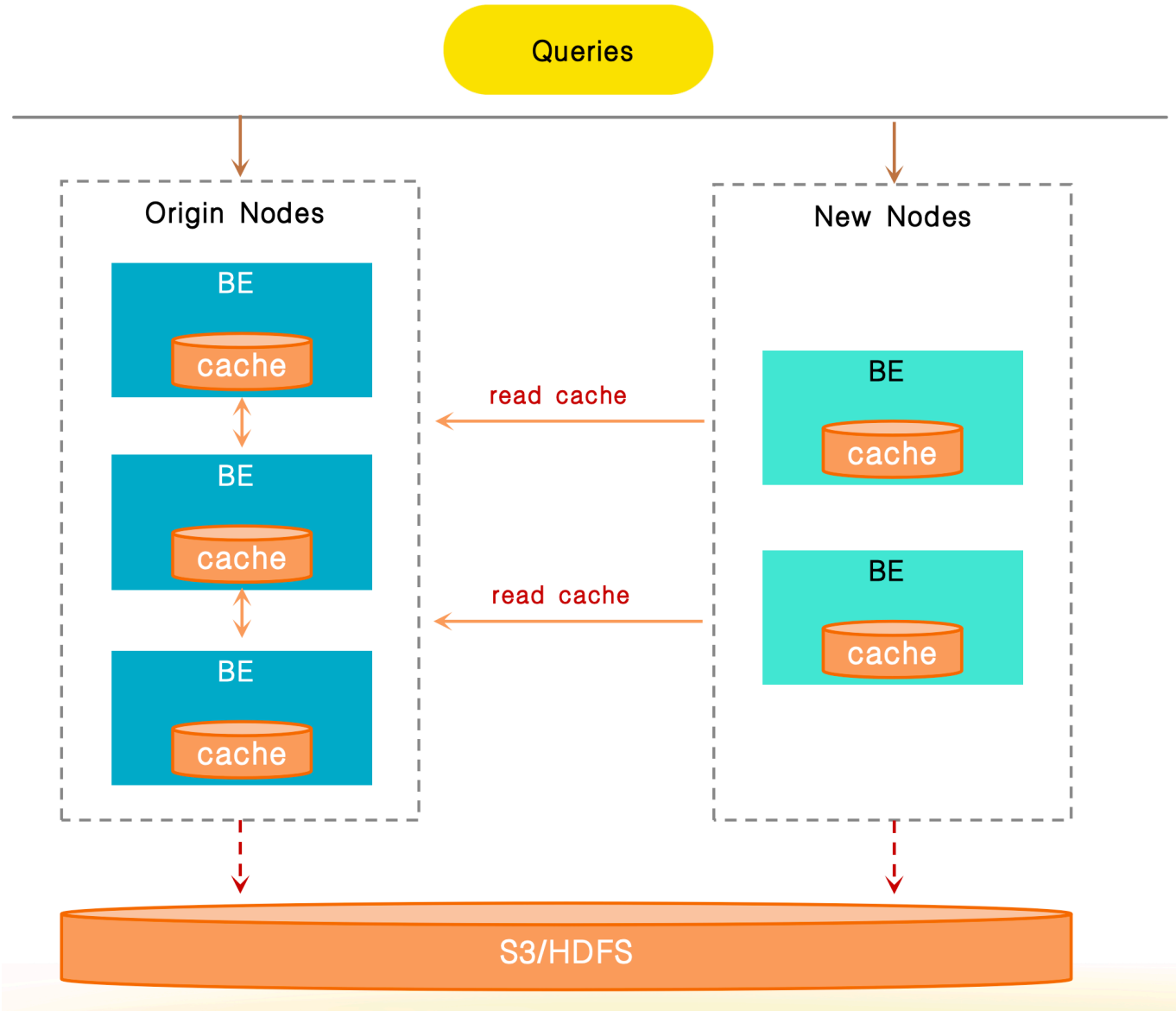
После включения автоматического масштабирования:

- Когда использование диска превышает порог, указанный в конфигурации BE `disk_high_level` (значение по умолчанию `90`, то есть 90% дискового пространства), система автоматически вытесняет данные кэша для освобождения дискового пространства.
- Когда использование диска постоянно ниже порога, указанного в конфигурации BE `disk_low_level` (значение по умолчанию `60`, то есть 60% дискового пространства), и текущее дисковое пространство, используемое Data Cache, заполнено, система автоматически расширяет ёмкость кэша.
- При автоматическом масштабировании ёмкости кэша система будет стремиться настроить ёмкость кэша до уровня, указанного в конфигурации BE `disk_safe_level` (значение по умолчанию `80`, то есть 80% дискового пространства).

Совместное использование кэша (Cache Sharing)

Поскольку Data Cache зависит от локального диска узла BE, при масштабировании cluster изменения в маршрутизации данных могут вызвать промахи кэша, что легко может привести к значительному снижению производительности во время эластичного масштабирования.

Cache Sharing используется для поддержки доступа к данным кэша между узлами через сеть. Во время масштабирования cluster, если происходит промах локального кэша, система сначала пытается получить данные кэша с других узлов в пределах того же cluster. Только если все кэши промахнулись, система повторно получит данные из удалённого хранилища. Эта функция эффективно снижает колебания производительности, вызванные инвалидацией кэша во время эластичного масштабирования, и обеспечивает стабильную производительность запросов.



Вы можете включить функцию Cache Sharing, настроив следующие два параметра:

- Установите параметр конфигурации FE `enable_trace_historical_node` в `true`.
- Установите системную переменную `enable_datacache_sharing` в `true`.

Кроме того, вы можете проверить следующие метрики в профиле запроса для мониторинга Cache Sharing:

- `DataCacheReadPeerCounter`: Количество чтений с других узлов.
- `DataCacheReadPeerBytes`: Количество байтов, прочитанных с других узлов.
- `DataCacheReadPeerTimer`: Время, затраченное на доступ к данным кэша с других узлов.

Конфигурации и переменные

Вы можете настроить Data Cache, используя следующие системные переменные и параметры.

Системные переменные

- populate_datacache_mode
- enable_datacache_io_adaptor
- enable_file_metacache
- enable_datacache_async_populate_mode
- enable_datacache_sharing

Конфигурации FE

- enable_trace_historical_node

Конфигурации BE

- datacache_enable
- datacache_mem_size
- datacache_disk_size
- datacache_auto_adjust_enable
- datacache_disk_high_level
- datacache_disk_safe_level
- datacache_disk_low_level
- datacache_disk_adjust_interval_seconds
- datacache_disk_idle_seconds_for_expansion
- datacache_min_disk_quota_for_adjustment
- datacache_eviction_policy
- datacache_inline_item_count_limit

5.8.2 Прогрев Block cache

Некоторые сценарии аналитики озер данных и shared-data cluster предъявляют высокие требования к производительности запросов, такие как BI-отчеты и тестирование производительности в рамках proof of concept (PoC). Загрузка удаленных данных в локальный block cache позволяет избежать необходимости многократно извлекать одни и те же данные, значительно ускоряя выполнение запросов и минимизируя использование ресурсов.

Selena v1.5.2 представляет функцию Block Cache Warmup, которая является улучшением Block Cache. Block Cache — это процесс пассивного заполнения кеша, при котором данные записываются в кеш во время выполнения запросов. Block Cache Warmup, однако, представляет собой активный процесс заполнения кеша. Он заранее проактивно извлекает необходимые данные из удаленного хранилища.

Сценарии использования

- Диск, используемый для block cache, имеет емкость хранения значительно большую, чем объем данных для прогрева. Если емкость диска меньше объема данных для прогрева, ожидаемый эффект прогрева не может быть достигнут. Например, если необходимо прогреть 100 ГБ данных, но диск имеет только 50 ГБ пространства, то только 50 ГБ данных можно загрузить в кеш, и ранее загруженные 50 ГБ данных будут заменены 50 ГБ данных, которые загружаются позже.
- Доступ к данным на диске, используемом для block cache, относительно стабилен. Если происходит всплеск объема доступа, ожидаемый эффект прогрева не может быть достигнут. Например, если необходимо прогреть 100 ГБ данных и диск имеет 200 ГБ пространства, то первое условие выполнено. Однако, если большой объем новых данных (150 ГБ) записывается в кеш во время процесса прогрева, или если неожиданный большой холодный запрос требует загрузки 150 ГБ данных в кеш, это может привести к вытеснению прогретых данных.

Как это работает

Selena предоставляет синтаксис CACHE SELECT для реализации Block Cache Warmup. Перед использованием CACHE SELECT убедитесь, что функция Block Cache включена.

Синтаксис CACHE SELECT:

```
CACHE SELECT <column_name> [, ...]
FROM [<catalog_name>.][<db_name>.]<table_name> [WHERE <boolean_expression>]
[PROPERTIES("verbose"="true")]
```

Параметры:

- `column_name`: Столбцы для извлечения. Вы можете использовать `*` для извлечения всех столбцов во внешней таблице.

- `catalog_name`: Имя catalog, по умолчанию DEFAULT_CATALOG. Если вы переключились на catalog с помощью SET CATALOG, его можно не указывать.
- `db_name`: Имя базы данных. Если вы переключились на эту базу данных, его можно не указывать.
- `table_name`: Имя таблицы, из которой извлекаются данные.
- `boolean_expression`: Условие фильтрации.
- `PROPERTIES`: В настоящее время поддерживается только свойство `verbose`. Оно используется для возврата подробных метрик прогрева.

CACHE SELECT является синхронным процессом и может прогревать только одну таблицу за раз. После успешного выполнения он возвращает метрики, связанные с прогревом.

Прогрев всех данных во внешней таблице

Следующий пример загружает все данные из внешней таблицы `lineitem`:

```
mysql> cache select * from hive_catalog.test_db.lineitem;
+-----------------+------------------+----------------------+--------------
-----+
| READ_CACHE_SIZE | WRITE_CACHE_SIZE | AVG_WRITE_CACHE_TIME |
TOTAL_CACHE_USAGE |
+-----------------+------------------+----------------------+--------------
-----+
| 48.2MB          | 3.7GB            | 59ms                 | 96.83%
|
+-----------------+------------------+----------------------+--------------
-----+
1 row in set (19.56 sec)
```

Возвращаемые поля:

- `READ_CACHE_SIZE`: Общий размер данных, прочитанных из block cache всеми узлами.
- `WRITE_CACHE_SIZE`: Общий размер данных, записанных в block cache всеми узлами.
- `AVG_WRITE_CACHE_TIME`: Среднее время, затраченное каждым узлом на запись данных в block cache.
- `TOTAL_CACHE_USAGE`: Использование дискового пространства block cache всего cluster после завершения этой задачи прогрева. Эта метрика может использоваться для оценки того, имеет ли block cache достаточно места.

Прогрев указанных столбцов с условиями фильтрации

Вы можете указать столбцы и предикаты для достижения детального прогрева, что помогает уменьшить объем данных для прогрева, снижая дисковый I/O и потребление CPU.

```
mysql> cache select l_orderkey from hive_catalog.test_db.lineitem where
l_shipdate='1994-10-28';
+-----------------+------------------+----------------------+--------------
-----+
| READ_CACHE_SIZE | WRITE_CACHE_SIZE | AVG_WRITE_CACHE_TIME |
TOTAL_CACHE_USAGE |
+-----------------+------------------+----------------------+--------------
-----+
| 957MB           | 713.5MB          | 3.6ms                | 97.33%
|
+-----------------+------------------+----------------------+--------------
-----+
1 row in set (9.07 sec)
```

Следующий пример предварительно извлекает определенный столбец из облачно-нативной таблицы `lineorder` в shared-data cluster:

```
mysql> cache select lo_orderkey from ssb.lineorder;
+-----------------+------------------+----------------------+--------------
-----+
| READ_CACHE_SIZE | WRITE_CACHE_SIZE | AVG_WRITE_CACHE_TIME |
TOTAL_CACHE_USAGE |
+-----------------+------------------+----------------------+--------------
-----+
| 118MB           | 558.9MB          | 200.6ms              | 4.66%
|
+-----------------+------------------+----------------------+--------------
-----+
1 row in set (29.88 sec)
```

Прогрев в подробном режиме

По умолчанию метрики, возвращаемые `CACHE SELECT`, являются метриками, объединенными на нескольких BE. Вы можете добавить `PROPERTIES("verbose"="true")` в конец CACHE SELECT, чтобы получить подробные метрики каждого BE.

```
mysql> cache select * from hive_catalog.test_db.lineitem
properties("verbose"="true");
+---------------+-----------------+---------------------+------------------
+----------------------+-------------------+
| IP            | READ_CACHE_SIZE | AVG_READ_CACHE_TIME | WRITE_CACHE_SIZE
| AVG_WRITE_CACHE_TIME | TOTAL_CACHE_USAGE |
+---------------+-----------------+---------------------+------------------
+----------------------+-------------------+
| 172.26.80.233 | 376MB           | 127.8micros         | 0B
| 0s                   | 3.85%             |
| 172.26.80.231 | 272.5MB         | 121.8micros         | 20.7MB
| 146.5micros          | 3.91%             |
| 172.26.80.232 | 355.5MB         | 147.7micros         | 0B
| 0s                   | 3.91%             |
+---------------+-----------------+---------------------+------------------
+----------------------+-------------------+
3 rows in set (0.54 sec)
```

В подробном режиме возвращается дополнительная метрика:

- `AVG_READ_CACHE_TIME`: среднее время для каждого узла на чтение данных при попадании в block cache.

Периодическое планирование задач CACHE SELECT

Вы можете использовать CACHE SELECT вместе с SUBMIT TASK для достижения периодического прогрева. Например, следующий случай прогревает таблицу `lineitem` каждые 5 минут:

```
mysql> submit task always_cache schedule every(interval 5 minute) as cache
select l_orderkey
from hive_catalog.test_db.lineitem
where l_shipdate='1994-10-28';
+--------------+-----------+
| TaskName     | Status    |
+--------------+-----------+
| always_cache | SUBMITTED |
+--------------+-----------+
1 row in set (0.03 sec)
```

Управление задачами CACHE SELECT

Просмотр созданных задач

```
mysql> select * from default_catalog.information_schema.tasks;
+--------------+---------------------+-------------------------------------
----------------+---------------+-----------------------------+-----------
-------------------------------------------------------------+----------------
-----+------------+
| TASK_NAME    | CREATE_TIME         | SCHEDULE
| CATALOG       | DATABASE                     | DEFINITION
| EXPIRE_TIME         | PROPERTIES |
+--------------+---------------------+-------------------------------------
----------------+---------------+-----------------------------+-----------
-------------------------------------------------------------+----------------
-----+------------+
| always_cache | 2024-04-11 16:01:00 | PERIODICAL START(2024-04-11T16:01)
EVERY(5 MINUTES) | emr_hive_test | zz_tpch_sf1000_hive_orc_zlib | cache
select l_orderkey from lineitem where l_shipdate='1994-10-28' | NULL
|            |
+--------------+---------------------+-------------------------------------
----------------+---------------+-----------------------------+-----------
-------------------------------------------------------------+----------------
-----+------------+
1 row in set (0.21 sec)
```

Просмотр истории выполнения задач

```
mysql> select * from default_catalog.information_schema.task_runs;
+--------------------------------------+--------------+--------------------
-+---------------------+---------+---------------+------------------------
-----+------------------------------------------------------------------------
+---------------------+------------+---------------+----------+------------
--------------------------------------------------------------------------------
---------------------------------+------------+
| QUERY_ID                             | TASK_NAME    | CREATE_TIME
| FINISH_TIME         | STATE   | CATALOG       | DATABASE
| DEFINITION                                                                 |
EXPIRE_TIME         | ERROR_CODE | ERROR_MESSAGE | PROGRESS | EXTRA_MESSAGE
| PROPERTIES |
+--------------------------------------+--------------+--------------------
-+---------------------+---------+---------------+------------------------
-----+------------------------------------------------------------------------
+---------------------+------------+---------------+----------+------------
--------------------------------------------------------------------------------
---------------------------------+------------+
| 55b30204-f7da-11ee-b03e-7ea526d0b618 | always_cache | 2024-04-11 16:06:00
| 2024-04-11 16:07:22 | SUCCESS | emr_hive_test |
zz_tpch_sf1000_hive_orc_zlib | cache select l_orderkey from lineitem where
l_shipdate='1994-10-28' | 2024-04-12 16:06:00 |          0 | NULL
| 100%     | AlreadyCachedSize: 15.7GB, AvgReadCacheTime: 1ms,
WriteCacheSize: 0B, AvgWriteCacheTime: 0s, TotalCacheUsage: 75.94%   |
|
| a2e3dc7e-f7d9-11ee-b03e-7ea526d0b618 | always_cache | 2024-04-11 16:01:00
| 2024-04-11 16:02:39 | SUCCESS | emr_hive_test |
zz_tpch_sf1000_hive_orc_zlib | cache select l_orderkey from lineitem where
l_shipdate='1994-10-28' | 2024-04-12 16:01:00 |          0 | NULL
| 100%     | AlreadyCachedSize: 15.7GB, AvgReadCacheTime: 1.2ms,
WriteCacheSize: 0B, AvgWriteCacheTime: 0s, TotalCacheUsage: 75.87% |
|
+--------------------------------------+--------------+--------------------
-+---------------------+---------+---------------+------------------------
-----+------------------------------------------------------------------------
+---------------------+------------+---------------+----------+------------
--------------------------------------------------------------------------------
---------------------------------+------------+
2 rows in set (0.04 sec)
```

Поле `EXTRA_MESSAGE` записывает метрики CACHE SELECT.

Удаление задач

```
DROP TASK <task_name>
```

Примеры использования

1. Во время тестирования производительности PoC, если вы хотите оценить производительность Selena без вмешательства внешних систем хранения, вы можете использовать оператор CACHE SELECT для предварительной загрузки данных тестируемой таблицы в block cache.

2. Бизнес-команде необходимо просматривать BI-отчеты каждое утро в 8 часов. Чтобы обеспечить относительно стабильную производительность запросов, вы можете запланировать задачу CACHE SELECT, которая начнет выполняться в 7 часов утра каждый день.

```
mysql> submit task BI schedule START('2024-02-03 07:00:00')
EVERY(interval 1 day)
AS cache select * from hive_catalog.test_db.lineitem
where l_shipdate='1994-10-28';
+--------------+-----------+
| TaskName     | Status    |
+--------------+-----------+
| BI           | SUBMITTED |
+--------------+-----------+
1 row in set (0.03 sec)
```

3. Чтобы минимизировать потребление системных ресурсов во время прогрева, вы можете указать переменные сессии в операторе SUBMIT TASK. Например, вы можете назначить группу ресурсов для задачи CACHE SELECT, настроить степень параллелизма (DOP) и указать условие фильтрации в WHERE, чтобы уменьшить влияние прогрева на регулярные запросы.

```
mysql> submit task cache_select properties("pipeline_dop"="1",
"resource_group"="warmup") schedule EVERY(interval 1 day)
AS cache select * from hive_catalog.test_db.lineitem where
l_shipdate>='1994-10-28';
+--------------+-----------+
| TaskName     | Status    |
+--------------+-----------+
| cache_select | SUBMITTED |
+--------------+-----------+
1 row in set (0.03 sec)
```

Ограничения и примечания по использованию

- Чтобы использовать CACHE SELECT, вы должны сначала включить функцию Block Cache и иметь привилегию SELECT на целевой таблице.
- CACHE SELECT поддерживает прогрев только одной таблицы и не поддерживает операторы типа ORDER BY, LIMIT или GROUP BY.
- CACHE SELECT может использоваться как в shared-nothing, так и в shared-data cluster.
- CACHE SELECT может прогревать удаленные файлы TEXT, ORC, Parquet.
- Данные, прогретые с помощью CACHE SELECT, могут не храниться в кеше вечно. Кешированные данные по-прежнему могут быть вытеснены на основе правила SLRU функции Block Cache.
  - Если вы пользователь озера данных, вы можете проверить оставшуюся емкость block cache, используя `SHOW BACKENDS\G` или `SHOW COMPUTE NODES\G`, чтобы оценить, может ли произойти вытеснение SLRU.
  - Если вы пользователь shared-data cluster, вы можете проверить использование block cache, просмотрев метрики shared-data cluster.
- В настоящее время реализация CACHE SELECT использует подход INSERT INTO BLACKHOLE(), который прогревает таблицу в соответствии с обычным процессом запроса. Следовательно, накладные расходы на производительность CACHE SELECT аналогичны расходам обычных запросов. Улучшения будут внесены в будущем для повышения производительности.

Что ожидать в будущих версиях

В будущем Selena представит адаптивный Block Cache Warmup для обеспечения более высокого процента попаданий в кеш.

5.8.3 Наблюдаемость Data Cache

В более ранних версиях отсутствовали богатые метрики или эффективные методы для мониторинга производительности, использования и состояния Data Cache.

В версии v1.5.2 Selena улучшает наблюдаемость Data Cache, предлагая эффективные методы мониторинга и больше метрик. Пользователи теперь могут проверять общее использование диска и памяти Data Cache, а также связанные метрики, улучшая мониторинг использования кэша.

> **ПРИМЕЧАНИЕ**
>
> Начиная с версии v1.5.2, запросы к внешним catalog и cloud-native таблицам (в shared-data cluster) используют единый экземпляр Data Cache. Поэтому, если не указано иное, следующие методы по умолчанию отображают метрики самого экземпляра Data Cache, который включает использование кэша запросов как к внешним catalog, так и к cloud-native таблицам.

SQL-команды

Вы можете выполнять SQL-команды для просмотра ёмкости и использования Data Cache на каждом узле BE.

SHOW BACKENDS

Поле `DataCacheMetrics` записывает используемое дисковое пространство и память Data Cache на конкретном BE.

```
mysql> show backends\G
*************************** 1. row ***************************
            BackendId: 10004
                   IP: XXX.XX.XX.XXX
        HeartbeatPort: 4450
               BePort: 4448
             HttpPort: 4449
             BrpcPort: 4451
        LastStartTime: 2023-12-13 20:09:30
        LastHeartbeat: 2023-12-13 20:10:43
                Alive: true
 SystemDecommissioned: false
ClusterDecommissioned: false
            TabletNum: 48
     DataUsedCapacity: 0.000 B
        AvailCapacity: 280.103 GB
        TotalCapacity: 1.968 TB
              UsedPct: 86.10 %
       MaxDiskUsedPct: 86.10 %
               ErrMsg:
              Version: datacache-heartbeat-c68caf7
               Status: {"lastSuccessReportTabletsTime":"2023-12-13
20:10:38"}
    DataTotalCapacity: 280.103 GB
          DataUsedPct: 0.00 %
             CpuCores: 104
    NumRunningQueries: 0
           MemUsedPct: 0.00 %
           CpuUsedPct: 0.0 %
     DataCacheMetrics: Status: Normal, DiskUsage: 0.00GB/2.00GB, MemUsage:
0.00GB/30.46GB
1 row in set (1.90 sec)
```

information_schema

Представление `be_datacache_metrics` в `information_schema` записывает следующую информацию, связанную с Data Cache.

```
mysql> select * from information_schema.be_datacache_metrics;
+-------+--------+------------------+-----------------+-----------------+--
--------------+-----------------+-----------------------------------------
-----------------------------------------------------+
| BE_ID | STATUS | DISK_QUOTA_BYTES | DISK_USED_BYTES | MEM_QUOTA_BYTES |
MEM_USED_BYTES | META_USED_BYTES | DIR_SPACES
|
+-------+--------+------------------+-----------------+-----------------+--
--------------+-----------------+-----------------------------------------
-----------------------------------------------------+
| 10004 | Normal |       2147483648 |               0 |     32706263420 |
0 |               0 |
[{"Path":"/home/disk1/datacache","QuotaBytes":2147483648}] |
+-------+--------+------------------+-----------------+-----------------+--
--------------+-----------------+-----------------------------------------
-----------------------------------------------------+
1 row in set (5.41 sec)
```

- `BE_ID`: ID узла BE
- `STATUS`: статус BE
- `DISK_QUOTA_BYTES`: ёмкость дискового кэша, настроенная пользователями, в байтах
- `DISK_USED_BYTES`: использованное пространство дискового кэша, в байтах
- `MEM_QUOTA_BYTES`: ёмкость кэша памяти, настроенная пользователями, в байтах
- `MEM_USED_BYTES`: использованное пространство кэша памяти, в байтах
- `META_USED_BYTES`: пространство, используемое для кэширования метаданных
- `DIR_SPACES`: путь кэша и его размер

Вызов API

Начиная с версии v1.5.2, Selena предоставляет два API для получения метрик кэша, которые отражают состояние кэша на разных уровнях:

- `/api/datacache/app_stat`: Запрос коэффициента попадания Block Cache и Page Cache.
- `/api/datacache/stat`: базовое состояние выполнения Data Cache. Этот интерфейс в основном используется для обслуживания и выявления узких мест Data Cache. Он не отражает фактический коэффициент попадания запроса. Обычным пользователям не нужно обращать внимание на этот интерфейс.

Просмотр метрик попадания в кэш

Просмотрите метрики попадания в кэш, обратившись к следующему интерфейсу API:

```
http://${BE_HOST}:${BE_HTTP_PORT}/api/datacache/app_stat
```

Возврат:

```
{
    "block_cache_hit_bytes": 1642106883,
    "block_cache_miss_bytes": 8531219739,
    "block_cache_hit_rate": 0.16,
    "block_cache_hit_bytes_last_minute": 899037056,
    "block_cache_miss_bytes_last_minute": 4163253265,
    "block_cache_hit_rate_last_minute": 0.18,
    "page_cache_hit_count": 15048,
    "page_cache_miss_count": 10032,
    "page_cache_hit_rate": 0.6,
    "page_cache_hit_count_last_minute": 10032,
    "page_cache_miss_count_last_minute": 5016,
    "page_cache_hit_rate_last_minute": 0.67
}
```

Метрика	Описание
block_cache_hit_bytes	Байты, прочитанные из Block Cache.
block_cache_miss_bytes	Байты, прочитанные из удалённого хранилища (промахи Block Cache).
block_cache_hit_rate	Коэффициент попадания Block Cache, `(block_cache_hit_bytes / (block_cache_hit_bytes + block_cache_miss_bytes))`.
block_cache_hit_bytes_last_minute	Байты, прочитанные из Block Cache за последнюю минуту.
block_cache_miss_bytes_last_minute	Байты, прочитанные из удалённого хранилища за последнюю минуту.
block_cache_hit_rate_last_minute	Коэффициент попадания Block Cache за последнюю минуту.
page_cache_hit_count	Количество страниц, прочитанных из Page Cache.
page_cache_miss_count	Количество промахов страниц в Page Cache.
block_cache_hit_rate	Коэффициент попадания Page Cache: `(page_cache_hit_count / (page_cache_hit_count + page_cache_miss_count))`.

Метрика	Описание
page_cache_hit_count_last_minute	Количество страниц, прочитанных из Page Cache за последнюю минуту.
page_cache_miss_count_last_minute	Количество промахов страниц в Page Cache за последнюю минуту.
page_cache_hit_rate_last_minute	Коэффициент попадания Page Cache за последнюю минуту.

Просмотр базового состояния выполнения Data Cache

Вы можете получить более детальные метрики о Data Cache, обратившись к следующим интерфейсам API.

```
http://${BE_HOST}:${BE_HTTP_PORT}/api/datacache/stat
```

Результаты выглядят следующим образом:

```
{
    "page_cache_mem_quota_bytes": 10679976935,
    "page_cache_mem_used_bytes": 10663052377,
    "page_cache_mem_used_rate": 1.0,
    "page_cache_hit_count": 276890,
    "page_cache_miss_count": 153126,
    "page_cache_hit_rate": 0.64,
    "page_cache_hit_count_last_minute": 11196,
    "page_cache_miss_count_last_minute": 9982,
    "page_cache_hit_rate_last_minute": 0.53,
    "block_cache_status": "NORMAL",
    "block_cache_disk_quota_bytes": 214748364800,
    "block_cache_disk_used_bytes": 11371020288,
    "block_cache_disk_used_rate": 0.05,
    "block_cache_disk_spaces":
"/disk1/sr/be/storage/datacache:107374182400;/disk2/sr/be/storage/datacache:1
    "block_cache_meta_used_bytes": 11756727,
    "block_cache_hit_count": 57707,
    "block_cache_miss_count": 2556,
    "block_cache_hit_rate": 0.96,
    "block_cache_hit_bytes": 15126253744,
    "block_cache_miss_bytes": 620687633,
    "block_cache_hit_count_last_minute": 18108,
    "block_cache_miss_count_last_minute": 2449,
    "block_cache_hit_bytes_last_minute": 4745613488,
    "block_cache_miss_bytes_last_minute": 607536783,
    "block_cache_read_disk_bytes": 15126253744,
    "block_cache_write_bytes": 11338218093,
    "block_cache_write_success_count": 43377,
    "block_cache_write_fail_count": 36394,
    "block_cache_remove_bytes": 0,
    "block_cache_remove_success_count": 0,
    "block_cache_remove_fail_count": 0,
    "block_cache_current_reading_count": 0,
    "block_cache_current_writing_count": 0,
    "block_cache_current_removing_count": 0
}
```


Описание метрик

Метрика	Описание
page_cache_mem_quota_	Текущий лимит памяти Page Cache.

Метрика	Описание
bytes	
page_cache_mem_used_bytes	Текущая фактически используемая память Page Cache.
page_cache_mem_used_rate	Текущий процент использования памяти Page Cache.
page_cache_hit_count	Количество попаданий Page Cache.
page_cache_miss_count	Количество промахов Page Cache.
page_cache_hit_rate	Коэффициент попадания Page Cache.
page_cache_hit_count_last_minute	Количество попаданий Page Cache за последнюю минуту.
page_cache_miss_count_last_minute	Количество промахов Page Cache за последнюю минуту.
page_cache_hit_rate_last_minute	Коэффициент попадания Page Cache за последнюю минуту.
block_cache_status	Статус Block Cache, включая: `NORMAL`: Экземпляр работает нормально. `ABNORMAL`: Данные не могут быть прочитаны или записаны в кэш. Проблема должна быть локализована с помощью логов. `UPDATING`: Экземпляр обновляется, например, состояние обновления во время онлайн-масштабирования.
block_cache_disk_quota_bytes	Ёмкость дискового кэша Block Cache, настроенная пользователями, в байтах.
block_cache_disk_used_bytes	Использованное пространство дискового кэша Block Cache, в байтах.
block_cache_disk_used_rate	Фактический процент использования дискового кэша Block Cache, в процентах.
block_cache_disk_spaces	Информация о дисковом кэше Block Cache, настроенная пользователями, включая каждый путь кэша и размер кэша.
block_cache_meta_used	Пространство памяти, используемое для кэширования метаданных Block Cache, в байтах.

Метрика	Описание
_bytes	
block_cache_hit_count	Количество попаданий Block Cache.
block_cache_miss_count	Количество промахов кэша.
block_cache_hit_rate	Коэффициент попадания Block Cache.
block_cache_hit_bytes	Количество байтов, попавших в Block Cache.
block_cache_miss_bytes	Количество байтов, пропущенных в Block Cache.
block_cache_hit_count_last_minute	Количество попаданий Block Cache за последнюю минуту.
block_cache_miss_count_last_minute	Количество промахов Block Cache за последнюю минуту.
block_cache_hit_bytes_last_minute	Количество байтов, попавших в Block Cache за последнюю минуту.
block_cache_miss_bytes_last_minute	Количество байтов, пропущенных в Block Cache за последнюю минуту.
block_cache_buffer_item_count	Текущее количество экземпляров Buffer в Block Cache. Экземпляры Buffer относятся к общим кэшам данных, например, при чтении части исходных данных из удалённого файла и кэшировании данных непосредственно в памяти или на дисках.
block_cache_buffer_item_bytes	Количество байтов, используемых для кэширования экземпляра Buffer в Block Cache.
block_cache_read_disk_bytes	Количество байтов, прочитанных из Block Cache.
block_cache_write_bytes	Количество байтов, записанных в Block Cache.
block_cache_write_success_count	Количество успешных записей Block Cache.
block_cache_write_fail_count	Количество неудачных записей Block Cache.

Метрика	Описание
block_cache_remove_bytes	Количество байтов, удалённых из Block Cache.
block_cache_remove_success_count	Количество успешных операций удаления из Block Cache.
block_cache_remove_fail_count	Количество неудачных операций удаления из Block Cache.
block_cache_current_reading_count	Количество операций чтения, выполняемых в данный момент в Block Cache.
block_cache_current_writing_count	Количество операций записи, выполняемых в данный момент в Block Cache.
block_cache_current_removing_count	Количество операций удаления, выполняемых в данный момент в Block Cache.

5.8.4 FAQ по Data Cache

Эта тема описывает часто задаваемые вопросы (FAQ) и распространённые проблемы, связанные с Data Cache, а также предоставляет шаги по устранению неполадок и решения этих проблем.

Включение Data Cache

Как убедиться, что Data Cache успешно включён?

В большинстве случаев вы можете проверить, успешно ли включён Data Cache, любым из следующих методов:

- Выполните `SHOW BACKENDS` (или `SHOW COMPUTE NODES`) из вашего SQL-клиента и проверьте значение `DataCacheMetrics`. Вы можете подтвердить, что Data Cache включён, если квота диска или памяти кэша больше 0.

```
mysql> show backends \G
*************************** 1. row ***************************
            BackendId: 89041
                   IP: X.X.X.X
        HeartbeatPort: 9050
               BePort: 9060
             HttpPort: 8040
             BrpcPort: 8060
        LastStartTime: 2025-05-29 14:45:37
        LastHeartbeat: 2025-05-29 19:20:32
                Alive: true
 SystemDecommissioned: false
ClusterDecommissioned: false
            TabletNum: 10
     DataUsedCapacity: 0.000 B
        AvailCapacity: 1.438 TB
        TotalCapacity: 1.718 TB
              UsedPct: 16.27 %
       MaxDiskUsedPct: 16.27 %
               ErrMsg:
              Version: main-c15b412
               Status: {"lastSuccessReportTabletsTime":"2025-05-29
19:20:30"}
    DataTotalCapacity: 1.438 TB
          DataUsedPct: 0.00 %
             CpuCores: 8
             MemLimit: 50.559GB
    NumRunningQueries: 0
           MemUsedPct: 0.50 %
           CpuUsedPct: 0.2 %
     DataCacheMetrics: Status: Normal, DiskUsage: 44MB/1TB, MemUsage: 0B/0B
             Location:
           StatusCode: OK
1 row in set (0.00 sec)
```

В приведённом выше примере квота дискового кэша Data Cache составляет 1TB, и в настоящее время используется 44MB; в то время как квота кэша памяти составляет 0B, поэтому кэш памяти не включён.

- Вы можете получить доступ к BE Web Console (`http://${BE_HOST}:${BE_HTTP_PORT}/api/datacache/stat`) для проверки текущей квоты Data Cache, коэффициента попадания и других метрик. Вы можете

подтвердить, что Data Cache включён, если `disk_quota_bytes` или `mem_quota_bytes` больше 0.

localhost:4042/api/datacache/stat


```
{
  "status": "NORMAL",
  "mem_quota_bytes": 5368709120,
  "mem_used_bytes": 4243834922,
  "disk_quota_bytes": 214748364800,
  "disk_used_bytes": 0,
  "mem_used_rate": 0.79,
  "disk_used_rate": 0,
  "disk_spaces": "/home/disk1/datacache:214748364800",
  "meta_used_bytes": 3150388,
  "hit_count": 95798,
  "miss_count": 1327,
  "hit_rate": 0.99,
  "hit_bytes": 24831547670,
  "miss_bytes": 268505666,
  "hit_count_last_minite": 92799,
  "miss_count_last_minite": 1327,
  "hit_bytes_last_minite": 24058866758,
  "miss_bytes_last_minite": 268505666,
  "buffer_item_count": 16268,
  "buffer_item_bytes": 4237890114,
  "object_item_count": 201,
  "object_item_bytes": 5944808,
  "read_mem_bytes": 24795937974,
  "read_disk_bytes": 0,
  "write_bytes": 4243834922,
  "write_success_count": 16469,
  "write_fail_count": 20856,
  "remove_bytes": 0,
  "remove_success_count": 0,
  "remove_fail_count": 0,
  "current_reading_count": 32,
  "current_writing_count": 0,
  "current_removing_count": 0
}
```

Почему Data Cache не включён по умолчанию?

Начиная с версии v1.5.2, BE будет пытаться включить Data Cache при запуске. Однако, если на текущем диске недостаточно свободного места, Data Cache не будет включён автоматически.

Это может быть вызвано следующими ситуациями:

- **Процент**: Текущее использование диска высокое.
- **Оставшееся пространство**: Размер оставшегося дискового пространства относительно мал.

Поэтому, если Data Cache не включён по умолчанию, вы можете сначала проверить текущее использование диска и при необходимости увеличить ёмкость диска.

Кроме того, вы можете вручную включить Data Cache, настроив квоту кэша в соответствии с текущим доступным дисковым пространством.

```
# отключить автоматическую настройку Data Cache
datacache_auto_adjust_enable = false
# Установить квоту диска Data Cache вручную
datacache_disk_size = 1T
```

Использование Data Cache

Какие типы catalog поддерживает Data Cache?

Data Cache в настоящее время поддерживает типы внешних catalog, которые используют Selena Native File Reader (такие как Parquet/ORC/CSV Reader), включая Hive, Iceberg, Hudi, Delta Lake и Paimon. Catalog, которые обращаются к данным на основе JNI (такие как JDBC Catalog), пока не поддерживаются.

> **ПРИМЕЧАНИЕ**
>
> Некоторые catalog могут использовать различные методы доступа к данным в зависимости от определённых условий (таких как тип файла и состояние данных). Например, для Paimon catalog, Selena может автоматически выбирать, использовать ли Native File Reader или JNI для доступа к данным в зависимости от состояния compaction текущих данных. Когда JNI используется для доступа к данным Paimon, ускорение Data Cache не поддерживается.

Как узнать, что запрос попал в кэш?

Вы можете проверить метрики, связанные с Data Cache, в соответствующем профиле запроса. Метрики `DataCacheReadBytes` и `DataCacheReadCounter` указывают на состояние попадания в локальный кэш.

```
- DataCacheReadBytes: 518.73 MB
  - __MAX_OF_DataCacheReadBytes: 4.73 MB
  - __MIN_OF_DataCacheReadBytes: 16.00 KB
- DataCacheReadCounter: 684
  - __MAX_OF_DataCacheReadCounter: 4
  - __MIN_OF_DataCacheReadCounter: 0
- DataCacheReadTimer: 737.357us
- DataCacheWriteBytes: 7.65 GB
  - __MAX_OF_DataCacheWriteBytes: 64.39 MB
  - __MIN_OF_DataCacheWriteBytes: 0.00
- DataCacheWriteCounter: 7.887K (7887)
  - __MAX_OF_DataCacheWriteCounter: 65
  - __MIN_OF_DataCacheWriteCounter: 0
- DataCacheWriteTimer: 23.467ms
  - __MAX_OF_DataCacheWriteTimer: 62.280ms
  - __MIN_OF_DataCacheWriteTimer: 0ns
```

Почему запрос пропускает кэш, когда Data Cache включён?

Выполните следующие шаги для устранения неполадок:

1. Проверьте, поддерживает ли Data Cache текущий тип catalog.
2. Подтвердите, соответствует ли оператор запроса условиям заполнения кэша. В определённых случаях Data Cache будет отклонять заполнение кэша для некоторых запросов. Подробности см. в разделе Правила заполнения Data Cache.

Команда `EXPLAIN VERBOSE` может быть использована для проверки, запускает ли запрос заполнение кэша. Пример:

```
mysql> EXPLAIN VERBOSE SELECT col1 FROM hudi_table;
|   0:HudiScanNode                          |
|      TABLE: hudi_table                    |
|      partitions=3/3                       |
|      cardinality=9084                     |
|      avgRowSize=2.0                       |
|      dataCacheOptions={populate: false} |
|      cardinality: 9084                    |
+-------------------------------------------+
```

В приведённом выше примере поле `populate` в разделе `dataCacheOptions` имеет значение `false`, что указывает на то, что кэш не будет заполняться для этого запроса.

Чтобы включить Data Cache для таких запросов, вы можете изменить поведение заполнения по умолчанию, установив системную переменную `populate_datacache_mode` в `always`.

Попадание в Data Cache

Почему иногда один и тот же запрос нужно выполнить несколько раз, прежде чем полностью попасть в кэш?

В текущей версии Data Cache по умолчанию использует асинхронное заполнение для снижения влияния на производительность запросов. При использовании асинхронного заполнения система будет пытаться кэшировать данные, к которым обращались, в фоновом режиме, не влияя на производительность чтения насколько это возможно. Поэтому выполнение запроса один раз может кэшировать только часть данных. Вам нужно выполнить запрос несколько раз, чтобы кэшировать все данные, необходимые для запроса.

Вы также можете использовать синхронное заполнение кэша, установив `enable_datacache_async_populate_mode=false`, или заранее прогреть целевые данные с помощью `CACHE SELECT`.

Почему при том, что все данные в текущем запросе были кэшированы, всё ещё есть небольшое количество данных, к которым обращаются удалённо?

В текущей версии адаптация I/O включена по умолчанию для оптимизации производительности кэша при высокой нагрузке I/O на диск, что может привести к тому, что небольшое количество запросов в некоторых случаях напрямую обращается к удалённому хранилищу.

Вы можете отключить функцию адаптации I/O, установив `enable_datacache_io-adapter` в `false`.

Прочее

Как очистить кэшированные данные?

В настоящее время Data Cache не предоставляет прямого интерфейса для очистки кэшированных данных, но вы можете выбрать один из следующих методов для очистки:

- Вы можете очистить кэшированные данные, удалив все данные (включая файлы блоков и каталоги метаданных) в директории `datacache` на узлах BE/CN, а затем перезапустив узлы. (Рекомендуется)
- Если вы хотите избежать перезапуска узлов BE/CN, вы также можете косвенно очистить кэшированные данные путём уменьшения квоты кэша во время работы. Например, если вы ранее установили квоту дискового кэша в 2TB, вы можете сначала уменьшить её до 0 (система автоматически очистит кэшированные данные), а затем снова установить её в 2TB.

Пример:

```
UPDATE be_configs SET VALUE="0" WHERE NAME="datacache_disk_size" and
BE_ID=10005;
UPDATE be_configs SET VALUE="2T" WHERE NAME="datacache_disk_size" and
BE_ID=10005;
```

> **ПРИМЕЧАНИЕ**
>
> При очистке кэшированных данных во время работы будьте осторожны с условием `WHERE` в операторе, чтобы случайно не повредить другие несвязанные параметры или узлы.

Как улучшить производительность Data Cache?

С Data Cache Selena по сути обращается к локальной памяти или диску вместо удалённого хранилища. Поэтому производительность напрямую связана с локальной средой кэширования. Если вы обнаружите, что задержка доступа к кэшу высока из-за высокой нагрузки на диск, вы можете рассмотреть улучшение производительности локальной среды кэширования:

- Приоритетно используйте высокопроизводительные NVME-диски в качестве дисков кэша.
- Если высокопроизводительные диски недоступны, вы также можете увеличить количество дисков для распределения нагрузки I/O.
- Увеличьте память сервера узлов BE/CN (а не квоту памяти Data Cache), используя Page Cache операционной системы для уменьшения количества прямых обращений к диску и нагрузки I/O на диск.

5.9 Внешняя таблица

> **ПРИМЕЧАНИЕ**
>
> Функция внешних таблиц больше не рекомендуется, за исключением определенных редких случаев использования, и может быть объявлена устаревшей в будущих выпусках. Для управления и запроса данных из внешних источников данных в общих сценариях рекомендуется использовать External Catalog.
>
> - Начиная с v1.5.2, мы рекомендуем использовать catalogs для запроса данных из Hive, Iceberg и Hudi. См. Hive catalog, Iceberg catalog и Hudi catalog.
> - Начиная с v1.5.2, мы рекомендуем использовать JDBC catalog для запроса данных из MySQL и PostgreSQL и использовать Elasticsearch catalog для запроса данных из Elasticsearch.
> - Начиная с v1.5.2, мы рекомендуем использовать JDBC catalog для запроса данных из Oracle и SQL Server.
> - Функция внешних таблиц была разработана для помощи в загрузке данных в Selena, а НЕ для выполнения эффективных запросов к внешним системам в качестве обычной операции. Более производительным решением будет загрузка данных в Selena.

Selena поддерживает доступ к другим источникам данных с использованием внешних таблиц. Внешние таблицы создаются на основе таблиц данных, которые хранятся в других источниках данных. Selena хранит только метаданные таблиц данных. Вы можете использовать внешние таблицы для прямого запроса данных из других источников данных. В настоящее время, за исключением внешних таблиц Selena, все другие внешние таблицы объявлены устаревшими. **Вы можете только записывать данные из другого cluster Selena в текущий cluster Selena. Вы не можете читать данные из него. Для источников данных, отличных от Selena, вы можете только читать данные из этих источников данных.**

Начиная с версии 2.5, Selena предоставляет функцию Data Cache, которая ускоряет запросы горячих данных во внешних источниках данных. Для получения дополнительной информации см. Data Cache.

5.9.1 Внешняя таблица Selena

Начиная с Selena 1.19, Selena позволяет вам использовать внешнюю таблицу Selena для записи данных из одного cluster Selena в другой. Это обеспечивает разделение чтения и записи и предоставляет лучшую изоляцию ресурсов. Вы можете сначала создать целевую таблицу в целевом cluster Selena. Затем в исходном cluster Selena вы можете создать внешнюю таблицу Selena, которая имеет ту же схему, что и целевая таблица, и указать информацию о целевом cluster и таблице в поле `PROPERTIES`.

Данные могут быть записаны из исходного cluster в целевой cluster с помощью оператора INSERT INTO для записи во внешнюю таблицу Selena. Это может помочь реализовать следующие цели:

- Синхронизация данных между clusters Selena.
- Разделение чтения и записи. Данные записываются в исходный cluster, а изменения данных из исходного cluster синхронизируются с целевым cluster, который предоставляет сервисы запросов.

Следующий код показывает, как создать целевую таблицу и внешнюю таблицу.

```
# Создайте целевую таблицу в целевом cluster Selena.
CREATE TABLE t
(
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    k4 VARCHAR(2048),
    k5 DATETIME
)
ENGINE=olap
DISTRIBUTED BY HASH(k1);

# Создайте внешнюю таблицу в исходном cluster Selena.
CREATE EXTERNAL TABLE external_t
(
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    k4 VARCHAR(2048),
    k5 DATETIME
)
ENGINE=olap
DISTRIBUTED BY HASH(k1)
PROPERTIES
(
    "host" = "127.0.0.1",
    "port" = "9020",
    "user" = "user",
    "password" = "passwd",
    "database" = "db_test",
    "table" = "t"
);

# Запишите данные из исходного cluster в целевой cluster путем записи
данных во внешнюю таблицу Selena. Второй оператор рекомендуется для
производственной среды.
insert into external_t values ('2020-10-11', 1, 1, 'hello', '2020-10-11
10:00:00');
insert into external_t select * from other_table;
```

Параметры:

- **EXTERNAL:** Это ключевое слово указывает, что создаваемая таблица является внешней таблицей.

- **host:** Этот параметр указывает IP-адрес узла leader FE целевого cluster Selena.
- **port:** Этот параметр указывает RPC порт узла FE целевого cluster Selena.

> **ПРИМЕЧАНИЕ**
>
> Чтобы обеспечить доступ исходного cluster, к которому принадлежат внешние таблицы Selena, к целевому cluster Selena, вы должны настроить свою сеть и брандмауэр для разрешения доступа к следующим портам:
>
> - RPC порт узла FE. См. `rpc_port` в файле конфигурации FE **fe/fe.conf**. RPC порт по умолчанию — `9020`.
> - bRPC порт узла BE. См. `brpc_port` в файле конфигурации BE **be/be.conf**. bRPC порт по умолчанию — `8060`.

- **user:** Этот параметр указывает имя пользователя, используемое для доступа к целевому cluster Selena.
- **password:** Этот параметр указывает пароль, используемый для доступа к целевому cluster Selena.
- **database:** Этот параметр указывает базу данных, к которой принадлежит целевая таблица.
- **table:** Этот параметр указывает имя целевой таблицы.

Следующие ограничения применяются при использовании внешней таблицы Selena:

- Вы можете выполнять только команды INSERT INTO и SHOW CREATE TABLE для внешней таблицы Selena. Другие методы записи данных не поддерживаются. Кроме того, вы не можете запрашивать данные из внешней таблицы Selena или выполнять операции DDL для внешней таблицы.
- Синтаксис создания внешней таблицы такой же, как при создании обычной таблицы, но имена столбцов и другая информация во внешней таблице должны быть такими же, как в целевой таблице.
- Внешняя таблица синхронизирует метаданные таблицы из целевой таблицы каждые 10 секунд. Если выполняется операция DDL для целевой таблицы, может возникнуть задержка синхронизации данных между двумя таблицами.

5.9.2 (Устарело) Внешняя таблица для базы данных, совместимой с JDBC

Начиная с v1.5.2, Selena предоставляет внешние таблицы для запроса баз данных, совместимых с JDBC. Таким образом, вы можете анализировать данные таких баз данных невероятно быстро без необходимости импорта данных в Selena. В этом разделе описывается, как создать внешнюю таблицу в Selena и запросить данные в базах данных, совместимых с JDBC.

Предварительные требования

Перед использованием внешней таблицы JDBC для запроса данных убедитесь, что FE и BE имеют доступ к URL загрузки драйвера JDBC. URL загрузки указывается параметром `driver_url` в операторе, используемом для создания ресурса JDBC.

Создание и управление ресурсами JDBC

Создание ресурса JDBC

Перед созданием внешней таблицы для запроса данных из базы данных вам необходимо создать ресурс JDBC в Selena для управления информацией о подключении к базе данных. База данных должна поддерживать драйвер JDBC и называется "целевой базой данных". После создания ресурса вы можете использовать его для создания внешних таблиц.

Выполните следующий оператор для создания ресурса JDBC с именем `jdbc0`:

```
CREATE EXTERNAL RESOURCE jdbc0
PROPERTIES (
    "type"="jdbc",
    "user"="postgres",
    "password"="changeme",
    "jdbc_uri"="jdbc:postgresql://127.0.0.1:5432/jdbc_test",

"driver_url"="https://repo1.maven.org/maven2/org/postgresql/postgresql/42.3.3
42.3.3.jar",
    "driver_class"="org.postgresql.Driver"
);
```

Обязательные параметры в `PROPERTIES` следующие:

- `type`: тип ресурса. Установите значение `jdbc`.
- `user`: имя пользователя, используемое для подключения к целевой базе данных.
- `password`: пароль, используемый для подключения к целевой базе данных.
- `jdbc_uri`: URI, который драйвер JDBC использует для подключения к целевой базе данных. Формат URI должен удовлетворять синтаксису URI базы данных. Для синтаксиса URI некоторых распространенных баз данных посетите официальные веб-сайты Oracle, PostgreSQL, SQL Server.

> Примечание: URI должен включать имя целевой базы данных. Например, в предыдущем примере кода `jdbc_test` — это имя целевой базы данных, к которой вы хотите подключиться.

- `driver_url`: URL загрузки JAR-пакета драйвера JDBC. Поддерживается HTTP URL или file URL, например, `https://repo1.maven.org/maven2/org/postgresql/postgresql/42.3.3/postgresql-42.3.3.jar` или `file:///home/disk1/postgresql-42.3.3.jar`.
- `driver_class`: имя класса драйвера JDBC. Имена классов драйверов JDBC распространенных баз данных следующие:
  - MySQL: com.mysql.jdbc.Driver (MySQL 5.x и более ранние версии), com.mysql.cj.jdbc.Driver (MySQL 6.x и более поздние версии)
  - SQL Server: com.microsoft.sqlserver.jdbc.SQLServerDriver
  - Oracle: oracle.jdbc.driver.OracleDriver
  - PostgreSQL: org.postgresql.Driver

При создании ресурса FE загружает JAR-пакет драйвера JDBC, используя URL, указанный в параметре `driver_url`, генерирует контрольную сумму и использует контрольную сумму для проверки драйвера JDBC, загруженного BE.

> Примечание: Если загрузка JAR-пакета драйвера JDBC не удается, создание ресурса также не удается.

Когда BE запрашивают внешнюю таблицу JDBC в первый раз и обнаруживают, что соответствующий JAR-пакет драйвера JDBC не существует на их машинах, BE загружают JAR-пакет драйвера JDBC, используя URL, указанный в параметре `driver_url`, и все

JAR-пакеты драйверов JDBC сохраняются в директории `${SELENA_HOME}/lib/jdbc_drivers`.

Просмотр ресурсов JDBC

Выполните следующий оператор для просмотра всех ресурсов JDBC в Selena:

```
SHOW RESOURCES;
```

> Примечание: Столбец `ResourceType` — это `jdbc`.

Удаление ресурса JDBC

Выполните следующий оператор для удаления ресурса JDBC с именем `jdbc0`:

```
DROP RESOURCE "jdbc0";
```

> Примечание: После удаления ресурса JDBC все внешние таблицы JDBC, созданные с использованием этого ресурса JDBC, становятся недоступными. Однако данные в целевой базе данных не теряются. Если вам все еще нужно использовать Selena для запроса данных в целевой базе данных, вы можете снова создать ресурс JDBC и внешние таблицы JDBC.

Создание базы данных

Выполните следующий оператор для создания и доступа к базе данных с именем `jdbc_test` в Selena:

```
CREATE DATABASE jdbc_test;
USE jdbc_test;
```

> Примечание: Имя базы данных, которое вы указываете в предыдущем операторе, не обязательно должно совпадать с именем целевой базы данных.

Создание внешней таблицы JDBC

Выполните следующий оператор для создания внешней таблицы JDBC с именем `jdbc_tbl` в базе данных `jdbc_test`:

```
create external table jdbc_tbl (
    `id` bigint NULL,
    `data` varchar(200) NULL
) ENGINE=jdbc
properties (
    "resource" = "jdbc0",
    "table" = "dest_tbl"
);
```

Обязательные параметры в `properties` следующие:

- `resource`: имя ресурса JDBC, используемого для создания внешней таблицы.
- `table`: имя целевой таблицы в базе данных.

Для поддерживаемых типов данных и сопоставления типов данных между Selena и целевыми базами данных см. [Сопоставление типов данных](External_table.md#Data type mapping).

> Примечание:
>
> - Индексы не поддерживаются.
> - Вы не можете использовать PARTITION BY или DISTRIBUTED BY для указания правил распределения данных.

Запрос внешней таблицы JDBC

Перед запросом внешних таблиц JDBC вы должны выполнить следующий оператор для включения движка Pipeline:

```
set enable_pipeline_engine=true;
```

> Примечание: Если движок Pipeline уже включен, вы можете пропустить этот шаг.

Выполните следующий оператор для запроса данных в целевой базе данных с использованием внешних таблиц JDBC.

```
select * from JDBC_tbl;
```

Selena поддерживает проталкивание предикатов путем проталкивания условий фильтрации в целевую таблицу. Выполнение условий фильтрации как можно ближе к источнику данных может улучшить производительность запросов. В настоящее время Selena может проталкивать операторы, включая бинарные операторы сравнения (`>`, `>=`, `=`, `<` и `<=`), `IN`, `IS NULL` и `BETWEEN ... AND ...`. Однако Selena не может проталкивать функции.

Сопоставление типов данных

В настоящее время Selena может запрашивать только данные базовых типов в целевой базе данных, такие как NUMBER, STRING, TIME и DATE. Если диапазоны значений данных в целевой базе данных не поддерживаются Selena, запрос сообщает об ошибке.

Сопоставление между целевой базой данных и Selena варьируется в зависимости от типа целевой базы данных.

MySQL и Selena

MySQL	Selena
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
MEDIUMINTINT	INT
BIGINT	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
DECIMAL	DECIMAL
CHAR	CHAR
VARCHAR	VARCHAR
DATE	DATE
DATETIME	DATETIME

Oracle и Selena

Oracle	Selena
CHAR	CHAR
VARCHARVARCHAR2	VARCHAR
DATE	DATE

Oracle	Selena
SMALLINT	SMALLINT
INT	INT
BINARY_FLOAT	FLOAT
BINARY_DOUBLE	DOUBLE
DATE	DATE
DATETIME	DATETIME
NUMBER	DECIMAL

PostgreSQL и Selena

PostgreSQL	Selena
SMALLINTSMALLSERIAL	SMALLINT
INTEGERSERIAL	INT
BIGINTBIGSERIAL	BIGINT
BOOLEAN	BOOLEAN
REAL	FLOAT
DOUBLE PRECISION	DOUBLE
DECIMAL	DECIMAL
TIMESTAMP	DATETIME
DATE	DATE
CHAR	CHAR
VARCHAR	VARCHAR
TEXT	VARCHAR

SQL Server и Selena

SQL Server	Selena
BOOLEAN	BOOLEAN
TINYINT	TINYINT
SMALLINT	SMALLINT
INT	INT
BIGINT	BIGINT
FLOAT	FLOAT
REAL	DOUBLE
DECIMALNUMERIC	DECIMAL
CHAR	CHAR

SQL Server	Selena
VARCHAR	VARCHAR
DATE	DATE
DATETIMEDATETIME2	DATETIME

Ограничения

- При создании внешних таблиц JDBC вы не можете создавать индексы для таблиц или использовать PARTITION BY и DISTRIBUTED BY для указания правил распределения данных для таблиц.

- При запросе внешних таблиц JDBC Selena не может проталкивать функции в таблицы.

5.9.3 (Устарело) Внешняя таблица Elasticsearch

Selena и Elasticsearch — это две популярные аналитические системы. Selena демонстрирует высокую производительность в крупномасштабных распределенных вычислениях. Elasticsearch идеально подходит для полнотекстового поиска. Selena в сочетании с Elasticsearch может предоставить более полное решение OLAP.

Пример создания внешней таблицы Elasticsearch

Синтаксис

```
CREATE EXTERNAL TABLE elastic_search_external_table
(
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    k4 VARCHAR(2048),
    k5 DATETIME
)
ENGINE=ELASTICSEARCH
PROPERTIES (
    "hosts" = "http://192.168.0.1:9200,http://192.168.0.2:9200",
    "user" = "root",
    "password" = "root",
    "index" = "tindex",
    "type" = "_doc",
    "es.net.ssl" = "true"
);
```

Следующая таблица описывает параметры.

Параметр	Обязательно	Значение по умолчанию	Описание
hosts	Да	Нет	Адрес подключения cluster Elasticsearch. Вы можете указать один или несколько адресов. Selena может разобрать версию Elasticsearch и распределение сегментов индекса по этому адресу. Selena взаимодействует с вашим cluster Elasticsearch на основе адреса, возвращаемого API операцией `GET /_nodes/http`. Следовательно, значение параметра `host` должно совпадать с адресом, возвращаемым API операцией `GET /_nodes/http`. В противном случае BE могут не иметь возможности связаться с вашим cluster Elasticsearch.
index	Да	Нет	Имя индекса Elasticsearch, созданного для таблицы в Selena. Имя может быть псевдонимом. Этот параметр поддерживает подстановочные знаки (*). Например, если вы установите `index` в `hello*`, Selena извлечет все индексы, имена которых начинаются с `hello`.
user	Нет	Пусто	Имя пользователя, используемое для входа в cluster Elasticsearch с включенной базовой аутентификацией. Убедитесь, что у вас есть доступ к `/*cluster/state/*nodes/http` и индексу.
password	Нет	Пусто	Пароль, используемый для входа в cluster Elasticsearch.
type	Нет	`_doc`	Тип индекса. Значение по умолчанию: `_doc`. Если вы хотите запросить данные в Elasticsearch 8 и более поздних версиях, вам не нужно настраивать этот параметр, потому что типы сопоставления были удалены в Elasticsearch 8 и более поздних версиях.
es.nodes.wan.only	Нет	`false`	Указывает, использует ли Selena только адреса, указанные `hosts`, для доступа к cluster Elasticsearch и получения данных. • `true`: Selena использует только адреса, указанные `hosts`, для доступа к cluster Elasticsearch и получения данных и не определяет узлы данных, на которых находятся сегменты индекса Elasticsearch. Если Selena не может получить доступ к адресам узлов данных внутри cluster Elasticsearch, вам необходимо установить этот параметр в `true`. • `false`: Selena использует адреса, указанные `host`, для определения узлов данных, на которых находятся сегменты индексов cluster Elasticsearch. После того, как Selena сгенерирует план выполнения запроса, соответствующие BE напрямую обращаются к узлам данных внутри cluster Elasticsearch для получения данных из сегментов индексов. Если Selena может получить доступ к адресам узлов данных внутри cluster Elasticsearch, мы рекомендуем сохранить значение по умолчанию `false`.
es.net.ssl	Нет	`false`	Указывает, можно ли использовать протокол HTTPS для доступа к вашему cluster Elasticsearch. Только Selena 2.4 и более поздние версии поддерживают настройку этого параметра.

Параметр	Обязательно	Значение по умолчанию	Описание
			• `true`: Для доступа к вашему cluster Elasticsearch можно использовать как протокол HTTPS, так и HTTP. • `false`: Для доступа к вашему cluster Elasticsearch можно использовать только протокол HTTP.
enable_docvalue_scan	Нет	`true`	Указывает, получать ли значения целевых полей из колоночного хранилища Elasticsearch. В большинстве случаев чтение данных из колоночного хранилища превосходит чтение данных из строкового хранилища.
enable_keyword_sniff	Нет	`true`	Указывает, определять ли поля типа TEXT в Elasticsearch на основе полей типа KEYWORD. Если этот параметр установлен в `false`, Selena выполняет сопоставление после токенизации.

Колоночное сканирование для более быстрых запросов

Если вы установите `enable_docvalue_scan` в `true`, Selena следует этим правилам при получении данных из Elasticsearch:

- **Попробовать и посмотреть**: Selena автоматически проверяет, включено ли колоночное хранилище для целевых полей. Если да, Selena получает все значения в целевых полях из колоночного хранилища.
- **Автоматическое понижение**: Если какое-либо одно из целевых полей недоступно в колоночном хранилище, Selena разбирает и получает все значения в целевых полях из строкового хранилища (`_source`).

> **ПРИМЕЧАНИЕ**
>
> - Колоночное хранилище недоступно для полей типа TEXT в Elasticsearch. Следовательно, если вы запрашиваете поля, содержащие значения типа TEXT, Selena получает значения полей из `_source`.
> - Если вы запрашиваете большое количество (больше или равно 25) полей, чтение значений полей из `docvalue` не показывает заметных преимуществ по сравнению с чтением значений полей из `_source`.

Определение полей типа KEYWORD

Если вы установите `enable_keyword_sniff` в `true`, Elasticsearch позволяет прямую загрузку данных без индекса, потому что он автоматически создаст индекс после загрузки. Для полей типа STRING Elasticsearch создаст поле как с типами TEXT, так и KEYWORD. Так работает функция Multi-Field в Elasticsearch. Сопоставление следующее:

```
"k4": {
   "type": "text",
   "fields": {
      "keyword": {
         "type": "keyword",
         "ignore_above": 256
      }
   }
}
```

Например, для выполнения фильтрации "=" по `k4`, Selena в Elasticsearch преобразует операцию фильтрации в Elasticsearch TermQuery.

Исходный SQL фильтр следующий:

```
k4 = "Selena On Elasticsearch"
```

Преобразованный Elasticsearch query DSL следующий:

```
"term" : {
    "k4": "Selena On Elasticsearch"

}
```

Первое поле `k4` — это TEXT, и оно будет токенизировано анализатором, настроенным для `k4` (или стандартным анализатором, если анализатор не был настроен для `k4`) после загрузки данных. В результате первое поле будет токенизировано на три термина: `Selena`, `On` и `Elasticsearch`. Детали следующие:

```
POST /_analyze
{
  "analyzer": "standard",
  "text": "Selena On Elasticsearch"
}
```

Результаты токенизации следующие:

```
{
   "tokens": [
      {
         "token": "selena",
         "start_offset": 0,
         "end_offset": 5,
         "type": "<ALPHANUM>",
         "position": 0
      },
      {
         "token": "on",
         "start_offset": 6,
         "end_offset": 8,
         "type": "<ALPHANUM>",
         "position": 1
      },
      {
         "token": "elasticsearch",
         "start_offset": 9,
         "end_offset": 11,
         "type": "<ALPHANUM>",
         "position": 2
      }
   ]
}
```

Предположим, вы выполняете запрос следующим образом:

```
"term" : {
    "k4": "Selena On Elasticsearch"
}
```

В словаре нет термина, который соответствует термину `Selena On Elasticsearch`, и поэтому результат не будет возвращен.

Однако, если вы установили `enable_keyword_sniff` в `true`, Selena преобразует `k4 = "Selena On Elasticsearch"` в `k4.keyword = "Selena On Elasticsearch"` для соответствия семантике SQL. Преобразованный запрос DSL `Selena On Elasticsearch` следующий:

```
"term" : {
    "k4.keyword": "Selena On Elasticsearch"
}
```

`k4.keyword` имеет тип KEYWORD. Следовательно, данные записываются в Elasticsearch как полный термин, что позволяет успешно сопоставить.

Сопоставление типов данных столбцов

При создании внешней таблицы вам необходимо указать типы данных столбцов во внешней таблице на основе типов данных столбцов в таблице Elasticsearch. Следующая таблица показывает сопоставление типов данных столбцов.

Elasticsearch	Selena
BOOLEAN	BOOLEAN
BYTE	TINYINT/SMALLINT/INT/BIGINT
SHORT	SMALLINT/INT/BIGINT
INTEGER	INT/BIGINT
LONG	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
KEYWORD	CHAR/VARCHAR
TEXT	CHAR/VARCHAR
DATE	DATE/DATETIME
NESTED	CHAR/VARCHAR
OBJECT	CHAR/VARCHAR
ARRAY	ARRAY

> **Примечание**

> - Selena читает данные типа NESTED с использованием функций, связанных с JSON.
> - Elasticsearch автоматически сглаживает многомерные массивы в одномерные массивы. Selena делает то же самое. **Поддержка запроса данных ARRAY из Elasticsearch добавлена с версии v1.5.2.**

Проталкивание предикатов

Selena поддерживает проталкивание предикатов. Фильтры могут быть протолкнуты в Elasticsearch для выполнения, что повышает производительность запросов. Следующая таблица перечисляет операторы, которые поддерживают проталкивание предикатов.

SQL синтаксис	ES синтаксис
`=`	term query
`in`	terms query
`>=, <=, >, <`	range
`and`	bool.filter
`or`	bool.should
`not`	bool.must_not
`not in`	bool.must_not + terms
`esquery`	ES Query DSL

Примеры

Функция **esquery** используется для проталкивания запросов **которые не могут быть выражены в SQL** (например, match и geoshape) в Elasticsearch для фильтрации. Первый параметр в функции esquery используется для связывания с индексом. Второй параметр — это JSON-выражение базового Query DSL, которое заключено в фигурные скобки . **JSON-выражение должно иметь только один корневой ключ**, такой как match, geo_shape или bool.

- запрос match

```
select * from es_table where esquery(k4, '{
    "match": {
       "k4": "Selena on elasticsearch"
    }
}');
```

- запрос, связанный с геолокацией

```
select * from es_table where esquery(k4, '{
  "geo_shape": {
     "location": {
        "shape": {
           "type": "envelope",
           "coordinates": [
              [
                 13,
                 53
              ],
              [
                 14,
                 52
              ]
           ]
        },
        "relation": "within"
     }
  }
}');
```

- запрос bool

```
select * from es_table where esquery(k4, ' {
     "bool": {
        "must": [
           {
              "terms": {
                 "k1": [
                    11,
                    12
                 ]
              }
           },
           {
              "terms": {
                 "k2": [
                    100
                 ]
              }
           }
        ]
     }
  }');
```

Примечания по использованию

- Elasticsearch версии ранее 5.x сканирует данные иначе, чем версии позже 5.x. В настоящее время поддерживаются **только версии позже 5.x**.
- Поддерживаются clusters Elasticsearch с включенной базовой HTTP-аутентификацией.
- Запрос данных из Selena может быть не таким быстрым, как прямой запрос данных из Elasticsearch, например, запросы, связанные с count. Причина в том, что Elasticsearch напрямую читает метаданные целевых документов без необходимости фильтрации реальных данных, что ускоряет запрос count.

5.9.4 (Устарело) Внешняя таблица Hive

Перед использованием внешних таблиц Hive убедитесь, что на ваших серверах установлен JDK 1.8.

Создание ресурса Hive

Ресурс Hive соответствует cluster Hive. Вы должны настроить cluster Hive, используемый Selena, например, адрес Hive metastore. Вы должны указать ресурс Hive, который используется внешней таблицей Hive.

- Создайте ресурс Hive с именем hive0.

```
CREATE EXTERNAL RESOURCE "hive0"
PROPERTIES (
  "type" = "hive",
  "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
);
```

- Просмотрите ресурсы, созданные в Selena.

```
SHOW RESOURCES;
```

- Удалите ресурс с именем `hive0`.

```
DROP RESOURCE "hive0";
```

Вы можете изменить `hive.metastore.uris` ресурса Hive в Selena 2.3 и более поздних версиях. Для получения дополнительной информации см. ALTER RESOURCE.

Создание базы данных

```
CREATE DATABASE hive_test;
USE hive_test;
```

Создание внешней таблицы Hive

Синтаксис

```
CREATE EXTERNAL TABLE table_name (
  col_name col_type [NULL | NOT NULL] [COMMENT "comment"]
) ENGINE=HIVE
PROPERTIES (
  "key" = "value"
);
```

Пример: Создайте внешнюю таблицу `profile_parquet_p7` в базе данных `rawdata` в cluster Hive, соответствующем ресурсу `hive0`.

```
CREATE EXTERNAL TABLE `profile_wos_p7` (
  `id` bigint NULL,
  `first_id` varchar(200) NULL,
  `second_id` varchar(200) NULL,
  `p__device_id_list` varchar(200) NULL,
  `p__is_deleted` bigint NULL,
  `p_channel` varchar(200) NULL,
  `p_platform` varchar(200) NULL,
  `p_source` varchar(200) NULL,
  `p__city` varchar(200) NULL,
  `p__province` varchar(200) NULL,
  `p__update_time` bigint NULL,
  `p__first_visit_time` bigint NULL,
  `p__last_seen_time` bigint NULL
) ENGINE=HIVE
PROPERTIES (
  "resource" = "hive0",
  "database" = "rawdata",
  "table" = "profile_parquet_p7"
);
```

Описание:

- Столбцы во внешней таблице
  - Имена столбцов должны совпадать с именами столбцов в таблице Hive.
  - Порядок столбцов **не обязательно должен быть** таким же, как порядок столбцов в таблице Hive.
  - Вы можете выбрать только **некоторые из столбцов в таблице Hive**, но вы должны выбрать все **столбцы ключей партиций**.
  - Столбцы ключей партиций внешней таблицы не нужно указывать с помощью `partition by`. Они должны быть определены в том же списке описаний, что и другие столбцы. Вам не нужно указывать информацию о партиции. Selena автоматически синхронизирует эту информацию из таблицы Hive.
  - Установите `ENGINE` в HIVE.
- PROPERTIES:
  - **hive.resource**: используемый ресурс Hive.
  - **database**: база данных Hive.
  - **table**: таблица в Hive. **view** не поддерживается.

- Следующая таблица описывает сопоставление типов данных столбцов между Hive и Selena.

Тип столбца Hive	Тип столбца Selena	Описание
INT/INTEGER	INT	
BIGINT	BIGINT	
TIMESTAMP	DATETIME	Информация о точности и часовом поясе будет потеряна при преобразовании данных TIMESTAMP в данные DATETIME. Вам нужно преобразовать данные TIMESTAMP в данные DATETIME, которые не имеют смещения часового пояса, на основе часового пояса в sessionVariable.
STRING	VARCHAR	
VARCHAR	VARCHAR	
CHAR	CHAR	
DOUBLE	DOUBLE	
FLOAT	FLOAT	
DECIMAL	DECIMAL	
ARRAY	ARRAY	

> Примечание:
>
> - В настоящее время поддерживаемые форматы хранения Hive — это Parquet, ORC и CSV. Если формат хранения — CSV, кавычки не могут использоваться в качестве escape-символов.
> - Поддерживаются форматы сжатия SNAPPY и LZ4.
> - Максимальная длина столбца строки Hive, которую можно запросить, составляет 1 МБ. Если столбец строки превышает 1 МБ, он будет обработан как нулевой столбец.

Использование внешней таблицы Hive

Запросите общее количество строк `profile_wos_p7`.

```
select count(*) from profile_wos_p7;
```

Обновление кэшированных метаданных таблицы Hive

- Информация о партициях Hive и соответствующая информация о файлах кэшируются в Selena. Кэш обновляется с интервалами, указанными `hive_meta_cache_refresh_interval_s`. Значение по умолчанию — 7200.
  - Кэшированные данные также могут быть обновлены вручную.
    a. Если партиция добавляется или удаляется из таблицы в Hive, вы должны выполнить команду `REFRESH EXTERNAL TABLE hive_t` для обновления метаданных таблицы, кэшированных в Selena. `hive_t` — это имя внешней таблицы Hive в Selena.
    b. Если данные в некоторых партициях Hive обновлены, вы должны обновить кэшированные данные в Selena, выполнив команду `REFRESH EXTERNAL TABLE hive_t PARTITION ('k1=01/k2=02', 'k1=03/k2=04')`. `hive_t` — это имя внешней таблицы Hive в Selena. `'k1=01/k2=02'` и `'k1=03/k2=04'` — это имена партиций Hive, данные которых обновлены.
    c. При выполнении `REFRESH EXTERNAL TABLE hive_t` Selena сначала проверяет, совпадает ли информация о столбцах внешней таблицы Hive с информацией о столбцах таблицы Hive, возвращаемой Hive Metastore. Если схема таблицы Hive изменяется, например добавление или удаление столбцов, Selena синхронизирует изменения с внешней таблицей Hive. После синхронизации порядок столбцов внешней таблицы Hive остается таким же, как порядок столбцов таблицы Hive, при этом столбец партиции является последним столбцом.
- Когда данные Hive хранятся в формате Parquet, ORC и CSV, вы можете синхронизировать изменения схемы (такие как ADD COLUMN и REPLACE COLUMN) таблицы Hive с внешней таблицей Hive в Selena 2.3 и более поздних версиях.

Доступ к объектному хранилищу

- Путь к файлу конфигурации FE — `fe/conf`, в который может быть добавлен файл конфигурации, если вам нужно настроить cluster Hadoop. Например: если cluster HDFS использует высокодоступный nameservice, вам нужно поместить `hdfs-`

`site.xml` в `fe/conf`. Если HDFS настроен с ViewFs, вам нужно поместить `core-site.xml` в `fe/conf`.

- Путь к файлу конфигурации BE — `be/conf`, в который может быть добавлен файл конфигурации, если вам нужно настроить cluster Hadoop. Например, если cluster HDFS использует высокодоступный nameservice, вам нужно поместить `hdfs-site.xml` в `be/conf`. Если HDFS настроен с ViewFs, вам нужно поместить `core-site.xml` в `be/conf`.

- На машине, где находится BE, настройте JAVA_HOME как среду JDK, а не среду JRE в **скрипте запуска** BE `bin/start_be.sh`, например, `export JAVA_HOME = <путь к JDK>`. Вы должны добавить эту конфигурацию в начало скрипта и перезапустить BE, чтобы конфигурация вступила в силу.

- Настройте поддержку Kerberos:

  i. Чтобы войти с `kinit -kt keytab_path principal` на все машины FE/BE, вам необходим доступ к Hive и HDFS. Вход по команде kinit действителен только в течение определенного периода времени и должен быть помещен в crontab для регулярного выполнения.
  ii. Поместите `hive-site.xml/core-site.xml/hdfs-site.xml` в `fe/conf` и поместите `core-site.xml/hdfs-site.xml` в `be/conf`.
  iii. Добавьте `-Djava.security.krb5.conf=/etc/krb5.conf` к значению опции `JAVA_OPTS` в файле **$FE_HOME/conf/fe.conf**. **/etc/krb5.conf** — это путь сохранения файла **krb5.conf**. Вы можете изменить путь в зависимости от вашей операционной системы.
  iv. Непосредственно добавьте `JAVA_OPTS="-Djava.security.krb5.conf=/etc/krb5.conf"` в файл **$BE_HOME/conf/be.conf**. **/etc/krb5.conf** — это путь сохранения файла **krb5.conf**. Вы можете изменить путь в зависимости от вашей операционной системы.
  v. При добавлении ресурса Hive вы должны передать доменное имя в `hive.metastore.uris`. Кроме того, вы должны добавить сопоставление между доменными именами Hive/HDFS и IP-адресами в файл **/etc/hosts**.
- Настройте поддержку AWS S3: Добавьте следующую конфигурацию в `fe/conf/core-site.xml` и `be/conf/core-site.xml`.

```
<configuration>
   <property>
      <name>fs.s3a.access.key</name>
      <value>******</value>
   </property>
   <property>
      <name>fs.s3a.secret.key</name>
      <value>******</value>
   </property>
   <property>
      <name>fs.s3a.endpoint</name>
      <value>s3.us-west-2.amazonaws.com</value>
   </property>
   <property>
   <name>fs.s3a.connection.maximum</name>
   <value>500</value>
   </property>
</configuration>
```

i. `fs.s3a.access.key`: AWS access key ID.
ii. `fs.s3a.secret.key`: AWS secret key.
iii. `fs.s3a.endpoint`: конечная точка AWS S3 для подключения.
iv. `fs.s3a.connection.maximum`: максимальное количество одновременных подключений из Selena к S3. Если во время запроса возникает ошибка `Timeout waiting for connection from poll`, вы можете установить этот параметр в большее значение.

5.9.5 (Устарело) Внешняя таблица Iceberg

Начиная с v1.5.2, Selena позволяет вам запрашивать данные из Apache Iceberg с использованием внешних таблиц. Чтобы запросить данные в Iceberg, вам необходимо создать внешнюю таблицу Iceberg в Selena. При создании таблицы вам необходимо установить сопоставление между внешней таблицей и таблицей Iceberg, которую вы хотите запросить.

Перед началом

Убедитесь, что Selena имеет разрешения на доступ к сервису метаданных (например, Hive metastore), файловой системе (например, HDFS) и системе объектного хранилища (например, Amazon S3 и Alibaba Cloud Object Storage Service), используемым Apache Iceberg.

Меры предосторожности

- Внешние таблицы Iceberg могут использоваться для запроса только следующих типов данных:
    - Таблицы Iceberg v1 (Analytic Data Tables). Таблицы Iceberg v2 (Row-level Deletes) в формате ORC поддерживаются начиная с v1.5.2, а таблицы Iceberg v2 в формате Parquet поддерживаются начиная с v1.5.2. Для различий между таблицами Iceberg v1 и таблицами Iceberg v2 см. Iceberg Table Spec.
    - Таблицы, сжатые в формате gzip (формат по умолчанию), Zstd, LZ4 или Snappy.
    - Файлы, которые хранятся в формате Parquet или ORC.
- Внешние таблицы Iceberg в Selena 2.3 и более поздних версиях поддерживают синхронизацию изменений схемы таблицы Iceberg, в то время как внешние таблицы Iceberg в версиях ранее Selena 2.3 не поддерживают. Если схема таблицы Iceberg изменяется, вы должны удалить соответствующую внешнюю таблицу и создать новую.

Процедура

Шаг 1: Создание ресурса Iceberg

Перед созданием внешней таблицы Iceberg вы должны создать ресурс Iceberg в Selena. Ресурс используется для управления информацией о доступе к Iceberg. Кроме того, вам также необходимо указать этот ресурс в операторе, который используется для создания внешней таблицы. Вы можете создать ресурс в соответствии с вашими бизнес-требованиями:

- Если метаданные таблицы Iceberg получены из Hive metastore, вы можете создать ресурс и установить тип catalog в `HIVE`.
- Если метаданные таблицы Iceberg получены из других сервисов, вам необходимо создать пользовательский catalog. Затем создайте ресурс и установите тип catalog в `CUSTOM`.

Создание ресурса с типом catalog `HIVE`

Например, создайте ресурс с именем `iceberg0` и установите тип catalog в `HIVE`.

```
CREATE EXTERNAL RESOURCE "iceberg0"
PROPERTIES (
   "type" = "iceberg",
   "iceberg.catalog.type" = "HIVE",
   "iceberg.catalog.hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
);
```

Следующая таблица описывает связанные параметры.

Параметр	Описание
type	Тип ресурса. Установите значение `iceberg`.
iceberg.catalog.type	Тип catalog ресурса. Поддерживаются как Hive catalog, так и пользовательский catalog. Если вы указываете Hive catalog, установите значение `HIVE`. Если вы указываете пользовательский catalog, установите значение `CUSTOM`.
iceberg.catalog.hive.metastore.uris	URI Hive metastore. Значение параметра в следующем формате: `thrift://< IP-адрес метаданных Iceberg >:< номер порта >`. Номер порта по умолчанию — 9083. Apache Iceberg использует Hive catalog для доступа к Hive metastore, а затем запрашивает метаданные таблиц Iceberg.

Создание ресурса с типом catalog `CUSTOM`

Пользовательский catalog должен наследовать абстрактный класс BaseMetastoreCatalog, и вам необходимо реализовать интерфейс IcebergCatalog. Кроме того, имя класса пользовательского catalog не может дублироваться с именем класса, который уже существует в Selena. После создания catalog упакуйте catalog и связанные с ним файлы и поместите их в путь **fe/lib** каждого frontend (FE). Затем перезапустите каждый FE. После завершения предыдущих операций вы можете создать ресурс, catalog которого является пользовательским catalog.

Например, создайте ресурс с именем `iceberg1` и установите тип catalog в `CUSTOM`.

```
CREATE EXTERNAL RESOURCE "iceberg1"
PROPERTIES (
   "type" = "iceberg",
   "iceberg.catalog.type" = "CUSTOM",
   "iceberg.catalog-impl" = "com.selena.IcebergCustomCatalog"
);
```

Следующая таблица описывает связанные параметры.

Параметр	Описание
type	Тип ресурса. Установите значение `iceberg`.
iceberg.catalog.type	Тип catalog ресурса. Поддерживаются как Hive catalog, так и пользовательский catalog. Если вы указываете Hive catalog, установите значение `HIVE`. Если вы указываете пользовательский catalog, установите значение `CUSTOM`.
iceberg.catalog-impl	Полное имя класса пользовательского catalog. FE ищут catalog на основе этого имени. Если catalog содержит пользовательские элементы конфигурации, вы должны добавить их в параметр `PROPERTIES` в виде пар ключ-значение при создании внешней таблицы Iceberg.

Вы можете изменить `hive.metastore.uris` и `iceberg.catalog-impl` ресурса Iceberg в Selena 2.3 и более поздних версиях. Для получения дополнительной информации см. ALTER RESOURCE.

Просмотр ресурсов Iceberg

```
SHOW RESOURCES;
```

Удаление ресурса Iceberg

Например, удалите ресурс с именем `iceberg0`.

```
DROP RESOURCE "iceberg0";
```

Удаление ресурса Iceberg делает все внешние таблицы, которые ссылаются на этот ресурс, недоступными. Однако соответствующие данные в Apache Iceberg не удаляются. Если вам все еще нужно запросить данные в Apache Iceberg, создайте новый ресурс и новую внешнюю таблицу.

Шаг 2: (Необязательно) Создание базы данных

Например, создайте базу данных с именем `iceberg_test` в Selena.

```
CREATE DATABASE iceberg_test;
USE iceberg_test;
```

> Примечание: Имя базы данных в Selena может отличаться от имени базы данных в Apache Iceberg.

Шаг 3: Создание внешней таблицы Iceberg

Например, создайте внешнюю таблицу Iceberg с именем `iceberg_tbl` в базе данных `iceberg_test`.

```
CREATE EXTERNAL TABLE `iceberg_tbl` (
    `id` bigint NULL,
    `data` varchar(200) NULL
) ENGINE=ICEBERG
PROPERTIES (
    "resource" = "iceberg0",
    "database" = "iceberg",
    "table" = "iceberg_table"
);
```

Следующая таблица описывает связанные параметры.

Параметр	Описание
ENGINE	Имя движка. Установите значение `ICEBERG`.
resource	Имя ресурса Iceberg, на который ссылается внешняя таблица.
database	Имя базы данных, к которой принадлежит таблица Iceberg.
table	Имя таблицы Iceberg.

> Примечание:
>
> - Имя внешней таблицы может отличаться от имени таблицы Iceberg.
> - Имена столбцов внешней таблицы должны совпадать с именами в таблице Iceberg. Порядок столбцов двух таблиц может отличаться.

Если вы определили элементы конфигурации в пользовательском catalog и хотите, чтобы элементы конфигурации вступили в силу при запросе данных, вы можете добавить элементы конфигурации в параметр `PROPERTIES` в виде пар ключ-значение при создании внешней таблицы. Например, если вы определили элемент конфигурации `custom-catalog.properties` в пользовательском catalog, вы можете выполнить следующую команду для создания внешней таблицы.

```
CREATE EXTERNAL TABLE `iceberg_tbl` (
    `id` bigint NULL,
    `data` varchar(200) NULL
) ENGINE=ICEBERG
PROPERTIES (
    "resource" = "iceberg0",
    "database" = "iceberg",
    "table" = "iceberg_table",
    "custom-catalog.properties" = "my_property"
);
```

При создании внешней таблицы вам необходимо указать типы данных столбцов во внешней таблице на основе типов данных столбцов в таблице Iceberg. Следующая таблица показывает сопоставление типов данных столбцов.

Таблица Iceberg	Внешняя таблица Iceberg
BOOLEAN	BOOLEAN
INT	TINYINT / SMALLINT / INT
LONG	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
DECIMAL(P, S)	DECIMAL
DATE	DATE / DATETIME
TIME	BIGINT
TIMESTAMP	DATETIME
STRING	STRING / VARCHAR
UUID	STRING / VARCHAR
FIXED(L)	CHAR
BINARY	VARCHAR
LIST	ARRAY

Selena не поддерживает запрос данных Iceberg, тип данных которых TIMESTAMPTZ, STRUCT и MAP.

Шаг 4: Запрос данных в Apache Iceberg

После создания внешней таблицы вы можете запросить данные в Apache Iceberg, используя внешнюю таблицу.

```
select count(*) from iceberg_tbl;
```

5.9.6 (Устарело) Внешняя таблица Hudi

Начиная с v1.5.2, Selena позволяет вам запрашивать данные из озер данных Hudi с использованием внешних таблиц Hudi, что способствует невероятно быстрой аналитике озер данных. В этом разделе описывается, как создать внешнюю таблицу Hudi в вашем cluster Selena и использовать внешнюю таблицу Hudi для запроса данных из озера данных Hudi.

Перед началом

Убедитесь, что вашему cluster Selena предоставлен доступ к Hive metastore, cluster HDFS или bucket, с которым вы можете зарегистрировать таблицы Hudi.

Меры предосторожности

- Внешние таблицы Hudi для Hudi предназначены только для чтения и могут использоваться только для запросов.
- Selena поддерживает запрос таблиц Copy on Write и Merge On Read (таблицы MOR поддерживаются с версии v1.5.2). Для различий между этими двумя типами таблиц см. Table & Query Types.
- Selena поддерживает следующие два типа запросов Hudi: Snapshot Queries и Read Optimized Queries (Hudi поддерживает выполнение Read Optimized Queries только для таблиц Merge On Read). Incremental Queries не поддерживаются. Для получения дополнительной информации о типах запросов Hudi см. Table & Query Types.
- Selena поддерживает следующие форматы сжатия для файлов Hudi: gzip, zstd, LZ4 и Snappy. Формат сжатия по умолчанию для файлов Hudi — gzip.
- Selena не может синхронизировать изменения схемы из управляемых таблиц Hudi. Для получения дополнительной информации см. Schema Evolution. Если схема управляемой таблицы Hudi изменяется, вы должны удалить связанную внешнюю таблицу Hudi из вашего cluster Selena, а затем повторно создать эту внешнюю таблицу.

Процедура

Шаг 1: Создание и управление ресурсами Hudi

Вы должны создать ресурсы Hudi в вашем cluster Selena. Ресурсы Hudi используются для управления базами данных Hudi и внешними таблицами, которые вы создаете в вашем cluster Selena.

Создание ресурса Hudi

Выполните следующий оператор для создания ресурса Hudi с именем `hudi0`:

```
CREATE EXTERNAL RESOURCE "hudi0"
PROPERTIES (
    "type" = "hudi",
    "hive.metastore.uris" = "thrift://xx.xx.xx.xx:9083"
);
```

Следующая таблица описывает параметры.

Параметр	Описание
type	Тип ресурса Hudi. Установите значение hudi.
hive.metastore.uris	Thrift URI Hive metastore, к которому подключается ресурс Hudi. После подключения ресурса Hudi к Hive metastore вы можете создавать и управлять таблицами Hudi с использованием Hive. Thrift URI имеет формат `<IP-адрес Hive metastore>:<Номер порта Hive metastore>`. Номер порта по умолчанию — 9083.

Начиная с v1.5.2, Selena позволяет изменять значение `hive.metastore.uris` ресурса Hudi. Для получения дополнительной информации см. ALTER RESOURCE.

Просмотр ресурсов Hudi

Выполните следующий оператор для просмотра всех ресурсов Hudi, созданных в вашем cluster Selena:

```
SHOW RESOURCES;
```

Удаление ресурса Hudi

Выполните следующий оператор для удаления ресурса Hudi с именем `hudi0`:

```
DROP RESOURCE "hudi0";
```

> Примечание:
>
> Удаление ресурса Hudi приводит к недоступности всех внешних таблиц Hudi, созданных с использованием этого ресурса Hudi. Однако удаление не влияет на ваши данные, хранящиеся в Hudi. Если вы все еще хотите запросить ваши данные из Hudi с использованием Selena, вы должны повторно создать ресурсы Hudi, базы данных Hudi и внешние таблицы Hudi в вашем cluster Selena.

Шаг 2: Создание баз данных Hudi

Выполните следующий оператор для создания и открытия базы данных Hudi с именем `hudi_test` в вашем cluster Selena:

```
CREATE DATABASE hudi_test;
USE hudi_test;
```

> Примечание:
>
> Имя, которое вы указываете для базы данных Hudi в вашем cluster Selena, не обязательно должно совпадать со связанной базой данных в Hudi.

Шаг 3: Создание внешних таблиц Hudi

Выполните следующий оператор для создания внешней таблицы Hudi с именем `hudi_tbl` в базе данных Hudi `hudi_test`:

```
CREATE EXTERNAL TABLE `hudi_tbl` (
    `id` bigint NULL,
    `data` varchar(200) NULL
) ENGINE=HUDI
PROPERTIES (
    "resource" = "hudi0",
    "database" = "hudi",
    "table" = "hudi_table"
);
```

Следующая таблица описывает параметры.

Параметр	Описание
ENGINE	Движок запросов внешней таблицы Hudi. Установите значение `HUDI`.
resource	Имя ресурса Hudi в вашем cluster Selena.
database	Имя базы данных Hudi, к которой принадлежит внешняя таблица Hudi в вашем cluster Selena.
table	Управляемая таблица Hudi, с которой связана внешняя таблица Hudi.

> Примечание:
>
> - Имя, которое вы указываете для внешней таблицы Hudi, не обязательно должно совпадать со связанной управляемой таблицей Hudi.
> - Столбцы во внешней таблице Hudi должны иметь те же имена, но могут быть в другой последовательности по сравнению с их аналогами в связанной управляемой таблице Hudi.
> - Вы можете выбрать некоторые или все столбцы из связанной управляемой таблицы Hudi и создать только выбранные столбцы во внешней таблице Hudi. Следующая таблица перечисляет сопоставление между типами данных, поддерживаемыми Hudi, и типами данных, поддерживаемыми Selena.

Типы данных, поддерживаемые Hudi	Типы данных, поддерживаемые Selena
BOOLEAN	BOOLEAN
INT	TINYINT/SMALLINT/INT
DATE	DATE
TimeMillis/TimeMicros	TIME
TimestampMillis/TimestampMicros	DATETIME
LONG	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE
STRING	CHAR/VARCHAR
ARRAY	ARRAY
DECIMAL	DECIMAL

> **Примечание**
>
> Selena не поддерживает запрос данных типа STRUCT или MAP, а также не поддерживает запрос данных типа ARRAY в таблицах Merge On Read.

Шаг 4: Запрос данных из внешней таблицы Hudi

После создания внешней таблицы Hudi, связанной с конкретной управляемой таблицей Hudi, вам не нужно загружать данные во внешнюю таблицу Hudi. Чтобы запросить данные из Hudi, выполните следующий оператор:

```
SELECT COUNT(*) FROM hudi_tbl;
```

5.9.7 (Устарело) Внешняя таблица MySQL

В схеме звезды данные обычно делятся на таблицы измерений и таблицы фактов. Таблицы измерений имеют меньше данных, но включают операции UPDATE. В настоящее время Selena не поддерживает прямые операции UPDATE (обновление может быть реализовано с использованием таблицы Unique Key). В некоторых сценариях вы можете хранить таблицы измерений в MySQL для прямого чтения данных.

Чтобы запросить данные MySQL, вы должны создать внешнюю таблицу в Selena и сопоставить ее с таблицей в вашей базе данных MySQL. Вам необходимо указать информацию о подключении MySQL при создании таблицы.

```
CREATE EXTERNAL TABLE mysql_external_table
(
    k1 DATE,
    k2 INT,
    k3 SMALLINT,
    k4 VARCHAR(2048),
    k5 DATETIME
)
ENGINE=mysql
PROPERTIES
(
    "host" = "127.0.0.1",
    "port" = "3306",
    "user" = "mysql_user",
    "password" = "mysql_passwd",
    "database" = "mysql_db_test",
    "table" = "mysql_table_test"
);
```

Параметры:

- **host**: адрес подключения к базе данных MySQL
- **port**: номер порта базы данных MySQL
- **user**: имя пользователя для входа в MySQL
- **password**: пароль для входа в MySQL
- **database**: имя базы данных MySQL
- **table**: имя таблицы в базе данных MySQL

5.10 Файловая внешняя таблица

Файловая внешняя таблица — это особый тип внешней таблицы. Она позволяет напрямую выполнять запросы к файлам данных Parquet и ORC во внешних системах хранения без загрузки данных в Selena. Кроме того, файловые внешние таблицы не зависят от metastore. В текущей версии Selena поддерживает следующие внешние системы хранения: HDFS, Amazon S3 и другие S3-совместимые системы хранения.

Эта функция поддерживается начиная с Selena v1.5.2.

> **ПРИМЕЧАНИЕ**
>
> - Начиная с версии 3.1 Selena поддерживает прямую загрузку данных из файлов в облачном хранилище с помощью команды INSERT и функции FILES, поэтому вам не нужно сначала создавать внешний каталог или файловую внешнюю таблицу. Кроме того, FILES() может автоматически определять схему таблицы из файлов, что значительно упрощает процесс загрузки данных.
> - Функция файловых внешних таблиц была разработана для помощи в загрузке данных в Selena, а НЕ для эффективного выполнения запросов к внешним системам в качестве обычной операции. Более производительным решением будет загрузка данных в Selena.

5.10.1 Ограничения

- Файловые внешние таблицы должны создаваться в базах данных внутри default_catalog. Вы можете выполнить SHOW CATALOGS для просмотра каталогов, созданных в cluster.
- Поддерживаются только файлы данных Parquet, ORC, Avro, RCFile и SequenceFile.
- Вы можете использовать файловые внешние таблицы только для выполнения запросов к данным в целевом файле данных. Операции записи данных, такие как INSERT, DELETE и DROP, не поддерживаются.

5.10.2 Предварительные требования

Перед созданием файловой внешней таблицы необходимо настроить cluster Selena так, чтобы Selena могла получить доступ к внешней системе хранения, где находится целевой файл данных. Конфигурации, необходимые для файловой внешней таблицы, такие же, как и для каталога Hive, за исключением того, что вам не нужно настраивать metastore. См. Каталог Hive - Подготовка к интеграции для получения дополнительной информации о конфигурациях.

5.10.3 Создание базы данных (опционально)

После подключения к cluster Selena вы можете создать файловую внешнюю таблицу в существующей базе данных или создать новую базу данных для управления файловыми внешними таблицами. Чтобы просмотреть существующие базы данных в cluster, выполните SHOW DATABASES. Затем вы можете выполнить `USE <db_name>` для переключения на целевую базу данных.

Синтаксис создания базы данных следующий.

```
CREATE DATABASE [IF NOT EXISTS] <db_name>
```

5.10.4 Создание файловой внешней таблицы

После подключения к целевой базе данных вы можете создать файловую внешнюю таблицу в этой базе данных.

Синтаксис

```
CREATE EXTERNAL TABLE <table_name>
(
    <col_name> <col_type> [NULL | NOT NULL] [COMMENT "<comment>"]
)
ENGINE=file
COMMENT ["comment"]
PROPERTIES
(
    FileLayoutParams,
    StorageCredentialParams
)
```

Параметры

Параметр	Обязательный	Описание
table_name	Да	Имя файловой внешней таблицы. Соглашения об именовании следующие: • Имя может содержать буквы, цифры (0-9) и символы подчеркивания (_). Должно начинаться с буквы. • Имя не может превышать 64 символа.
col_name	Да	Имя столбца в файловой внешней таблице. Имена столбцов в файловой внешней таблице должны совпадать с именами в целевом файле данных, но не чувствительны к регистру. Порядок столбцов в файловой внешней таблице может отличаться от порядка в целевом файле данных.
col_type	Да	Тип столбца в файловой внешней таблице. Этот параметр необходимо указать на основе типа столбца в целевом файле данных. Дополнительную информацию см. в разделе Сопоставление типов столбцов.
NULL | NOT NULL	Нет	Допускается ли NULL для столбца в файловой внешней таблице. • NULL: NULL допускается. • NOT NULL: NULL не допускается. Этот модификатор необходимо указывать на основе следующих правил: • Если этот параметр не указан для столбцов в целевом файле данных, вы можете не указывать его для столбцов в файловой внешней таблице или указать NULL для столбцов в файловой внешней таблице. • Если NULL указан для столбцов в целевом файле данных, вы можете не указывать этот параметр для столбцов в файловой внешней таблице или указать NULL для столбцов в файловой внешней таблице. • Если NOT NULL указан для столбцов в целевом файле данных, вы также должны указать NOT NULL для столбцов в файловой внешней таблице.
comment	Нет	Комментарий к столбцу в файловой внешней таблице.
ENGINE	Да	Тип движка. Установите значение file.
comment	Нет	Описание файловой внешней таблицы.
PROPERTIES	Да	• `FileLayoutParams`: указывает путь и формат целевого файла. Это свойство обязательно. • `StorageCredentialParams`: указывает информацию аутентификации, необходимую для доступа к системам объектного хранения. Это свойство обязательно только для AWS S3 и других S3-совместимых систем хранения.

FileLayoutParams

Набор параметров для доступа к целевому файлу данных.

```
"path" = "<file_path>",
"format" = "<file_format>"
"enable_recursive_listing" = "{ true | false }"
"enable_wildcards" = "{ true | false }"
```

Параметр	Обязательный	Описание
path	Да	Путь к файлу данных. • Если файл данных хранится в HDFS, формат пути: `hdfs://<IP-адрес HDFS>:<порт>/<путь>`. Порт по умолчанию — 8020. Если вы используете порт по умолчанию, указывать его не нужно. • Если файл данных хранится в AWS S3 или другой S3-совместимой системе хранения, формат пути: `s3://<имя бакета>/<папка>/`. Обратите внимание на следующие правила при вводе пути: • Если вы хотите получить доступ ко всем файлам в пути, завершите этот параметр косой чертой (`/`), например, `hdfs://x.x.x.x/user/hive/warehouse/array2d_parq/data/`. При выполнении запроса Selena просматривает все файлы данных в этом пути. Она не обходит файлы данных рекурсивно. • Если вы хотите получить доступ к одному файлу, введите путь, который напрямую указывает на этот файл, например, `hdfs://x.x.x.x/user/hive/warehouse/array2d_parq/data`. При выполнении запроса Selena сканирует только этот файл данных.
format	Да	Формат файла данных. Допустимые значения: `parquet`, `orc`, `avro`, `rctext` или `rcbinary` и `sequence`.
enable_recursive_listing	Нет	Указывает, следует ли рекурсивно обходить все файлы в текущем пути. Значение по умолчанию: `true`. Значение `true` указывает на рекурсивный обход подкаталогов, а значение `false` указывает на игнорирование подкаталогов.
enable_wildcards	Нет	Поддерживать ли использование подстановочных знаков (`*`) в `path`. Значение по умолчанию: `false`. Например, `2024-07-*` соответствует всем файлам с префиксом `2024-07-`. Этот параметр поддерживается начиная с версии 3.1.9.

StorageCredentialParams (опционально)

Набор параметров о том, как Selena интегрируется с целевой системой хранения. Этот набор параметров **опционален**.

Вам нужно настроить `StorageCredentialParams` только когда целевая система хранения — AWS S3 или другое S3-совместимое хранилище.

Для других систем хранения вы можете игнорировать `StorageCredentialParams`.

AWS S3

Если вам нужен доступ к файлу данных, хранящемуся в AWS S3, настройте следующие параметры аутентификации в `StorageCredentialParams`.

- Если вы выбираете метод аутентификации на основе профиля экземпляра, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.region" = "<aws_s3_region>"
```

- Если вы выбираете метод аутентификации на основе предполагаемой роли, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "true",
"aws.s3.iam_role_arn" = "<ARN вашей предполагаемой роли>",
"aws.s3.region" = "<aws_s3_region>"
```

- Если вы выбираете метод аутентификации на основе пользователя IAM, настройте `StorageCredentialParams` следующим образом:

```
"aws.s3.use_instance_profile" = "false",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>",
"aws.s3.region" = "<aws_s3_region>"
```

Имя параметра	Обязательный	Описание
aws.s3.use_instance_profile	Да	Указывает, следует ли включить метод аутентификации на основе профиля экземпляра и метод аутентификации на основе предполагаемой роли при доступе к AWS S3. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`.
aws.s3.iam_role_arn	Да	ARN роли IAM, которая имеет привилегии на ваш бакет AWS S3. Если вы используете метод аутентификации на основе предполагаемой роли для

Имя параметра	Обязательный	Описание
		доступа к AWS S3, вы должны указать этот параметр. Тогда Selena примет эту роль при доступе к целевому файлу данных.
aws.s3.region	Да	Регион, в котором находится ваш бакет AWS S3. Пример: us-west-1.
aws.s3.access_key	Нет	Ключ доступа вашего пользователя IAM. Если вы используете метод аутентификации на основе пользователя IAM для доступа к AWS S3, вы должны указать этот параметр.
aws.s3.secret_key	Нет	Секретный ключ вашего пользователя IAM. Если вы используете метод аутентификации на основе пользователя IAM для доступа к AWS S3, вы должны указать этот параметр.

Информацию о том, как выбрать метод аутентификации для доступа к AWS S3 и как настроить политику контроля доступа в консоли AWS IAM, см. в разделе Параметры аутентификации для доступа к AWS S3.

S3-совместимое хранилище

Если вам нужен доступ к S3-совместимой системе хранения, такой как MinIO, настройте `StorageCredentialParams` следующим образом для обеспечения успешной интеграции:

```
"aws.s3.enable_ssl" = "false",
"aws.s3.enable_path_style_access" = "true",
"aws.s3.endpoint" = "<s3_endpoint>",
"aws.s3.access_key" = "<iam_user_access_key>",
"aws.s3.secret_key" = "<iam_user_secret_key>"
```

В следующей таблице описаны параметры, которые необходимо настроить в `StorageCredentialParams`.

Параметр	Обязательный	Описание
aws.s3.enable_ssl	Да	Указывает, следует ли включить SSL-соединение. Допустимые значения: `true` и `false`. Значение по умолчанию: `true`.
aws.s3.enable_path_style_access	Да	Указывает, следует ли включить доступ в стиле пути. Допустимые значения: `true` и `false`. Значение по умолчанию: `false`. Для MinIO вы должны установить значение `true`. URL в стиле пути используют следующий формат: `https://s3.<region_code>.amazonaws.com/<buck`

Параметр	Обязательный	Описание
		`et_name>/<key_name>`. Например, если вы создаете бакет с именем `DOC-EXAMPLE-BUCKET1` в регионе US West (Oregon) и хотите получить доступ к объекту `alice.jpg` в этом бакете, вы можете использовать следующий URL в стиле пути: `https://s3.us-west-2.amazonaws.com/DOC-EXAMPLE-BUCKET1/alice.jpg`.
aws.s3.endpoint	Да	Конечная точка, используемая для подключения к S3-совместимой системе хранения вместо AWS S3.
aws.s3.access_key	Да	Ключ доступа вашего пользователя IAM.
aws.s3.secret_key	Да	Секретный ключ вашего пользователя IAM.

Сопоставление типов столбцов

В следующей таблице представлено сопоставление типов столбцов между целевым файлом данных и файловой внешней таблицей.

Файл данных	Файловая внешняя таблица
INT/INTEGER	INT
BIGINT	BIGINT
TIMESTAMP	DATETIME. Обратите внимание, что TIMESTAMP преобразуется в DATETIME без часового пояса на основе настройки часового пояса текущей сессии и теряет часть своей точности.
STRING	STRING
VARCHAR	VARCHAR
CHAR	CHAR
DOUBLE	DOUBLE
FLOAT	FLOAT
DECIMAL	DECIMAL
BOOLEAN	BOOLEAN
ARRAY	ARRAY

Файл данных	Файловая внешняя таблица
MAP	MAP
STRUCT	STRUCT

Примеры

HDFS

Создание файловой внешней таблицы с именем `t0` для выполнения запросов к файлам данных Parquet, хранящимся в пути HDFS.

```
USE db_example;

CREATE EXTERNAL TABLE t0
(
    name string,
    id int
)
ENGINE=file
PROPERTIES
(
    "path"="hdfs://x.x.x.x:8020/user/hive/warehouse/person_parq/",
    "format"="parquet"
);
```

AWS S3

Пример 1: Создание файловой внешней таблицы и использование **профиля экземпляра** для доступа к **одному файлу Parquet** в AWS S3.

```
USE db_example;

CREATE EXTERNAL TABLE table_1
(
    name string,
    id int
)
ENGINE=file
PROPERTIES
(
    "path" = "s3://bucket-test/folder1/raw_0.parquet",
    "format" = "parquet",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.region" = "us-west-2"
);
```

Пример 2: Создание файловой внешней таблицы и использование **предполагаемой роли** для доступа ко **всем файлам ORC** в целевом пути файла в AWS S3.

```
USE db_example;

CREATE EXTERNAL TABLE table_1
(
    name string,
    id int
)
ENGINE=file
PROPERTIES
(
    "path" = "s3://bucket-test/folder1/",
    "format" = "orc",
    "aws.s3.use_instance_profile" = "true",
    "aws.s3.iam_role_arn" =
"arn:aws:iam::51234343412:role/role_name_in_aws_iam",
    "aws.s3.region" = "us-west-2"
);
```

Пример 3: Создание файловой внешней таблицы и использование **пользователя IAM** для доступа ко **всем файлам ORC** в пути файла в AWS S3.

```
USE db_example;

CREATE EXTERNAL TABLE table_1
(
    name string,
    id int
)
ENGINE=file
PROPERTIES
(
    "path" = "s3://bucket-test/folder1/",
    "format" = "orc",
    "aws.s3.use_instance_profile" = "false",
    "aws.s3.access_key" = "<iam_user_access_key>",
    "aws.s3.secret_key" = "<iam_user_access_key>",
    "aws.s3.region" = "us-west-2"
);
```

5.10.5 Выполнение запросов к файловой внешней таблице

Синтаксис:

```
SELECT <clause> FROM <file_external_table>
```

Например, чтобы выполнить запрос к данным из файловой внешней таблицы `t0`, созданной в разделе Примеры - HDFS, выполните следующую команду:

```
SELECT * FROM t0;

+--------+------+
| name   | id   |
+--------+------+
| jack   |    2 |
| lily   |    1 |
+--------+------+
2 rows in set (0.08 sec)
```

5.10.6 Управление файловыми внешними таблицами

Вы можете просмотреть схему таблицы с помощью DESC или удалить таблицу с помощью DROP TABLE.

5.11 FAQ по Data Lake

Эта тема описывает некоторые часто задаваемые вопросы (FAQ) о Data Lake и предоставляет решения этих проблем. Некоторые метрики, упомянутые в этой теме, могут быть получены только из профилей SQL-запросов. Чтобы получить профили SQL-запросов, необходимо указать `set enable_profile=true`.

5.11.1 Медленные HDFS DataNode

Описание проблемы

При доступе к файлам данных, хранящимся в вашем HDFS cluster, вы можете обнаружить большую разницу между значениями метрик `__MAX_OF_FSIOTime` и `__MIN_OF_FSIOTime` в профилях выполняемых SQL-запросов. Это указывает на то, что некоторые DataNode в HDFS cluster работают медленно. Следующий пример является типичным профилем, указывающим на проблему медленных HDFS DataNode:

```
- InputStream: 0
  - AppIOBytesRead: 22.72 GB
    - __MAX_OF_AppIOBytesRead: 187.99 MB
    - __MIN_OF_AppIOBytesRead: 64.00 KB
  - AppIOCounter: 964.862K (964862)
    - __MAX_OF_AppIOCounter: 7.795K (7795)
    - __MIN_OF_AppIOCounter: 1
  - AppIOTime: 1s372ms
    - __MAX_OF_AppIOTime: 4s358ms
    - __MIN_OF_AppIOTime: 1.539ms
  - FSBytesRead: 15.40 GB
    - __MAX_OF_FSBytesRead: 127.41 MB
    - __MIN_OF_FSBytesRead: 64.00 KB
  - FSIOCounter: 1.637K (1637)
    - __MAX_OF_FSIOCounter: 12
    - __MIN_OF_FSIOCounter: 1
  - FSIOTime: 9s357ms
    - __MAX_OF_FSIOTime: 60s335ms
    - __MIN_OF_FSIOTime: 1.536ms
```

Решение

Вы можете использовать одно из следующих решений для устранения этой проблемы:

- **[Рекомендуется]** Включите функцию Data Cache, которая устраняет влияние медленных HDFS DataNode на запросы путём автоматического кэширования данных из внешних систем хранения на BE или CN вашего Selena cluster.
- **[Рекомендуется]** Сократите время ожидания между HDFS-клиентом и DataNode. Это решение подходит, когда Data Cache не помогает решить проблему медленных HDFS DataNode.
- Включите функцию Hedged Read. При включении этой функции, если чтение из блока происходит медленно, Selena запускает новое чтение, которое выполняется параллельно с исходным чтением, для чтения из другой replica блока. Как только одно из двух чтений возвращается, другое отменяется. **Функция Hedged Read может помочь ускорить чтение, но она также значительно увеличивает потребление heap-памяти на виртуальных машинах Java (JVM). Поэтому, если ваши физические машины имеют небольшой объём памяти, мы рекомендуем не включать функцию Hedged Read.**

[Рекомендуется] Data Cache

См. раздел Data Cache.

[Рекомендуется] Сокращение времени ожидания между HDFS-клиентом и DataNode

Настройте свойство `dfs.client.socket-timeout` в файле `hdfs-site.xml`, чтобы сократить время ожидания между HDFS-клиентом и DataNode. (Время ожидания по умолчанию составляет 60 секунд, что довольно много.) Таким образом, когда Selena сталкивается с медленным DataNode, запрос на подключение от него может истечь за очень короткий период времени, а затем быть перенаправлен на другой DataNode. Следующий пример устанавливает время ожидания в 5 секунд:

```
<configuration>
  <property>
      <name>dfs.client.socket-timeout</name>
      <value>5000</value>
   </property>
</configuration>
```

Hedged Read

Используйте следующие параметры (поддерживаются с версии v1.5.2) в файле конфигурации BE или CN `be.conf` для включения и настройки функции Hedged Read в вашем HDFS cluster.

Параметр	Значение по умолчанию	Описание
hdfs_client_enable_hedged_read	false	Указывает, следует ли включить функцию Hedged Read.
hdfs_client_hedged_read_threadpool_size	128	Указывает размер пула потоков Hedged Read на вашем HDFS-клиенте. Размер пула потоков ограничивает количество потоков, выделенных для выполнения hedged-чтений в вашем HDFS-клиенте. Этот параметр эквивалентен параметру `dfs.client.hedged.read.threadpool.size` в файле `hdfs-site.xml` вашего HDFS cluster.
hdfs_client_hedged_read_threshold_millis	2500	Указывает количество миллисекунд ожидания перед запуском hedged-чтения. Например, вы установили этот параметр в `30`. В этой ситуации, если чтение из блока не вернулось в течение 30 миллисекунд, ваш HDFS-клиент немедленно запускает hedged-чтение из другой replica блока. Этот параметр эквивалентен параметру `dfs.client.hedged.read.threshold.millis` в файле `hdfs-site.xml` вашего HDFS cluster.

Если значение любой из следующих метрик в профилях ваших запросов превышает `0`, функция Hedged Read включена.

Метрика	Описание
TotalHedgedReadOps	Количество запущенных hedged-чтений.
TotalHedgedReadOpsInCurThread	Количество раз, когда Selena пришлось запустить hedged-чтение в текущем потоке вместо нового потока, потому что пул потоков Hedged Read достиг максимального размера, указанного параметром `hdfs_client_hedged_read_threadpool_size`.
TotalHedgedReadOpsWin	Количество раз, когда hedged-чтение опередило исходное чтение.

5.11.2 Как решить ошибку "ERROR 1064 (HY000): Type mismatches on column [is_refund], JDBC result type is Integer, please set the

type to one of tinyint,smallint,int,bigint" при запросе таблицы в Hive Catalog?

Эта проблема вызвана неправильной конфигурацией JDBC-подключения. Добавьте параметр `tinyInt1isBit=false` в ваш JDBC URI для предотвращения этой проблемы:

```
"jdbc_uri" = "jdbc:mysql://xxx:3306?
database=yl_spmibill&tinyInt1isBit=false"
```

5.11.3 Почему я не могу запросить последние обновлённые данные в Iceberg Catalog (даже после обновления или пересоздания catalog), и как это устранить?

Сначала проверьте, не вызвана ли проблема включённым Data Cache. Выполните следующие шаги для проверки:

1. Сравните сканируемые файлы данных между Selena и Spark:
   - В Selena: `select file_path, spec_id from db.table_name$files;`
   - В Spark: `select file_path, spec_id from db.table_name.files;`
2. Если результаты совпадают, продолжите устранение неполадок, отключив Data Cache и выполнив запрос снова, чтобы проверить, сохраняется ли проблема.

Причина: Обновление данных таблицы Iceberg означает перезапись старых файлов, что повреждает исторические данные Iceberg. Правильное поведение — генерировать новые имена файлов при записи обновлений. Data Cache в Selena использует имя файла, размер файла и время модификации для определения, действительны ли кэшированные данные. Поскольку Iceberg не перезаписывает файлы, а время модификации всегда равно 0, Selena ошибочно считает файлы неизменёнными и читает из кэша, что приводит к устаревшим результатам запросов.

5.11.4 FE постоянно падает после запросов к таблицам во внешнем catalog, интегрированном с JuiceFS. Как это решить?

Чтобы решить эту проблему, перезапустите FE после добавления следующих параметров конфигурации в **fe.conf**:

```
proc_profile_mem_enable=false
proc_profile_cpu_enable=false
```

5.12 Поддержка функций: аналитика Data Lake

Начиная с версии v1.5.2, Selena поддерживает управление внешними источниками данных и анализ данных в Data Lake через внешние catalog.

Этот документ описывает поддержку функций для внешних catalog и поддерживаемые версии задействованных функций.

5.12.1 Универсальные функции

В этом разделе перечислены универсальные функции внешних catalog, включая системы хранения, читатели файлов, учётные данные, привилегии и Data Cache.

Внешние системы хранения

Система хранения	Поддерживаемая версия
HDFS	v1.5.2+
AWS S3	v1.5.2+
Microsoft Azure Storage	v1.5.2+
Google GCS	v1.5.2+
Alibaba Cloud OSS	v1.5.2+
Huawei Cloud OBS	v1.5.2+
Tencent Cloud COS	v1.5.2+
Volcengine TOS	v1.5.2+
Kingsoft Cloud KS3	v1.5.2+
MinIO	v1.5.2+
Ceph S3	v1.5.2+

В дополнение к нативной поддержке систем хранения, перечисленных выше, Selena также поддерживает следующие типы сервисов объектного хранения:

- **HDFS-совместимые сервисы объектного хранения, такие как COS Cloud HDFS, OSS-HDFS и OBS PFS**
  - **Описание**: Необходимо указать префикс URI объектного хранилища в параметре конфигурации BE `fallback_to_hadoop_fs_list` и загрузить .jar-пакет,

предоставленный облачным поставщиком, в директорию **/lib/hadoop/hdfs/**. Обратите внимание, что вы должны создать внешний catalog, используя префикс, указанный в `fallback_to_hadoop_fs_list`.
  - **Поддерживаемые версии**: v1.5.2+, v1.5.2+
- **S3-совместимые сервисы объектного хранения, отличные от перечисленных выше**
  - **Описание**: Необходимо указать префикс URI объектного хранилища в параметре конфигурации BE `s3_compatible_fs_list`. Обратите внимание, что вы должны создать внешний catalog, используя префикс, указанный в `s3_compatible_fs_list`.
  - **Поддерживаемые версии**: v1.5.2+, v1.5.2+

Форматы сжатия

В этом разделе перечислены только форматы сжатия, поддерживаемые каждым форматом файлов. Информацию о форматах файлов, поддерживаемых каждым внешним catalog, см. в разделе соответствующего внешнего catalog.

Формат файла	Форматы сжатия
Parquet	NO_COMPRESSION, SNAPPY, LZ4, ZSTD, GZIP, LZO (v1.5.2+)
ORC	NO_COMPRESSION, ZLIB, SNAPPY, LZO, LZ4, ZSTD
Text	NO_COMPRESSION, LZO (v1.5.2+)
Avro	NO_COMPRESSION, DEFLATE, SNAPPY, BZIP2 (v1.5.2+)
RCFile	NO_COMPRESSION, DEFLATE, SNAPPY, GZIP (v1.5.2+)
SequenceFile	NO_COMPRESSION, DEFLATE, SNAPPY, BZIP2, GZIP (v1.5.2+)

> **ПРИМЕЧАНИЕ**
>
> Форматы файлов Avro, RCFile и SequenceFile читаются через Java Native Interface (JNI), а не через нативные читатели Selena. Поэтому производительность чтения для этих форматов файлов может быть не такой хорошей, как для Parquet и ORC.

Управление, учётные данные и контроль доступа

Функция	Описание	Поддерживаемые версии
Information Schema	Поддерживает Information Schema для внешних catalog.	v1.5.2+

Функция	Описание	Поддерживаемые версии
Контроль доступа к Data Lake	Поддерживает нативную модель RBAC Selena для внешних catalog. Вы можете управлять привилегиями баз данных, таблиц и представлений (в настоящее время только Hive views и Iceberg views) во внешних catalog так же, как и в default catalog Selena.	v1.5.2+
Повторное использование внешних сервисов на Apache Ranger	Поддерживает повторное использование внешнего сервиса (например, Hive Service) на Apache Ranger для контроля доступа.	v1.5.2+
Аутентификация Kerberos	Поддерживает аутентификацию Kerberos для HDFS или Hive Metastore.	v1.5.2+

Data Cache

Функция	Описание	Поддерживаемые версии
Data Cache (Block Cache)	Начиная с версии v1.5.2, Selena поддерживала функцию Data Cache (тогда называвшуюся Block Cache), реализованную с использованием CacheLib, что приводило к ограниченному потенциалу оптимизации её расширяемости. Начиная с версии v1.5.2, Selena переработала реализацию кэша и добавила новые функции в Data Cache, что привело к улучшению производительности с каждой последующей версией.	v1.5.2+
Ребалансировка данных между локальным и дисками	Поддерживает стратегию ребалансировки данных для обеспечения контроля перекоса данных в пределах 10%.	v1.5.2+
Замена Block Cache на Data Cache	**Изменения параметров** Конфигурации BE: • Замените `block_cache_enable` на `datacache_enable`. • Замените `block_cache_mem_size` на `datacache_mem_size`. • Замените `block_cache_disk_size` на `datacache_disk_size`. • Замените `block_cache_disk_path` на `datacache_disk_path`. • Замените `block_cache_meta_path` на `datacache_meta_path`. • Замените `block_cache_block_size` на `datacache_block_size`. Сессионные переменные: • Замените `enable_scan_block_cache` на `enable_scan_datacache`.	v1.5.2+

Функция	Описание	Поддерживаемые версии
	• Замените `enable_populate_block_cache` на `enable_populate_datacache`. После обновления cluster до версии, где доступен Data Cache, параметры Block Cache по-прежнему действуют. Новые параметры переопределят старые после включения Data Cache. Смешанное использование обеих групп параметров не допускается. В противном случае некоторые параметры не вступят в силу.	
Новые метрики для API мониторинга Data Cache	Поддерживает отдельный API для мониторинга Data Cache, включая ёмкость кэша и попадания. Вы можете просматривать метрики Data Cache через интерфейс `http://${BE_HOST}:${BE_HTTP_PORT}/api/datacache/stat`.	v1.5.2 +
Memory Tracker для Data Cache	Поддерживает Memory Tracker для Data Cache. Вы можете просматривать метрики, связанные с памятью, через интерфейс `http://${BE_HOST}:${BE_HTTP_PORT}/mem_tracker`.	v1.5.2 +
Прогрев Data Cache	Выполняя CACHE SELECT, вы можете заранее активно заполнить кэш нужными данными из удалённого хранилища, чтобы первый запрос не занимал слишком много времени на получение данных. CACHE SELECT не выводит данные и не выполняет вычисления. Он только получает данные.	v1.5.2 +

5.12.2 Hive Catalog

Метаданные

При выполнении запросов к данным Hive через Hive catalog Selena будет кэшировать метаданные таблиц для снижения затрат на частый доступ к удалённому хранилищу. Этот механизм обеспечивает производительность запросов, поддерживая свежесть данных через политику асинхронного обновления и истечения срока действия.

Кэшируемые метаданные

Selena будет кэшировать следующие метаданные во время запросов:

- **Метаданные уровня таблицы или partition**
  - Содержимое:
    - Информация о таблице: база данных, схема таблицы, имена столбцов и ключи partition

        - Информация о partition: список partition и расположение partition
    - Влияние: обнаружение существования таблицы (была ли таблица удалена и/или пересоздана)
    - Свойства catalog:
        - `enable_metastore_cache`: Управляет включением кэша metastore. Значение по умолчанию: `true`.
        - `metastore_cache_refresh_interval_sec`: Управляет интервалом времени, в течение которого кэшированные метаданные считаются свежими. Значение по умолчанию: `60`. Единица измерения: секунды.
    - Расположение: Metastore (HMS или Glue)
- **Список имён partition**
    - Содержимое: Список имён partition, используемый для поиска и отсечения partition. Хотя список имён partition был собран как информация о partition в разделе выше, существует отдельная конфигурация для включения или отключения этой функции в определённых обстоятельствах.
    - Влияние: обнаружение существования partition (появился ли новый partition или partition был удалён и/или пересоздан)
    - Свойства catalog:
        - `enable_cache_list_names`: Управляет включением кэша списка имён partition. Значение по умолчанию: `true`.
        - `metastore_cache_refresh_interval_sec`: Управляет интервалом времени, в течение которого кэшированные метаданные считаются свежими. Значение по умолчанию: `60`. Единица измерения: секунды.
    - Расположение: Metastore (HMS или Glue)
- **Метаданные уровня файла**
    - Содержимое: Пути к файлам в папке partition.
    - Влияние: Загрузка данных в существующий partition.
    - Свойства catalog:
        - `enable_remote_file_cache`: Управляет включением кэша метаданных для файлов в удалённом хранилище. Значение по умолчанию: `true`.
        - `remote_file_cache_refresh_interval_sec`: Управляет интервалом времени, в течение которого метаданные файлов считаются свежими. Значение по умолчанию: `60`. Единица измерения: секунды.
        - `remote_file_cache_memory_ratio`: Управляет долей памяти, которая может использоваться для кэша метаданных файлов. Значение по умолчанию: `0.1`

(10%).
- Расположение: Удалённое хранилище (HDFS или S3)

Политика асинхронного обновления

Следующий параметр конфигурации FE управляет политикой асинхронного обновления метаданных:

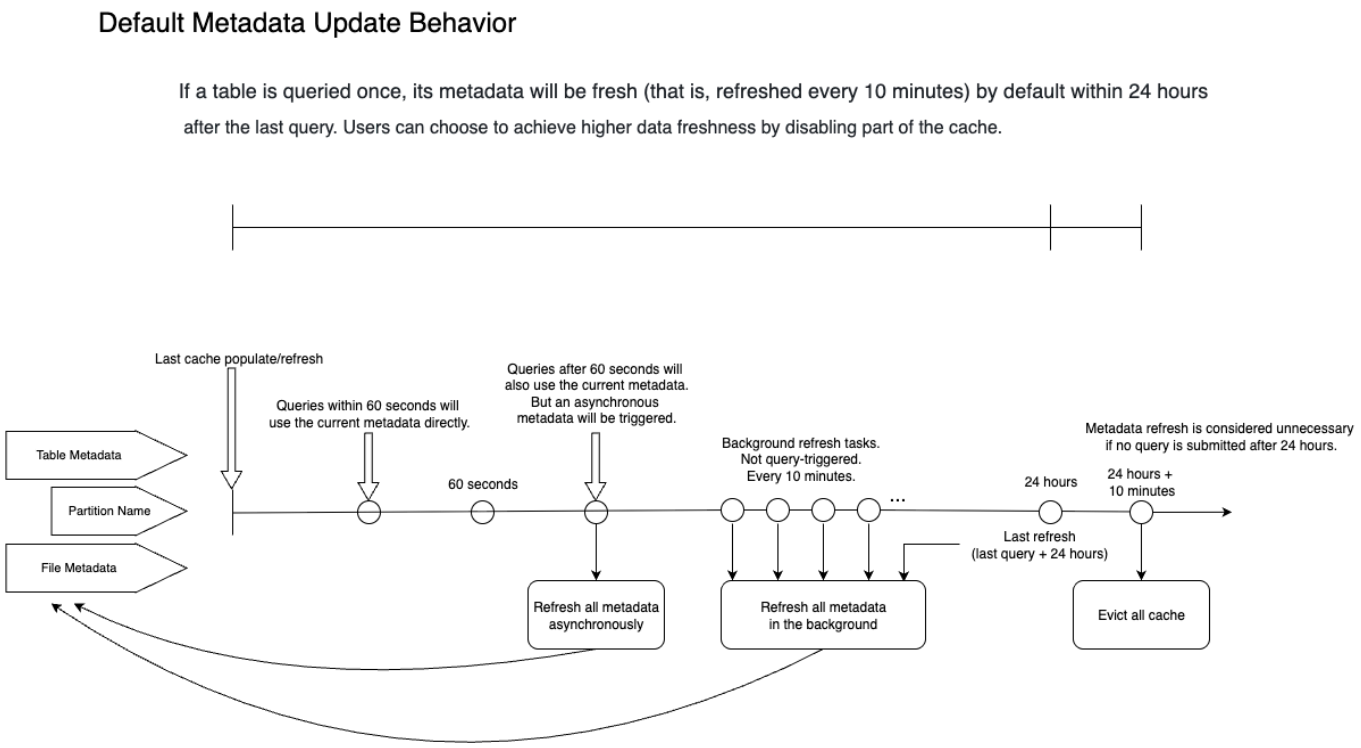
Параметр конфигурации	Значение по умолчанию	Описание
enable_background_refresh_connector_metadata	`true` в v1.5.2 `false` в v1.5.2	Включать ли периодическое обновление кэша метаданных. После включения Selena опрашивает metastore и обновляет кэшированные метаданные часто используемых внешних catalog для обнаружения изменений данных. `true` означает включить обновление кэша метаданных Hive, `false` означает отключить. Это динамический параметр FE. Вы можете изменить его с помощью команды ADMIN SET FRONTEND CONFIG.
background_refresh_metadata_interval_millis	`600000` (10 минут)	Интервал между двумя последовательными обновлениями кэша метаданных. Единица измерения: миллисекунды. Это динамический параметр FE. Вы можете изменить его с помощью команды ADMIN SET FRONTEND CONFIG.
background_refresh_metadata_time_secs_since_last_access_secs	`86400` (24 часа)	Время истечения срока действия задачи обновления кэша метаданных. Для внешнего catalog, к которому был доступ, если к нему не обращались более указанного времени, Selena прекращает обновление его кэшированных метаданных. Для внешнего catalog, к которому не было доступа, Selena не будет обновлять его кэшированные метаданные. Единица измерения: секунды. Это динамический параметр FE. Вы можете изменить его с помощью команды ADMIN SET FRONTEND CONFIG.

Поведение кэша метаданных

В этом разделе используется поведение по умолчанию для объяснения поведения метаданных во время обновления метаданных и запросов.

По умолчанию, когда выполняется запрос к таблице, Selena кэширует метаданные таблицы, partition и файлов и поддерживает их активными в течение следующих 24 часов. В течение 24 часов система обеспечит обновление кэша не реже, чем каждые 10 минут (обратите внимание, что 10 минут — это расчётное время для раунда обновления метаданных. Если есть слишком много внешних таблиц, ожидающих обновления метаданных, общий интервал обновления метаданных может быть больше 10 минут).

Если к таблице не обращались более 24 часов, Selena удаляет связанные метаданные. Другими словами, любой запрос, который вы делаете в течение 24 часов, в худшем случае будет использовать метаданные 10-минутной давности.



Подробнее:

1. Предположим, первый запрос затрагивает partition `P1` таблицы `A`. Selena кэширует метаданные уровня таблицы, списки имён partition и информацию о путях к файлам в `P1`. Кэш заполняется синхронно во время выполнения запроса.
2. Если второй запрос отправлен в течение 60 секунд после заполнения кэша и затрагивает partition `P1` таблицы `A`, Selena использует кэш метаданных напрямую, и в этот момент Selena считает все кэшированные метаданные свежими (`metastore_cache_refresh_interval_sec` и `remote_file_cache_refresh_interval_sec` управляют временным окном, в течение которого Selena считает метаданные свежими).
3. Если третий запрос отправлен через 90 секунд и затрагивает partition `P1` таблицы `A`, Selena всё равно будет использовать кэш метаданных напрямую для завершения запроса. Однако, поскольку прошло более 60 секунд с момента последнего обновления метаданных, Selena будет считать метаданные устаревшими. Поэтому Selena запустит асинхронное обновление для устаревших метаданных. Асинхронное обновление не повлияет на результат текущего запроса, потому что запрос всё ещё будет использовать устаревшие метаданные.

4. Поскольку был выполнен запрос к partition `P1` таблицы `A`, ожидается, что метаданные будут обновляться каждые 10 минут (управляется `background_refresh_metadata_interval_millis`) в течение следующих 24 часов (управляется `background_refresh_metadata_time_secs_since_last_access_secs`). Фактический интервал между раундами обновления метаданных также зависит от общего количества ожидающих задач обновления в системе.
5. Если таблица `A` не участвует ни в каком запросе в течение 24 часов, Selena удалит её кэш метаданных через 24 часа.

Лучшие практики

Поддержка Hive Catalog для Hive Metastore (HMS) и AWS Glue в основном совпадает, за исключением того, что функция автоматического инкрементального обновления для HMS не рекомендуется. Конфигурация по умолчанию рекомендуется в большинстве случаев.

Производительность получения метаданных во многом зависит от производительности HMS пользователя или HDFS NameNode. Пожалуйста, учитывайте все факторы и основывайте своё решение на результатах тестирования.

- **[По умолчанию и рекомендуется] Лучшая производительность с допуском несогласованности данных на уровне минут**
  - **Конфигурация**: Вы можете использовать настройку по умолчанию. Данные, обновлённые в течение 10 минут (по умолчанию), не видны. В течение этого периода запросы будут возвращать старые данные.
  - **Преимущество**: Лучшая производительность запросов.
  - **Недостаток**: Несогласованность данных из-за задержки.
  - **Поддерживаемые версии**: v1.5.2+ (отключено по умолчанию в v1.5.2 и включено по умолчанию в v1.5.2+)
- **Мгновенная видимость недавно загруженных данных (файлов) без ручного обновления**
  - **Конфигурация**: Отключите кэш для метаданных базовых файлов данных, установив свойство catalog `enable_remote_file_cache` в `false`.
  - **Преимущество**: Видимость изменений файлов без задержки.
  - **Недостаток**: Более низкая производительность при отключённом кэше метаданных файлов. Каждый запрос должен обращаться к списку файлов.
  - **Поддерживаемые версии**: v1.5.2+
- **Мгновенная видимость изменений partition без ручного обновления**

- **Конфигурация**: Отключите кэш для имён partition Hive, установив свойство catalog `enable_cache_list_names` в `false`.
- **Преимущество**: Видимость изменений partition без задержки
- **Недостаток**: Более низкая производительность при отключённом кэше имён partition. Каждый запрос должен обращаться к списку partition.
- **Поддерживаемые версии**: v1.5.2+

> **ПОДСКАЗКА**
>
> Если вы требуете обновлений в реальном времени об изменениях данных, а производительность вашего HMS не оптимизирована, вы можете включить кэш, отключить автоматическое инкрементальное обновление и вручную обновлять метаданные (используя REFRESH EXTERNAL TABLE) через систему планирования всякий раз, когда происходит изменение данных upstream.

Система хранения

Функция	Описание	Поддерживаемые версии
Рекурсивный листинг поддиректорий	Включите рекурсивный листинг поддиректорий, установив свойство catalog `enable_recursive_listing` в `true`. Когда рекурсивный листинг включён, Selena будет читать данные из таблицы и её partition, а также из поддиректорий в физических расположениях таблицы и её partition. Эта функция предназначена для решения проблемы многоуровневых вложенных директорий.	v1.5.2+

Форматы файлов и типы данных

Форматы файлов

Функция	Поддерживаемые форматы файлов
Чтение	Parquet, ORC, TEXT, Avro, RCFile, SequenceFile
Запись	Parquet (v1.5.2+), ORC (v1.5.2+), TEXT (v1.5.2+)

Типы данных

Типы INTERVAL, BINARY и UNION не поддерживаются.

Таблицы Hive в формате TEXT не поддерживают типы MAP и STRUCT.

Типы таблиц

Чтение транзакционных таблиц Hive не поддерживается.

Hive views

Selena поддерживает запросы к Hive views начиная с версии v1.5.2.

> **ПРИ ВЫПОЛНЕНИИ ЗАПРОСОВ К HIVE VIEW SELENA ПОПЫТАЕТСЯ РАЗОБРАТЬ ОПРЕДЕЛЕНИЕ VIEW, ИСПОЛЬЗУЯ СИНТАКСИС SELENA И TRINO. ОШИБКА БУДЕТ ВОЗВРАЩЕНА, ЕСЛИ SELENA НЕ МОЖЕТ РАЗОБРАТЬ ОПРЕДЕЛЕНИЕ VIEW. СУЩЕСТВУЕТ ВЕРОЯТНОСТЬ, ЧТО SELENA НЕ СМОЖЕТ РАЗОБРАТЬ HIVE VIEWS, СОЗДАННЫЕ С ФУНКЦИЯМИ, ЭКСКЛЮЗИВНЫМИ ДЛЯ HIVE ИЛИ SPARK.**

Интерфейсы статистики запросов

Функция	Поддерживаемые версии
Поддерживает SHOW CREATE TABLE для просмотра схемы таблицы Hive	v1.5.2+
Поддерживает ANALYZE для сбора статистики	v1.5.2+
Поддерживает сбор гистограмм и статистики подполей STRUCT	v1.5.2+

Запись данных

Функция	Поддерживаемые версии	Примечание
CREATE DATABASE	v1.5.2+	Вы можете выбрать, указывать ли расположение для базы данных, созданной в Hive, или нет. Если вы не укажете расположение для базы данных, вам нужно будет указать расположение для таблиц, созданных в этой базе данных. В противном случае будет возвращена ошибка. Если вы указали расположение для базы данных, таблицы без указанного расположения унаследуют расположение базы данных. И если вы указали расположения как для базы данных, так и для таблицы, расположение таблицы будет иметь приоритет.
CREATE TABLE	v1.5.2+	Для партиционированных и непартиционированных таблиц.
CREATE TABLE AS	v1.5.2+	

Функция	Поддерживаемые версии	Примечание
SELECT		
INSERT INTO/OVERWRITE	v1.5.2+	Для партиционированных и непартиционированных таблиц.
CREATE TABLE LIKE	v1.5.2+	
Размер файла при записи	v1.5.2+	Вы можете определить максимальный размер каждого записываемого файла данных с помощью сессионной переменной `connector_sink_target_max_file_size`.

5.12.3 Iceberg Catalog

Метаданные

При выполнении запросов к данным Iceberg через Iceberg catalog Selena будет кэшировать метаданные таблиц для снижения затрат на частый доступ к удалённому хранилищу. Этот механизм обеспечивает производительность запросов, поддерживая свежесть данных через политику асинхронного обновления и истечения срока действия.

Кэшируемые метаданные

Selena будет кэшировать следующие метаданные во время запросов:

- **Кэш указателя метаданных**
  - Содержимое: JSON-файл указателя метаданных
    - Snapshot ID

        - Расположение списков манифестов
    - Влияние: обнаружение изменений данных (если происходит изменение данных, Snapshot ID изменится.)
    - Свойства catalog:
        - `enable_iceberg_metadata_cache`: Управляет включением кэша метаданных Iceberg. Значение по умолчанию: `true`.
        - `iceberg_table_cache_refresh_interval_sec`: Управляет интервалом времени, в течение которого кэшированные метаданные считаются свежими. Значение по умолчанию: `60`. Единица измерения: секунды.
- **Кэш метаданных**
    - Содержимое:
        - Манифест для пути к файлу данных
        - Манифест для пути к файлу удаления
        - База данных
        - Partition (для перезаписи материализованного представления)
    - Влияние:
        - Не влияет на свежесть данных для запросов, поскольку манифесты для файлов данных или удаления не могут быть изменены.
        - Может влиять на перезапись материализованного представления, заставляя запросы пропускать материализованное представление. Метаданные partition будут удалены при обновлении Snapshot ID. Поэтому новый Snapshot ID не имеет метаданных partition, что приводит к пропуску перезаписи материализованного представления.
    - Свойства catalog:
        - `enable_cache_list_names`: Управляет включением кэша списка имён partition. Значение по умолчанию: `true`.
        - `metastore_cache_refresh_interval_sec`: Управляет интервалом времени, в течение которого кэшированные метаданные считаются свежими. Значение по умолчанию: `60`. Единица измерения: секунды.
        - `iceberg_data_file_cache_memory_usage_ratio`: Управляет долей памяти, которая может использоваться для кэша метаданных файлов данных. Значение по умолчанию: `0.1` (10%).
        - `iceberg_delete_file_cache_memory_usage_ratio`: Управляет долей памяти, которая может использоваться для кэша метаданных файлов удаления. Значение по умолчанию: `0.1` (10%).

Политика асинхронного обновления

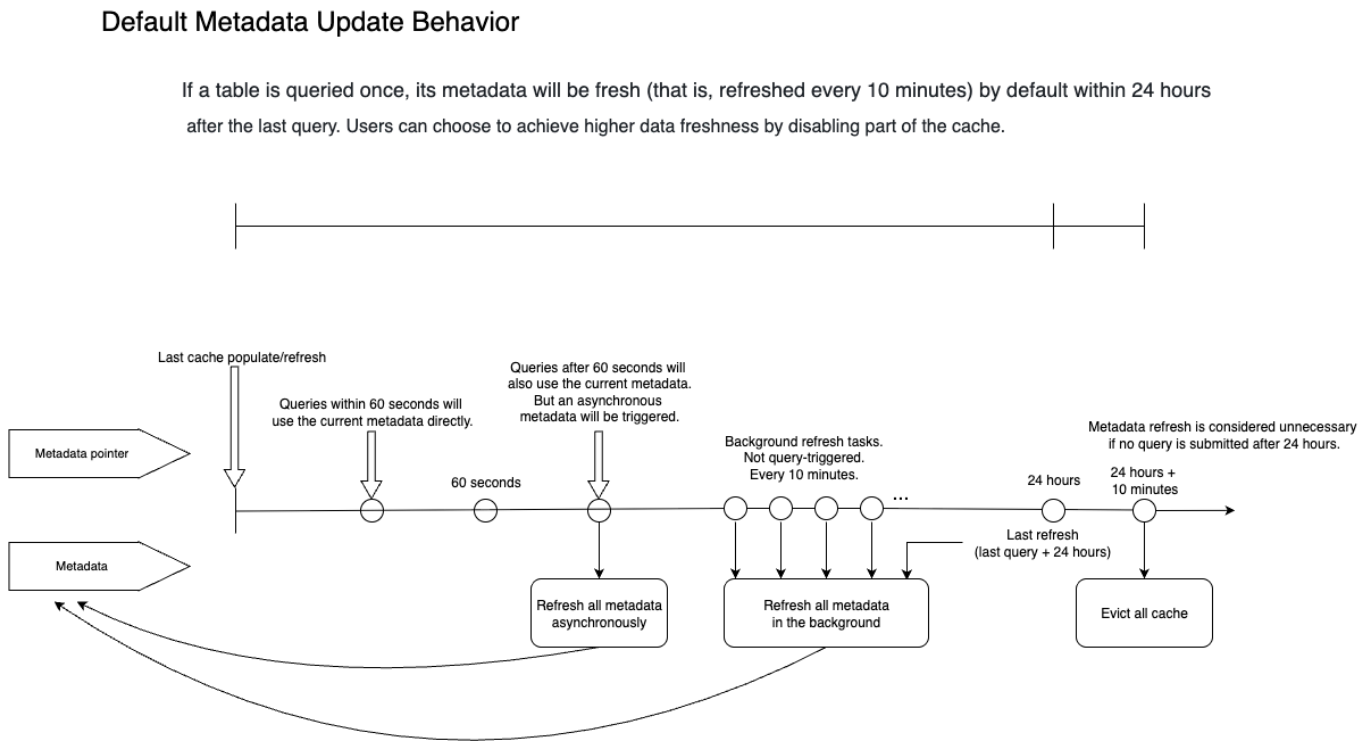
Следующий параметр конфигурации FE управляет политикой асинхронного обновления метаданных:

Параметр конфигурации	Значение по умолчанию	Описание
enable_background_refresh_connector_metadata	`true` в v1.5.2 `false` в v1.5.2	Включать ли периодическое обновление кэша метаданных. После включения Selena опрашивает metastore и обновляет кэшированные метаданные часто используемых внешних catalog для обнаружения изменений данных. `true` означает включить обновление кэша метаданных Hive, `false` означает отключить. Это динамический параметр FE. Вы можете изменить его с помощью команды ADMIN SET FRONTEND CONFIG.
background_refresh_metadata_interval_millis	`600000` (10 минут)	Интервал между двумя последовательными обновлениями кэша метаданных. Единица измерения: миллисекунды. Это динамический параметр FE. Вы можете изменить его с помощью команды ADMIN SET FRONTEND CONFIG.
background_refresh_metadata_time_secs_since_last_access_secs	`86400` (24 часа)	Время истечения срока действия задачи обновления кэша метаданных. Для внешнего catalog, к которому был доступ, если к нему не обращались более указанного времени, Selena прекращает обновление его кэшированных метаданных. Для внешнего catalog, к которому не было доступа, Selena не будет обновлять его кэшированные метаданные. Единица измерения: секунды. Это динамический параметр FE. Вы можете изменить его с помощью команды ADMIN SET FRONTEND CONFIG.

Поведение кэша метаданных

В этом разделе используется поведение по умолчанию для объяснения поведения метаданных во время обновления метаданных и запросов.

По умолчанию, когда выполняется запрос к таблице, Selena кэширует метаданные таблицы и поддерживает их активными в течение следующих 24 часов. В течение 24 часов система обеспечит обновление кэша не реже, чем каждые 10 минут (обратите внимание, что 10 минут — это расчётное время для раунда обновления метаданных. Если есть слишком много внешних таблиц, ожидающих обновления метаданных, общий интервал обновления метаданных может быть больше 10 минут). Если к таблице не обращались более 24 часов, Selena удаляет связанные метаданные. Другими словами, любой запрос, который вы делаете в течение 24 часов, в худшем случае будет использовать метаданные 10-минутной давности.



Подробнее:

1. Предположим, первый запрос затрагивает таблицу `A`. Selena кэширует её последний snapshot и метаданные. Кэш заполняется синхронно во время выполнения запроса.
2. Если второй запрос отправлен в течение 60 секунд после заполнения кэша и затрагивает таблицу `A`, Selena использует кэш метаданных напрямую, и в этот момент Selena считает все кэшированные метаданные свежими (`iceberg_table_cache_refresh_interval_sec` управляет временным окном, в течение которого Selena считает метаданные свежими).
3. Если третий запрос отправлен через 90 секунд и затрагивает таблицу `A`, Selena всё равно будет использовать кэш метаданных напрямую для завершения запроса. Однако, поскольку прошло более 60 секунд с момента последнего обновления метаданных, Selena будет считать метаданные устаревшими. Поэтому Selena запустит асинхронное обновление для устаревших метаданных. Асинхронное обновление не повлияет на результат текущего запроса, потому что запрос всё ещё будет использовать устаревшие метаданные.
4. Поскольку был выполнен запрос к таблице `A`, ожидается, что метаданные будут обновляться каждые 10 минут (управляется `background_refresh_metadata_interval_millis`) в течение следующих 24 часов (управляется `background_refresh_metadata_time_secs_since_last_access_secs`).

Фактический интервал между раундами обновления метаданных также зависит от общего количества ожидающих задач обновления в системе.

5. Если таблица `A` не участвует ни в каком запросе в течение 24 часов, Selena удалит её кэш метаданных через 24 часа.

Лучшие практики

Iceberg Catalog поддерживает HMS, Glue и Tabular в качестве metastore. Конфигурация по умолчанию рекомендуется в большинстве случаев.

Обратите внимание, что значение по умолчанию сессионной переменной `enable_iceberg_metadata_cache` было изменено для адаптации к различным сценариям:

- С версии v1.5.2 по v1.5.2 этот параметр по умолчанию установлен в `true`, независимо от используемого сервиса metastore.
- В версии v1.5.2 и позже, если Iceberg cluster использует AWS Glue в качестве metastore, этот параметр по-прежнему по умолчанию `true`. Однако, если Iceberg cluster использует другие сервисы metastore, такие как Hive metastore, этот параметр по умолчанию `false`.
- С версии v1.5.2 значение этого параметра по умолчанию снова установлено в `true`, поскольку Selena поддерживает новый фреймворк метаданных Iceberg. Iceberg Catalog и Hive Catalog теперь используют один и тот же механизм опроса метаданных и параметр конфигурации FE `background_refresh_metadata_interval_millis`.

Функция	Поддерживаемые версии
Распределённый план метаданных (рекомендуется для сценариев с большим объёмом метаданных)	v1.5.2+
Кэш манифестов (рекомендуется для сценариев с небольшим объёмом метаданных, но высокими требованиями к задержке)	v1.5.2+

Начиная с версии v1.5.2, Selena поддерживает политики чтения и кэширования метаданных, описанные выше. Система автоматически настроит выбор политики в соответствии с машинами в вашем cluster. Обычно вам не нужно его менять. Поскольку кэширование метаданных включено, возможно, что свежесть метаданных может быть скомпрометирована из соображений производительности. Поэтому вы можете настроить его в соответствии с вашими конкретными требованиями к запросам:

- **[По умолчанию и рекомендуется] Оптимальная производительность с допуском несогласованности данных на уровне минут**
  - **Настройка**: Дополнительная настройка не требуется. По умолчанию данные, обновлённые в течение 10 минут, не видны. В течение этого времени запросы будут возвращать старые данные.
  - **Преимущества**: Лучшая производительность запросов.
  - **Недостаток**: несогласованность данных из-за задержек.
- **Новые файлы данных, сгенерированные импортом, и добавления или удаления partition видны немедленно, ручное обновление не требуется**
  - **Настройка**: Установите свойство catalog `iceberg_table_cache_ttl_sec` в `0`, чтобы позволить Selena получать новый snapshot для каждого запроса.
  - **Преимущества**: Изменения файлов и partition видны без задержки.
  - **Недостаток**: Более низкая производительность из-за поведения получения snapshot для каждого запроса.

Форматы файлов

Функция	Поддерживаемые форматы файлов
Чтение	Parquet, ORC
Запись	Parquet

- Таблицы Iceberg V1 в форматах Parquet и ORC поддерживают position deletes и equality deletes.
- Таблицы Iceberg V2 в формате ORC поддерживают position deletes с версии v1.5.2, а в формате Parquet — с версии v1.5.2.
- Таблицы Iceberg V2 в формате ORC поддерживают equality deletes с v1.5.2.

Iceberg views

Selena поддерживает запросы к Iceberg views начиная с версии v1.5.2 и создание Iceberg views с версии v1.5.2.

В настоящее время поддерживаются только Iceberg views, созданные через Selena. Начиная с версии v1.5.2, поддерживается добавление определений в стиле синтаксиса Selena к существующим Iceberg views.

Интерфейсы статистики запросов

Функция	Поддерживаемые версии
Поддерживает SHOW CREATE TABLE для просмотра схемы таблицы Iceberg	v1.5.2+
Поддерживает ANALYZE для сбора статистики	v1.5.2+
Поддерживает сбор гистограмм и статистики подполей STRUCT	v1.5.2+

Запись данных

Функция	Поддерживаемые версии	Примечание
CREATE DATABASE	v1.5.2+	Вы можете выбрать, указывать ли расположение для базы данных, созданной в Iceberg, или нет. Если вы не укажете расположение для базы данных, вам нужно будет указать расположение для таблиц, созданных в этой базе данных. В противном случае будет возвращена ошибка. Если вы указали расположение для базы данных, таблицы без указанного расположения унаследуют расположение базы данных. И если вы указали расположения как для базы данных, так и для таблицы, расположение таблицы будет иметь приоритет.
CREATE TABLE	v1.5.2+	Поддерживает партиционированные и непартиционированные таблицы. Начиная с v2.0.0, поддерживает создание таблиц со скрытыми partition.
CREATE TABLE AS SELECT	v1.5.2+	
INSERT INTO/OVERWRITE	v1.5.2+	Для партиционированных и непартиционированных таблиц.

Прочая поддержка

Функция	Поддерживаемые версии
Поддерживает чтение форматов partition типа TIMESTAMP `yyyy-MM-ddTHH:mm` и `yyyy-MM-dd HH:mm`.	v1.5.2+
Поддерживает таблицу метаданных Iceberg	v1.5.2+
Поддерживает Iceberg TimeTravel	v1.5.2+

5.12.4 Hudi Catalog

- Selena поддерживает запросы к данным в формате Parquet в Hudi и поддерживает форматы сжатия SNAPPY, LZ4, ZSTD, GZIP и NO_COMPRESSION для файлов Parquet.
- Selena полностью поддерживает таблицы Hudi Copy On Write (COW) и Merge On Read (MOR).
- Selena поддерживает SHOW CREATE TABLE для просмотра схемы таблицы Hudi начиная с версии v1.5.2.
- Selena v1.5.2 поддерживает Hudi 0.15.0.

5.12.5 Delta Lake Catalog

- Selena поддерживает запросы к данным в формате Parquet в Delta Lake и поддерживает форматы сжатия SNAPPY, LZ4, ZSTD, GZIP и NO_COMPRESSION для файлов Parquet.
- Selena не поддерживает запросы к данным типов MAP и STRUCT в Delta Lake.
- Selena поддерживает SHOW CREATE TABLE для просмотра схемы таблицы Delta Lake начиная с версии v1.5.2.
- В настоящее время Delta Lake catalog поддерживают следующие функции таблиц:
  - V2 Checkpoint (с версии v1.5.2)
  - Timestamp without Timezone (с версии v1.5.2)
  - Column mapping (с версии v1.5.2)
  - Deletion Vector (с версии v1.5.2)

5.12.6 JDBC Catalog

Тип catalog	Поддерживаемые версии
MySQL	v1.5.2+
PostgreSQL	v1.5.2+

Тип catalog	Поддерживаемые версии
ClickHouse	v1.5.2+
Oracle	v1.5.2+
SQL Server	v1.5.2+

MySQL

Функция	Поддерживаемые версии
Кэш метаданных	v1.5.2+

Соответствие типов данных

MySQL	Selena	Поддерживаемые версии
BOOLEAN	BOOLEAN	v1.5.2+
BIT	BOOLEAN	v1.5.2+
SIGNED TINYINT	TINYINT	v1.5.2+
UNSIGNED TINYINT	SMALLINT	v1.5.2+
SIGNED SMALLINT	SMALLINT	v1.5.2+
UNSIGNED SMALLINT	INT	v1.5.2+
SIGNED INTEGER	INT	v1.5.2+
UNSIGNED INTEGER	BIGINT	v1.5.2+
SIGNED BIGINT	BIGINT	v1.5.2+
UNSIGNED BIGINT	LARGEINT	v1.5.2+
FLOAT	FLOAT	v1.5.2+
REAL	FLOAT	v1.5.2+
DOUBLE	DOUBLE	v1.5.2+
DECIMAL	DECIMAL32	v1.5.2+
CHAR	VARCHAR(columnsize)	v1.5.2+
VARCHAR	VARCHAR	v1.5.2+
TEXT	VARCHAR(columnsize)	v1.5.2+
DATE	DATE	v1.5.2+
TIME	TIME	v1.5.2+
TIMESTAMP	DATETIME	v1.5.2+

© ООО «Платформа Данных Селена», 2026

PostgreSQL

Соответствие типов данных

PGSQL	Selena	Поддерживаемые версии
BIT	BOOLEAN	v1.5.2+
SMALLINT	SMALLINT	v1.5.2+
INTEGER	INT	v1.5.2+
BIGINT	BIGINT	v1.5.2+
REAL	FLOAT	v1.5.2+
DOUBLE	DOUBLE	v1.5.2+
NUMERIC	DECIMAL32	v1.5.2+
CHAR	VARCHAR(columnsize)	v1.5.2+
VARCHAR	VARCHAR	v1.5.2+
TEXT	VARCHAR(columnsize)	v1.5.2+
DATE	DATE	v1.5.2+
TIMESTAMP	DATETIME	v1.5.2+
UUID	VARBINARY	v1.5.2+

ClickHouse

Поддерживается с версии v1.5.2.

Oracle

Поддерживается с версии v1.5.2.

SQL Server

Поддерживается с версии v1.5.2.

5.12.7 Elasticsearch Catalog

Elasticsearch Catalog поддерживается с версии v1.5.2.

5.12.8 Paimon Catalog

Paimon Catalog поддерживается с версии v1.5.2.

5.12.9 MaxCompute Catalog

MaxCompute Catalog поддерживается с версии v1.5.2.

5.12.10 Kudu Catalog

Kudu Catalog поддерживается с версии v1.5.2.

6 Ускорение запросов

6.1 Просмотр выполняющихся запросов

В этой статье описывается, как просматривать выполняющиеся запросы в Selena и анализировать потребление ими ресурсов.

6.1.1 Current_queries

`show proc '/current_queries'` позволяет просматривать запросы, выполняющиеся на текущем узле FE, включая следующую информацию:

Имя столбца	Описание
StartTime	Время начала запроса
feIp	IP-адрес узла FE, выполняющего запрос
QueryId	Идентификатор запроса
ConnectionId	Идентификатор соединения
Database	База данных, в которой выполняется запрос
User	Пользователь, выполняющий запрос
ScanBytes	Объём просканированных данных
ScanRows	Количество просканированных строк
MemoryUsage	Объём памяти, используемой запросом
DiskSpillSize	Объём данных, сброшенных на диск
CPUTime	Процессорное время, потреблённое запросом
ExecTime	Время выполнения запроса
Warehouse	Warehouse, используемый запросом
CustomQueryId	Пользовательский идентификатор запроса
ResourceGroup	Группа ресурсов, используемая запросом

Пример:

```
MySQL > show proc '/current_queries'\G
***************************[ 1. row ]***************************
StartTime     | 2025-03-07 02:16:04
feIp          | 172.26.92.227
QueryId       | 10db481c-fab7-11ef-8063-461f20abc3f0
ConnectionId  | 13
Database      | tpcds_2
User          | root
ScanBytes     | 120.573 MB
ScanRows      | 5859503 rows
MemoryUsage   | 225.893 MB
DiskSpillSize | 0.000 B
CPUTime       | 47.878 s
ExecTime      | 4.077 s
Warehouse     | default_warehouse
CustomQueryId |
ResourceGroup | rg1
```

6.1.2 Global_current_queries

Аналогично `current_queries`, `show proc '/global_current_queries'` показывает информацию о запросах, выполняющихся на всех узлах FE. Эта команда поддерживается начиная с версии 3.4.

Пример:

```
MySQL root@127.1:(none)> show proc '/global_current_queries'\G
***************************[ 1. row ]***************************
StartTime     | 2025-03-07 02:21:48
feIp          | 172.26.92.227
QueryId       | de516505-fab7-11ef-8063-461f20abc3f0
ConnectionId  | 14
Database      | tpcds_2
User          | root
ScanBytes     | 120.573 MB
ScanRows      | 5859503 rows
MemoryUsage   | 346.915 MB
DiskSpillSize | 0.000 B
CPUTime       | 33.265 s
ExecTime      | 3.032 s
Warehouse     | default_warehouse
CustomQueryId |
ResourceGroup | rg1
```

6.1.3 Running queries

`SHOW RUNNING QUERIES` в основном используется для просмотра статуса очереди запросов. Если запрос находится в очереди, он будет в состоянии `PENDING`.

Имя поля	Описание
QueryId	Идентификатор запроса
ResourceGroupId	Идентификатор группы ресурсов
StartTime	Время начала запроса
PendingTimeout	Таймаут ожидания
QueryTimeout	Таймаут запроса
State	Состояние запроса, включая PENDING и RUNNING
Slots	Количество занятых слотов
Fragments	Количество фрагментов в плане запроса
DOP	Степень параллелизма
Frontend	Информация об узле FE
FeStartTime	Время запуска FE

Пример:

```
MySQL root@127.1:(none)> show running queries\G
***************************[ 1. row ]***************************
QueryId         | 50029ec1-fab8-11ef-8063-461f20abc3f0
ResourceGroupId | 562275
StartTime       | 2025-03-07 02:24:59
PendingTimeout  | 2025-03-07 02:27:29
QueryTimeout    | 2025-03-07 02:27:29
State           | RUNNING
Slots           | 1
Fragments       | 11
DOP             | 0
Frontend        | 172.26.92.227_8034_1709578860161
FeStartTime     | 2025-03-06 23:39:00
```

6.2 Сбор статистики для CBO

В этой теме описывается базовая концепция оптимизатора на основе стоимости (CBO) Selena и как собирать статистику для CBO для выбора оптимального плана запроса. Selena 2.4 вводит гистограммы для сбора точной статистики распределения данных.

Начиная с версии 3.2.0, Selena поддерживает сбор статистики из таблиц Hive, Iceberg и Hudi, уменьшая зависимость от других систем metastore. Синтаксис аналогичен сбору статистики для внутренних таблиц Selena.

6.2.1 Что такое CBO

CBO критически важен для оптимизации запросов. После того как SQL-запрос поступает в Selena, он парсится в логический план выполнения. CBO переписывает и преобразует логический план в несколько физических планов выполнения. Затем CBO оценивает стоимость выполнения каждого оператора в плане (например, CPU, память, сеть и I/O) и выбирает путь запроса с наименьшей стоимостью в качестве финального физического плана.

Selena CBO был запущен в Selena 1.16.0 и включен по умолчанию начиная с версии 1.19. Разработанный на основе фреймворка Cascades, Selena CBO оценивает стоимость на основе различной статистической информации. Он способен выбрать план выполнения с наименьшей стоимостью среди десятков тысяч планов выполнения, значительно повышая эффективность и производительность сложных запросов.

Статистика важна для CBO. Она определяет, является ли оценка стоимости точной и полезной. Следующие разделы подробно описывают типы статистической информации, политику сбора, а также как собирать статистику и просматривать статистическую информацию.

6.2.2 Типы статистической информации

Selena собирает различную статистику в качестве входных данных для оценки стоимости.

Базовая статистика

По умолчанию Selena периодически собирает следующую базовую статистику таблиц и столбцов:

- row_count: общее количество строк в таблице
- data_size: размер данных столбца
- ndv: мощность столбца, то есть количество различных значений в столбце
- null_count: количество данных со значениями NULL в столбце
- min: минимальное значение в столбце
- max: максимальное значение в столбце

Полная статистика хранится в таблице `column_statistics` базы данных `_statistics_`. Вы можете просмотреть эту таблицу для запроса статистики таблиц. Ниже приведен пример запроса статистических данных из этой таблицы.

```
SELECT * FROM _statistics_.column_statistics\G
*************************** 1. row ***************************
      table_id: 10174
  partition_id: 10170
   column_name: is_minor
         db_id: 10169
    table_name: zj_test.source_wiki_edit
partition_name: p06
     row_count: 2
     data_size: 2
           ndv: NULL
    null_count: 0
           max: 1
           min: 0
```

Гистограмма

Selena 2.4 вводит гистограммы для дополнения базовой статистики. Гистограмма считается эффективным способом представления данных. Для таблиц со смещенными данными гистограммы могут точно отражать распределение данных.

Selena использует гистограммы равной высоты, которые построены на нескольких корзинах (buckets). Каждая корзина содержит одинаковое количество данных. Для значений данных, которые часто запрашиваются и оказывают значительное влияние на селективность, Selena выделяет отдельные корзины. Больше корзин означает более точную оценку, но может также вызвать небольшое увеличение использования памяти. Вы можете настроить количество корзин и наиболее частых значений (MCVs) для задачи сбора гистограмм.

Гистограммы применимы к столбцам с сильно смещенными данными и частыми запросами. Если данные вашей таблицы равномерно распределены, вам не нужно создавать гистограммы. Гистограммы могут быть созданы только для столбцов числовых типов, DATE, DATETIME или строковых типов.

Гистограммы хранятся в таблице `histogram_statistics` базы данных `_statistics_`. Ниже приведен пример запроса статистических данных из этой таблицы:

```
SELECT * FROM _statistics_.histogram_statistics\G
*************************** 1. row ***************************
   table_id: 10174
column_name: added
      db_id: 10169
 table_name: zj_test.source_wiki_edit
    buckets: NULL
        mcv: [["23","1"],["5","1"]]
update_time: 2023-12-01 15:17:10.274000
```

Многостолбцовая совместная статистика

Начиная с версии 3.5.0, Selena поддерживает сбор многостолбцовой совместной статистики. В настоящее время, когда Selena выполняет оценку мощности, оптимизатор в большинстве сценариев предполагает, что несколько столбцов полностью независимы друг от друга, то есть между ними нет корреляции. Однако, если между столбцами есть корреляция, текущий метод оценки может привести к неправильному результату. Это заставляет оптимизатор генерировать неправильный план выполнения. В настоящее время поддерживается только многостолбцовый совместный NDV, и он в основном используется в следующих сценариях для оценки мощности.

- Оценка нескольких эквивалентных предикатов, соединенных AND.
- Оценка узлов Agg.
- Применение к стратегии проталкивания агрегации.

В настоящее время сбор многостолбцовой совместной статистики поддерживает только ручной сбор. Тип по умолчанию - выборочный сбор. Многостолбцовая статистика хранится в таблице `multi_column_statistics` базы данных `_statistics_` в каждом кластере Selena. Запрос вернет информацию, похожую на следующую:

```
mysql> select * from _statistics_.multi_column_statistics \G
*************************** 1. row ***************************
    table_id: 1695021
  column_ids: 0#1
       db_id: 110099
  table_name: db.test_multi_col_stats
column_names: id,name
         ndv: 11
 update_time: 2025-04-11 15:09:50
```

6.2.3 Типы и методы сбора

Размер данных и распределение данных постоянно меняются в таблице. Статистика должна регулярно обновляться, чтобы отражать эти изменения данных. Перед созданием задачи сбора статистики вы должны выбрать тип и метод сбора, которые лучше всего подходят для ваших бизнес-требований.

Selena поддерживает полный и выборочный сбор, оба могут выполняться автоматически и вручную. По умолчанию Selena автоматически собирает полную статистику таблицы. Она проверяет наличие обновлений данных каждые 5 минут. Если обнаружено изменение данных, сбор данных будет автоматически запущен. Если вы не хотите использовать автоматический полный сбор, вы можете настроить задачу сбора.

Тип сбора	Метод сбора	Описание	Преимущества и недостатки
Полный сбор	Автоматический/ручной	Сканирует всю таблицу для сбора статистики. Статистика собирается по partition. Если в partition нет изменений данных, данные не будут собираться из этого partition, что снижает потребление ресурсов. Полная статистика хранится в таблице `_statistics_.column_statistics`.	Преимущество: Статистика точная, что помогает CBO делать точную оценку. Недостаток: Потребляет системные ресурсы и медленный. Начиная с версии 2.5, Selena позволяет указать период автоматического сбора, что снижает потребление ресурсов.
Выборочный сбор	Автоматический/ручной	Равномерно извлекает `N` строк данных из каждого partition таблицы. Статистика собирается по таблице. Базовая статистика каждого столбца хранится как одна запись. Информация о мощности (ndv) столбца оценивается на основе выборочных данных, что неточно. Выборочная статистика хранится в таблице `_statistics_.table_statistic_v1`.	Преимущество: Потребляет меньше системных ресурсов и быстрый. Недостаток: Статистика неполная, что может повлиять на точность оценки стоимости.

Начиная с версии 3.5.0, как выборочная, так и полная статистика сбора хранится в таблице `_statistics_.column_statistics`. Это связано с тем, что текущий оптимизатор будет фокусироваться на последней собранной статистике при оценке мощности, в то время как каждый раз, когда фоновый автоматический сбор статистики может использовать другой метод сбора из-за различного состояния здоровья таблицы. Если есть смещение данных, частота ошибок выборочной статистической информации полной таблицы может быть выше, и один и тот же запрос может использовать разную статистическую информацию из-за различных методов сбора, что может привести к

генерации оптимизатором неправильного плана выполнения. Поэтому как выборочный сбор, так и полный сбор предполагают, что статистическая информация собирается на уровне partition. Вы можете настроить предыдущий метод сбора и хранения, изменив параметр конфигурации FE `statistic_use_meta_statistics` на `false`.

6.2.4 Predicate Column

Начиная с версии 3.5.0, Selena поддерживает сбор статистики Predicate Column.

Predicate Column - это столбец, который часто используется в качестве условия фильтра (предложение WHERE, условие JOIN, столбец GROUP BY, столбец DISTINCT) в запросе. Selena автоматически записывает каждый Predicate Column для таблицы, участвующей в запросе, и сохраняет его в таблице `_statistics_.predicate_columns`. Запрос вернет следующее:

```
select * from _statistics_.predicate_columns \G
*************************** 1. row ***************************
      fe_id: 127.0.0.1_9011_1735874871718
      db_id: 1684773
   table_id: 1685786
  column_id: 1
      usage: normal,predicate,join,group_by
  last_used: 2025-04-11 20:39:32
    created: 2025-04-11 20:39:53
```

Вы также можете запросить представление `information_schema.column_stats_usage` для получения более наблюдаемой информации. В экстремальных сценариях (с большим количеством столбцов) существует большая нагрузка для полного сбора, тогда как в реальности для стабильной рабочей нагрузки вам часто не нужна статистика для всех столбцов, а только для столбцов, участвующих в некоторых ключевых операциях Filter, Join и Aggregation. Поэтому, чтобы сбалансировать стоимость и точность, Selena поддерживает ручной сбор статистики Predicate Column и сбор статистики Predicate Column в соответствии с политикой при автоматическом сборе статистики, избегая сбора статистики всех столбцов в таблице. Каждый узел FE в кластере синхронно обновляет информацию Predicate Column в кэш FE с регулярными интервалами для ускорения использования.

6.2.5 Сбор статистики

Selena предлагает гибкие методы сбора статистики. Вы можете выбрать автоматический, ручной или пользовательский сбор, в зависимости от того, что подходит для ваших бизнес-сценариев.

По умолчанию Selena автоматически собирает полное количество статистики для таблицы на периодической основе. Интервал по умолчанию составляет 10 минут. Если система обнаруживает, что коэффициент обновления данных соответствует условию, сбор будет запущен автоматически. **Сбор полной статистики может потреблять много системных ресурсов**. Если вы не хотите использовать автоматический полный сбор статистики, вы можете установить параметр конфигурации FE `enable_collect_full_statistic` в `false`, и периодическая задача сбора будет изменена с полного сбора статистики на выборочный сбор.

Начиная с версии 3.5.0, Selena автоматически изменит полный сбор на выборочный сбор, если обнаружит, что данные таблицы сильно изменились с момента последнего сбора до текущей таблицы во время автоматического сбора. В случае использования выборочного сбора, если в таблице есть Predicate Column (PREDICATE/JOIN/GROUP BY/DISTINCT), Selena преобразует задачу выборки в полный сбор и соберет статистику Predicate Column для обеспечения точности статистики и не будет собирать статистику всех столбцов в таблице. Вы можете настроить это с помощью параметра конфигурации FE `statistic_auto_collect_use_full_predicate_column_for_sample`. Кроме того, Selena также переключается с полного сбора всех столбцов на полный сбор Predicate Columns, если количество столбцов в таблице превышает параметр конфигурации FE `statistic_auto_collect_predicate_columns_threshold` при использовании полного сбора.

Автоматический сбор

Для базовой статистики Selena автоматически собирает полную статистику таблицы по умолчанию, не требуя ручных операций. Для таблиц, для которых статистика не была собрана, Selena автоматически собирает статистику в пределах периода планирования. Для таблиц, для которых статистика была собрана, Selena обновляет общее количество строк и измененных строк в таблицах и регулярно сохраняет эту информацию для определения необходимости запуска автоматического сбора. Гистограммы и многостолбцовая совместная статистика не захватываются текущей задачей автоматического сбора.

Начиная с версии 2.4.5, Selena позволяет указать период сбора для автоматического полного сбора, что предотвращает колебания производительности кластера, вызванные автоматическим полным сбором. Этот период указывается параметрами FE `statistic_auto_analyze_start_time` и `statistic_auto_analyze_end_time`.

Условия, которые запустят автоматический сбор:

- Данные таблицы изменились с момента предыдущего сбора статистики.
- Время сбора попадает в диапазон настроенного периода сбора. (Период сбора по умолчанию - весь день.)
- Время обновления предыдущей задачи сбора раньше последнего времени обновления partition.
- Здоровье статистики таблицы ниже указанного порога (`statistic_auto_collect_ratio`).

> Формула для расчета здоровья статистики:
>
> 1. Если количество partition с обновленными данными меньше 10, формула: `1 - (Количество обновленных строк с момента предыдущего сбора/Общее количество строк)`.
> 2. Если количество partition с обновленными данными больше или равно 10, формула: `1 - MIN(Количество обновленных строк с момента предыдущего сбора/Общее количество строк, Количество обновленных partition с момента предыдущего сбора/Общее количество partition)`.
> 3. Начиная с версии 3.5.0, чтобы определить, является ли partition здоровым или нет, Selena больше не сравнивает время сбора статистической информации со временем обновления данных, а скорее соотношение обновленных строк partition. Вы можете настроить это через параметр конфигурации FE `statistic_partition_health_v2_threshold`. Также вы можете установить параметр конфигурации FE `statistic_partition_healthy_v2` в `false`, чтобы использовать предыдущее поведение проверки здоровья.

Кроме того, Selena позволяет настраивать политики сбора на основе размера таблицы и частоты обновления таблицы:

- Для таблиц с небольшим объемом данных **статистика собирается в реальном времени без ограничений, даже если данные таблицы часто обновляются. Параметр `statistic_auto_collect_small_table_size` можно использовать для определения, является ли таблица малой или большой. Вы также можете использовать `statistic_auto_collect_small_table_interval` для настройки интервалов сбора для малых таблиц.

- Для таблиц с большим объемом данных применяются следующие ограничения:

  - Интервал сбора по умолчанию составляет не менее 12 часов, который можно настроить с помощью `statistic_auto_collect_large_table_interval`.

  - Когда интервал сбора выполнен и здоровье статистики ниже порога для автоматического выборочного сбора, запускается выборочный сбор. Вы можете настроить это поведение с помощью параметра конфигурации FE `statistic_auto_collect_sample_threshold`. Начиная с версии 3.5.0, выборочный сбор преобразуется в полный сбор Predicate Columns, если выполнены все следующие условия:

    - Predicate Column существует в таблице, и количество Predicate Columns меньше `statistic_auto_collect_max_predicate_column_size_on_sample_strategy`.
    - Параметр конфигурации FE `statistic_auto_collect_use_full_predicate_column_for_sample` установлен в `true`.
  - Когда интервал сбора выполнен и здоровье статистики выше порога для автоматического выборочного сбора (`statistic_auto_collect_sample_threshold`) и ниже порога автоматического сбора (`statistic_auto_collect_ratio`), запускается полный сбор.

  - Когда размер partition для сбора данных (`statistic_max_full_collect_data_size`) больше 100 ГБ, запускается выборочный сбор.

  - Собирается только статистика partition, время обновления которых позже времени предыдущей задачи сбора. Статистика partition без изменения данных не собирается.

- Для таблиц, которые имеют Predicate Column и общее количество столбцов превышает `statistic_auto_collect_predicate_columns_threshold`, статистика собирается только для Predicate Column в таблице.

> **ПОДСКАЗКА**
>
> После изменения данных таблицы ручной запуск задачи выборочного сбора для этой таблицы сделает время обновления задачи выборочного сбора позже времени обновления данных, что не запустит автоматический полный сбор для этой таблицы в этом периоде планирования.

Автоматический полный сбор включен по умолчанию и выполняется системой с использованием настроек по умолчанию.

В следующей таблице описаны настройки по умолчанию. Если вам нужно изменить их, выполните команду **ADMIN SET CONFIG**.

Параметр конфигурации FE	Тип	Значение по умолчанию	Описание
enable_statistic_collect	BOOLEAN	TRUE	Включить ли задачи автоматического сбора по умолчанию и пользовательские. Этот переключатель включен по умолчанию.
enable_collect_full_statistic	BOOLEAN	TRUE	Включить ли автоматический сбор по умолчанию. Этот переключатель включен по умолчанию.
statistic_collect_interval_sec	LONG	600	Интервал для проверки обновлений данных во время автоматического сбора. Единица: секунды.
statistic_auto_analyze_start_time	STRING	00:00:00	Время начала автоматического сбора. Диапазон значений: `00:00:00` - `23:59:59`.
statistic_auto_analyze_end_time	STRING	23:59:59	Время окончания автоматического сбора. Диапазон значений: `00:00:00` - `23:59:59`.

Параметр конфигурации FE	Тип	Значение по умолчанию	Описание
statistic_auto_collect_small_table_size	LONG	5368709120	Порог для определения, является ли таблица малой для автоматического полного сбора. Таблица, размер которой больше этого значения, считается большой таблицей, тогда как таблица, размер которой меньше или равен этому значению, считается малой таблицей. Единица: байт. Значение по умолчанию: 5368709120 (5 ГБ).
statistic_auto_collect_small_table_interval	LONG	0	Интервал для автоматического сбора полной статистики малых таблиц. Единица: секунды.
statistic_auto_collect_large_table_interval	LONG	43200	Интервал для автоматического сбора полной статистики больших таблиц. Единица: секунды. Значение по умолчанию: 43200 (12 часов).
statistic_auto_collect_ratio	FLOAT	0.8	Порог для определения, является ли статистика для автоматического сбора здоровой. Если здоровье статистики ниже этого порога, запускается автоматический сбор.
statistic_auto_collect_sample_threshold	DOUBLE	0.3	Порог здоровья статистики для запуска автоматического выборочного сбора. Если значение здоровья статистики ниже этого порога, запускается автоматический выборочный сбор.
statistic_max_full_collect_data_size	LONG	107374182400	Размер данных partition для автоматического сбора для сбора данных. Единица: байт. Значение по умолчанию: 107374182400 (100 ГБ). Если размер данных превышает это значение, полный сбор отменяется и выполняется выборочный сбор.
statistic_full_collect_buffer	LONG	20971520	Максимальный размер буфера, занимаемый задачами автоматического сбора. Единица: байт. Значение по умолчанию: 20971520 (20 МБ).
statistic_collect_max_row_count_per_query	INT	5000000000	Максимальное количество строк для запроса для одной задачи analyze. Задача analyze будет разделена на несколько запросов, если это значение превышено.
statistic_collect_too_many_version_sleep	LONG	600000	Время сна задач автоматического сбора, если таблица, на которой выполняется задача сбора, имеет слишком много версий данных. Единица: мс. Значение по умолчанию: 600000 (10 минут).
statistic_auto_collect_use_full_predicate_column_for_sample	BOOLEAN	TRUE	Преобразовывать ли задачу автоматического полного сбора в полный сбор Predicate Column, когда она попадает в политику выборочного сбора.
statistic_auto_collect_max_pr	INT	16	Когда задача автоматического полного сбора попадает в политику выборочного сбора, если таблица имеет необычно большое количество Predicate Columns и

Параметр конфигурации FE	Тип	Значение по умолчанию	Описание
edicate_column_size_on_sample_strategy			превышает этот параметр конфигурации, задача не переключится на полный сбор Predicate Columns, а сохранит выборочный сбор всех столбцов. Этот параметр конфигурации контролирует максимальное значение Predicate Column для этого поведения.
statistic_auto_collect_predicate_columns_threshold	INT	32	Если количество столбцов в таблице превышает эту конфигурацию во время автоматического сбора, будет собираться только статистика столбцов для Predicate Column.
statistic_predicate_columns_persist_interval_sec	LONG	60	Интервал, с которым FE синхронизирует и сохраняет статистику Predicate Column.
statistic_predicate_columns_ttl_hours	LONG	24	Время удаления статистики Predicate Column, кэшированной в FE.
enable_predicate_columns_collection	BOOLEAN	TRUE	Включить ли сбор predicate columns. Если отключено, predicate columns не будут записываться во время оптимизации запросов.
enable_manual_collect_array_ndv	BOOLEAN	FALSE	Включить ли ручной сбор для информации NDV типа ARRAY.
enable_auto_collect_array_ndv	BOOLEAN	FALSE	Включить ли автоматический сбор для информации NDV типа ARRAY.

Вы можете полагаться на автоматические задачи для большинства задач сбора статистики, но если у вас есть специфические требования, вы можете вручную создать задачу, выполнив оператор ANALYZE TABLE, или настроить автоматическую задачу, выполнив оператор CREATE ANALYZE.

Ручной сбор

Вы можете использовать ANALYZE TABLE для создания задачи ручного сбора. По умолчанию ручной сбор - это синхронная операция. Вы также можете установить ее как асинхронную операцию. В асинхронном режиме после выполнения ANALYZE TABLE система немедленно возвращает, была ли эта инструкция успешной. Однако задача сбора будет выполняться в фоновом режиме, и вам не нужно ждать результата. Вы можете проверить статус задачи, выполнив SHOW ANALYZE STATUS. Асинхронный сбор подходит для таблиц с большим объемом данных, тогда как синхронный сбор подходит для таблиц с небольшим объемом данных. **Задачи ручного сбора выполняются только один раз после создания. Вам не нужно удалять задачи ручного сбора.** Вы должны иметь привилегии INSERT и SELECT для соответствующей таблицы для выполнения операции ANALYZE TABLE.

Ручной сбор базовой статистики

```
ANALYZE [FULL|SAMPLE] TABLE tbl_name
    [( col_name [, col_name]... )
    | col_name [, col_name]...
    | ALL COLUMNS
    | PREDICATE COLUMNS
    | MULTIPLE COLUMNS ( col_name [, col_name]... )]
[PARTITION (partition_name [, partition_name]...)]
[WITH [SYNC | ASYNC] MODE]
[PROPERTIES (property [, property]...)]
```

Описание параметров:

- Тип сбора
  - FULL: указывает полный сбор.
  - SAMPLE: указывает выборочный сбор.
  - Если тип сбора не указан, по умолчанию используется полный сбор.
- Тип столбца, из которого собирать статистику:
  - `col_name`: Столбцы, из которых собирать статистику. Разделите несколько столбцов запятыми (`,`). Если этот параметр не указан, собирается вся таблица.
  - `ALL COLUMNS`: Собрать статистику из всех столбцов. Поддерживается с версии 3.5.0.

- `PREDICATE COLUMNS`: Собрать статистику только из Predicate Columns. Поддерживается с версии 3.5.0.
- `MULTIPLE COLUMNS`: Собирает совместную статистику из указанных нескольких столбцов. В настоящее время поддерживается только ручной синхронный сбор нескольких столбцов. Количество столбцов для ручного сбора статистики не может превышать `statistics_max_multi_column_combined_num`, значение по умолчанию - `10`. Поддерживается с версии 3.5.0.

- [WITH SYNC | ASYNC MODE]: выполнять ли задачу ручного сбора в синхронном или асинхронном режиме. Синхронный сбор используется по умолчанию, если вы не указываете этот параметр.

- `PROPERTIES`: пользовательские параметры. Если `PROPERTIES` не указан, используются настройки по умолчанию в файле `fe.conf`. Фактически используемые свойства можно просмотреть в столбце `Properties` в выводе SHOW ANALYZE STATUS.

PROPERTIES	Тип	Значение по умолчанию	Описание
statistic_sample_collect_rows	INT	200000	Минимальное количество строк для сбора для выборочного сбора. Если значение параметра превышает фактическое количество строк в вашей таблице, выполняется полный сбор.

Примеры

Ручной полный сбор

```
-- Вручную собрать полную статистику таблицы, используя настройки по умолчанию.
ANALYZE TABLE tbl_name;

-- Вручную собрать полную статистику таблицы, используя настройки по умолчанию.
ANALYZE FULL TABLE tbl_name;

-- Вручную собрать статистику указанных столбцов в таблице, используя настройки по умолчанию.
ANALYZE TABLE tbl_name(c1, c2, c3);
```

Ручной выборочный сбор

```
-- Вручную собрать частичную статистику таблицы, используя настройки по умолчанию.
ANALYZE SAMPLE TABLE tbl_name;

-- Вручную собрать статистику указанных столбцов в таблице с указанием количества строк для сбора.
ANALYZE SAMPLE TABLE tbl_name (v1, v2, v3) PROPERTIES(
    "statistic_sample_collect_rows" = "1000000"
);
```

- Ручной сбор многостолбцовой совместной статистики

```
-- Ручной выборочный сбор многостолбцовой совместной статистики
ANALYZE SAMPLE TABLE tbl_name MULTIPLE COLUMNS (v1, v2);

-- Ручной полный сбор многостолбцовой совместной статистики
ANALYZE FULL TABLE tbl_name MULTIPLE COLUMNS (v1, v2);
```

- Ручной сбор Predicate Column

```
-- Ручной выборочный сбор Predicate Column
ANALYZE SAMPLE TABLE tbl_name PREDICATE COLUMNS

-- Ручной полный сбор Predicate Column
ANALYZE FULL TABLE tbl_name PREDICATE COLUMNS
```

Ручной сбор гистограмм

```
ANALYZE TABLE tbl_name UPDATE HISTOGRAM ON col_name [, col_name]
[WITH SYNC | ASYNC MODE]
[WITH N BUCKETS]
[PROPERTIES (property [,property])]
```

Описание параметров:

- `col_name`: столбцы, из которых собирать статистику. Разделите несколько столбцов запятыми (`,`). Если этот параметр не указан, собирается вся таблица. Этот параметр

обязателен для гистограмм.

- [WITH SYNC | ASYNC MODE]: выполнять ли задачу ручного сбора в синхронном или асинхронном режиме. Синхронный сбор используется по умолчанию, если вы не указываете этот параметр.

- `WITH N BUCKETS`: `N` - количество корзин для сбора гистограммы. Если не указано, используется значение по умолчанию в `fe.conf`.

- PROPERTIES: пользовательские параметры. Если `PROPERTIES` не указан, используются настройки по умолчанию в `fe.conf`.

PROPERTIES	Тип	Значение по умолчанию	Описание
statistic_sample_collect_rows	INT	200000	Минимальное количество строк для сбора. Если значение параметра превышает фактическое количество строк в вашей таблице, выполняется полный сбор.
histogram_buckets_size	LONG	64	Количество корзин по умолчанию для гистограммы.
histogram_mcv_size	INT	100	Количество наиболее частых значений (MCV) для гистограммы.
histogram_sample_ratio	FLOAT	0.1	Коэффициент выборки для гистограммы.
histogram_max_sample_row_count	LONG	10000000	Максимальное количество строк для сбора для гистограммы.
histogram_collect_bucket_ndv_mode	STRING	none	Режим для оценки количества различных значений (NDV) на корзину гистограммы. `none` (по умолчанию, отдельный подсчет не собирается), `hll` (использует HyperLogLog для точной оценки) или `sample` (использует оценщик на основе выборки с низкими накладными расходами).

Количество строк для сбора для гистограммы контролируется несколькими параметрами. Это большее значение между `statistic_sample_collect_rows` и количество строк таблицы * `histogram_sample_ratio`. Число не может превышать значение, указанное в `histogram_max_sample_row_count`. Если значение превышено, приоритет имеет `histogram_max_sample_row_count`.

Фактически используемые свойства можно просмотреть в столбце `Properties` в выводе SHOW ANALYZE STATUS.

Примеры

```
-- Вручную собрать гистограммы для v1, используя настройки по умолчанию.
ANALYZE TABLE tbl_name UPDATE HISTOGRAM ON v1;

-- Вручную собрать гистограммы для v1 и v2 с 32 корзинами, 32 MCVs и 50%
коэффициентом выборки.
ANALYZE TABLE tbl_name UPDATE HISTOGRAM ON v1,v2 WITH 32 BUCKETS
PROPERTIES(
   "histogram_mcv_size" = "32",
   "histogram_sample_ratio" = "0.5"
);

-- Собрать гистограммы для v3, используя режим 'hll' для точной оценки
отдельного подсчета на корзину.
ANALYZE TABLE tbl_name UPDATE HISTOGRAM ON v3
PROPERTIES(
   "histogram_collect_bucket_ndv_mode" = "hll"
);
```

Пользовательский сбор

Настройка автоматической задачи сбора

Задачи сбора по умолчанию, предоставляемые Selena, автоматически собирают статистику для всех баз данных и всех таблиц в соответствии с политикой, и по умолчанию вам не нужно создавать пользовательские задачи сбора.

Если вы хотите настроить автоматическую задачу сбора, вам нужно создать ее через оператор CREATE ANALYZE. Чтобы создать задачу сбора, вам нужны разрешения INSERT и SELECT для собираемой таблицы.

```
-- Автоматически собрать статистику всех баз данных.
CREATE ANALYZE [FULL|SAMPLE] ALL [PROPERTIES (property [,property])]

-- Автоматически собрать статистику всех таблиц в базе данных.
CREATE ANALYZE [FULL|SAMPLE] DATABASE db_name
[PROPERTIES (property [,property])]

-- Автоматически собрать статистику указанных столбцов в таблице.
CREATE ANALYZE [FULL|SAMPLE] TABLE tbl_name (col_name [,col_name])
[PROPERTIES (property [,property])]

-- Автоматически собрать гистограммы указанных столбцов в таблице.
CREATE ANALYZE TABLE tbl_name UPDATE HISTOGRAM ON col_name [, col_name]
[WITH SYNC | ASYNC MODE]
[WITH N BUCKETS]
[PROPERTIES (property [,property])]
```

Описание параметров:

- Тип сбора
  - FULL: указывает полный сбор.
  - SAMPLE: указывает выборочный сбор.
  - Если тип сбора не указан, по умолчанию используется выборочный сбор.
- `col_name`: столбцы, из которых собирать статистику. Разделите несколько столбцов запятыми (`,`). Если этот параметр не указан, собирается вся таблица.
- `PROPERTIES`: пользовательские параметры. Если `PROPERTIES` не указан, используются настройки по умолчанию в `fe.conf`.

PROPERTIES	Тип	Значение по умолчанию	Описание
statistic_auto_collect_ratio	FLOAT	0.8	Порог для определения, является ли статистика для автоматического сбора здоровой. Если здоровье статистики ниже этого порога, запускается автоматический сбор.
statistic_sam	INT	200000	Минимальное количество строк для сбора. Если значение параметра превышает фактическое количество строк в вашей таблице, выполняется полный сбор.

PROPERTIES	Тип	Значение по умолчанию	Описание
ple_collect_rows			
statistic_exclude_pattern	String	null	Имя базы данных или таблицы, которую необходимо исключить в задаче. Вы можете указать базу данных и таблицу, которые не собирают статистику в задаче. Обратите внимание, что это шаблон регулярного выражения, а содержимое совпадения - `database.table`.
statistic_auto_collect_interval	LONG	0	Интервал для автоматического сбора. Единица: секунды. По умолчанию Selena выбирает `statistic_auto_collect_small_table_interval` или `statistic_auto_collect_large_table_interval` в качестве интервала сбора на основе размера таблицы. Если вы указали свойство `statistic_auto_collect_interval` при создании задачи analyze, эта настройка имеет приоритет над `statistic_auto_collect_small_table_interval` и `statistic_auto_collect_large_table_interval`.

Примеры

Автоматический полный сбор

```
-- Автоматически собрать полную статистику всех баз данных.
CREATE ANALYZE ALL;

-- Автоматически собрать полную статистику базы данных.
CREATE ANALYZE DATABASE db_name;

-- Автоматически собрать полную статистику всех таблиц в базе данных.
CREATE ANALYZE FULL DATABASE db_name;

-- Автоматически собрать полную статистику указанных столбцов в таблице.
CREATE ANALYZE TABLE tbl_name(c1, c2, c3);

-- Автоматически собрать статистику всех баз данных, исключая указанную
базу данных 'db_name'.
CREATE ANALYZE ALL PROPERTIES (
   "statistic_exclude_pattern" = "db_name\."
);

-- Автоматически собрать гистограммы указанной базы данных, таблицы или
столбцов.
CREATE ANALYZE TABLE tbl_name UPDATE HISTOGRAM ON c1,c2;
```

Автоматический выборочный сбор

```
-- Автоматически собрать статистику всех таблиц в базе данных с настройками
по умолчанию.
CREATE ANALYZE SAMPLE DATABASE db_name;

-- Автоматически собрать статистику всех таблиц в базе данных, исключая
указанную таблицу 'db_name.tbl_name'.
CREATE ANALYZE SAMPLE DATABASE db_name PROPERTIES (
   "statistic_exclude_pattern" = "db_name.tbl_name"
);

-- Автоматически собрать статистику указанных столбцов в таблице с
указанием здоровья статистики и количества строк для сбора.
CREATE ANALYZE SAMPLE TABLE tbl_name(c1, c2, c3) PROPERTIES (
   "statistic_auto_collect_ratio" = "0.5",
   "statistic_sample_collect_rows" = "1000000"
);
```

Вместо задачи автоматического сбора, предоставляемой Selena, используйте пользовательскую задачу сбора, которая не собирает таблицу `db_name.tbl_name`.

```
ADMIN SET FRONTEND CONFIG("enable_auto_collect_statistics"="false");
DROP ALL ANALYZE JOB;
CREATE ANALYZE FULL ALL db_name PROPERTIES (
   "statistic_exclude_pattern" = "db_name.tbl_name"
);
```

Сбор статистики во время загрузки данных

Чтобы обеспечить хороший план выполнения для запросов сразу после загрузки данных, Selena запускает асинхронную задачу сбора статистики в конце операторов INSERT INTO/OVERWRITE DML, которая по умолчанию ждет 30 секунд после завершения DML. Если задача сбора статистики не завершается за 30 секунд, возвращается результат выполнения DML.

INSERT INTO

- Статистика собирается только для первого импорта данных в partition.
- Если количество строк в этом импорте больше `statistic_sample_collect_rows`, запускается задача выборочного сбора, в противном случае используется полный сбор.

INSERT OVERWRITE

- Если коэффициент изменения количества строк до и после OVERWRITE меньше `statistic_sample_collect_ratio_threshold_of_first_load`, задача сбора статистики не запускается.
- Если количество строк в этой операции OVERWRITE больше `statistic_sample_collect_rows`, то запускается задача выборочного сбора, в противном случае используется полный сбор.

Следующие свойства (PROPERTIES) используются для создания настроенных задач сбора для загрузки данных. Если не настроено, используется значение соответствующего параметра конфигурации FE.

PROPERTIES	Тип	По умолчанию	Описание
enable_statistic_collect_on_first_load	BOOLEAN	TRUE	Запускать ли задачу сбора статистики после выполнения INSERT INTO/OVERWRITE.
semi_sync_collect_statistic_await_seconds	LONG	30	Максимальное время ожидания сбора статистики перед возвратом результатов.
statistic_sample_collect_ratio_threshold_of_first_load	DOUBLE	0.1	Коэффициент изменения данных в операциях OVERWRITE, которые не запустят задачи сбора статистики.
statistic_sample_collect_rows	LONG	200000	Когда общее количество строк данных, загруженных через операторы DML, превышает это значение, для сбора статистики используется выборочный сбор.

Просмотр пользовательских задач сбора

```
SHOW ANALYZE JOB [WHERE predicate][ORDER BY columns][LIMIT num]
```

Вы можете фильтровать результаты, используя предложение WHERE. Оператор возвращает следующие столбцы.

Столбец	Описание
Id	ID задачи сбора.
Database	Имя базы данных.
Table	Имя таблицы.
Columns	Имена столбцов.
Type	Тип статистики, включая `FULL` и `SAMPLE`.
Schedule	Тип планирования. Тип `SCHEDULE` для автоматической задачи.
Properties	Пользовательские параметры.
Status	Статус задачи, включая PENDING, RUNNING, SUCCESS и FAILED.
LastWorkTime	Время последнего сбора.
Reason	Причина, по которой задача не удалась. Возвращается NULL, если выполнение задачи было успешным.

Примеры

```
-- Просмотреть все пользовательские задачи сбора.
SHOW ANALYZE JOB

-- Просмотреть пользовательские задачи сбора базы данных `test`.
SHOW ANALYZE JOB where `database` = 'test';
```

Удаление пользовательской задачи сбора

```
DROP ANALYZE <ID>
| DROP ALL ANALYZE JOB
```

ID задачи можно получить с помощью оператора SHOW ANALYZE JOB.

Примеры

```
DROP ANALYZE 266030;
```

```
DROP ALL ANALYZE JOB;
```

6.2.6 Просмотр статуса задач сбора

Вы можете просмотреть статус всех текущих задач, выполнив оператор SHOW ANALYZE STATUS. Этот оператор нельзя использовать для просмотра статуса пользовательских задач сбора. Чтобы просмотреть статус пользовательских задач сбора, используйте SHOW ANALYZE JOB.

```
SHOW ANALYZE STATUS [WHERE predicate];
```

Вы можете использовать `LIKE или WHERE` для фильтрации возвращаемой информации.

Этот оператор возвращает следующие столбцы.

Имя списка	Описание
Id	ID задачи сбора.
Database	Имя базы данных.
Table	Имя таблицы.

Имя списка	Описание
Columns	Имена столбцов.
Type	Тип статистики, включая FULL, SAMPLE и HISTOGRAM.
Schedule	Тип планирования. `ONCE` означает ручной, а `SCHEDULE` означает автоматический.
Status	Статус задачи.
StartTime	Время, когда задача начинает выполнение.
EndTime	Время, когда выполнение задачи заканчивается.
Properties	Пользовательские параметры.
Reason	Причина, по которой задача не удалась. Возвращается NULL, если выполнение было успешным.

6.2.7 Просмотр статистики

Просмотр метаданных базовой статистики

```
SHOW STATS META [WHERE predicate][ORDER BY columns][LIMIT num]
```

Этот оператор возвращает следующие столбцы.

Столбец	Описание
Database	Имя базы данных.
Table	Имя таблицы.
Columns	Имена столбцов.
Type	Тип статистики. `FULL` означает полный сбор, а `SAMPLE` означает выборочный сбор.
UpdateTime	Последнее время обновления статистики для текущей таблицы.
Properties	Пользовательские параметры.
Healthy	Здоровье статистической информации.
ColumnStats	Тип ANALYZE столбца.
TabletStatsReportTime	Время, когда метаданные Tablet для таблицы были обновлены в FE.
TableHealthyMetrics	Метрика здоровья в статистической информации.
TableUpdateTime	Время, когда таблица была обновлена.

Просмотр метаданных гистограмм

```
SHOW HISTOGRAM META [WHERE predicate]
```

Этот оператор возвращает следующие столбцы.

Столбец	Описание
Database	Имя базы данных.
Table	Имя таблицы.
Column	Столбцы.
Type	Тип статистики. Значение `HISTOGRAM` для гистограмм.
UpdateTime	Последнее время обновления статистики для текущей таблицы.
Properties	Пользовательские параметры.

6.2.8 Удаление статистики

Вы можете удалить статистическую информацию, которая вам не нужна. При удалении статистики удаляются как данные, так и метаданные статистики, а также статистика в устаревшем кэше. Обратите внимание, что если выполняется задача автоматического сбора, ранее удаленная статистика может быть снова собрана. Вы можете использовать `SHOW ANALYZE STATUS` для просмотра истории задач сбора.

Удаление базовой статистики

Следующий оператор удаляет статистику, хранящуюся в таблице `default_catalog._statistics_.column_statistics`, и соответствующая статистика таблицы, кэшированная FE, также будет аннулирована. Начиная с версии 3.5.0, этот оператор также удаляет многостолбцовую совместную статистику для этой таблицы.

```
DROP STATS tbl_name
```

Следующий оператор удаляет многостолбцовую совместную статистику, хранящуюся в таблице `default_catalog._statistics_.multi_column_statistics`, и многостолбцовая совместная статистика в соответствующей таблице, кэшированной FE, также будет аннулирована. Этот оператор не удаляет базовую статистику таблицы.

```
DROP MULTIPLE COLUMNS STATS tbl_name
```

Удаление гистограмм

```
ANALYZE TABLE tbl_name DROP HISTOGRAM ON col_name [, col_name]
```

6.2.9 Отмена задачи сбора

Вы можете использовать оператор KILL ANALYZE для отмены **выполняющейся** задачи сбора, включая ручные и пользовательские задачи.

```
KILL ANALYZE <ID>
```

ID задачи для задачи ручного сбора можно получить из SHOW ANALYZE STATUS. ID задачи для пользовательской задачи сбора можно получить из SHOW ANALYZE JOB.

6.2.10 Другие параметры конфигурации FE

Параметр конфигурации FE	Тип	Значение по умолчанию	Описание
statistic_manager_sleep_time_sec	LONG	60	Интервал, с которым планируются метаданные. Единица: секунды. Система выполняет следующие операции на основе этого интервала: Создание таблиц для хранения статистики. Удаление удаленной статистики. Удаление просроченной статистики.
statistic_analyze_status_keep_second	LONG	259200	Длительность хранения истории задач сбора. Единица: секунды. Значение по умолчанию: 259200 (3 дня).

6.2.11 Переменная сессии

`statistic_collect_parallel`: Используется для настройки параллелизма задач сбора статистики, которые могут выполняться на BE. Значение по умолчанию: 1. Вы можете увеличить это значение для ускорения задач сбора.

6.2.12 Сбор статистики внешних таблиц

Начиная с версии 3.2.0, Selena поддерживает сбор статистики таблиц Hive, Iceberg и Hudi. Синтаксис аналогичен сбору внутренних таблиц Selena. **Однако поддерживается только ручной полный сбор, ручной сбор гистограмм (начиная с версии 3.2.7) и автоматический полный сбор. Выборочный сбор не поддерживается.** Начиная с версии 3.3.0, Selena поддерживает сбор статистики таблиц Delta Lake и статистики подполей в STRUCT. Начиная с версии 3.4.0, Selena поддерживает автоматический сбор статистики через задачи ANALYZE, запускаемые запросами.

Собранная статистика хранится в таблице `external_column_statistics` базы данных `_statistics_` в `default_catalog`. Они не хранятся в Hive Metastore и не могут быть совместно использованы другими поисковыми системами. Вы можете запросить данные из таблицы `default_catalog._statistics_.external_column_statistics`, чтобы проверить, собрана ли статистика для таблицы Hive/Iceberg/Hudi.

Ниже приведен пример запроса статистических данных из `external_column_statistics`.

```
SELECT * FROM _statistics_.external_column_statistics\G
*************************** 1. row ***************************
    table_uuid: hive_catalog.tn_test.ex_hive_tbl.1673596430
partition_name:
   column_name: k1
  catalog_name: hive_catalog
       db_name: tn_test
    table_name: ex_hive_tbl
     row_count: 3
     data_size: 12
           ndv: NULL
    null_count: 0
           max: 3
           min: 2
   update_time: 2023-12-01 14:57:27.137000
```

Ограничения

При сборе статистики для внешних таблиц применяются следующие ограничения:

- Вы можете собирать статистику только таблиц Hive, Iceberg, Hudi и Delta Lake (начиная с версии 3.3.0).
- Поддерживается только ручной полный сбор, ручной сбор гистограмм (начиная с версии 3.2.7) и автоматический полный сбор. Выборочный сбор не поддерживается.
- Для автоматического сбора полной статистики системой необходимо создать задачу Analyze, что отличается от сбора статистики внутренних таблиц Selena, где система делает это в фоновом режиме по умолчанию.
- Для автоматических задач сбора:
    - Вы можете собирать статистику только для конкретной таблицы. Вы не можете собирать статистику всех таблиц в базе данных или статистику всех баз данных во внешнем каталоге.

    - Selena может определить, обновлены ли данные в таблицах Hive и Iceberg, и если да, собрать статистику только для partition, данные которых обновлены. Selena не может воспринять, обновлены ли данные в таблицах Hudi, и может только выполнять периодический полный сбор.
- Для задач сбора, запускаемых запросами:
    - В настоящее время только узел Leader FE может запускать задачи ANALYZE.
    - Система поддерживает только проверку изменений partition в таблицах Hive и Iceberg и собирает статистику только для partition, где данные изменились. Для таблиц Delta Lake/Hudi система собирает статистику всей таблицы.
    - Если к таблицам Iceberg применяются Partition Transforms, сбор статистики поддерживается только для Transforms типа `identity`, `year`, `month`, `day`, `hour`.
    - Сбор статистики для Partition Evolution для таблиц Iceberg не поддерживается.

Следующие примеры происходят в базе данных под внешним каталогом Hive. Если вы хотите собрать статистику таблицы Hive из `default_catalog`, ссылайтесь на таблицу в формате `[catalog_name.][database_name.]<table_name>`.

Сбор, запускаемый запросами

Начиная с версии 3.4.0, система поддерживает автоматический сбор статистики внешних таблиц с помощью задач ANALYZE, запускаемых запросами. При запросе таблиц Hive, Iceberg, Hudi или Delta Lake система автоматически запустит задачи ANALYZE в фоновом режиме для сбора статистики соответствующих таблиц и столбцов и использует ее для последующей оптимизации плана запроса.

Рабочий процесс:

1. Когда оптимизатор запрашивает кэшированную статистику в FE, он определит объект задачи ANALYZE на основе запрашиваемых таблиц и столбцов (задача ANALYZE будет собирать только статистику столбцов, включенных в запрос).
2. Система инкапсулирует объект задачи как задачу ANALYZE и добавляет ее в PendingTaskQueue.
3. Поток Scheduler периодически извлекает задачи из PendingTaskQueue и добавляет их в RunningTasksQueue.
4. Во время выполнения задачи ANALYZE она собирает статистику и записывает ее в BE и очищает просроченную статистику, кэшированную в FE.

Эта функция включена по умолчанию. Вы можете контролировать вышеуказанный процесс с помощью следующих системных переменных и параметров конфигурации.

Системная переменная

enable_query_trigger_analyze

- По умолчанию: true
- Тип: Boolean
- Описание: Включить ли задачи ANALYZE, запускаемые запросами.
- Введено в: v1.5.2

Конфигурация FE

connector_table_query_trigger_analyze_small_table_rows

- По умолчанию: 10000000
- Тип: Int
- Единица: -
- Изменяемый: Да
- Описание: Порог для определения, является ли таблица малой для задач ANALYZE, запускаемых запросами.
- Введено в: v1.5.2

connector_table_query_trigger_analyze_small_table_interval

- По умолчанию: 2 * 3600
- Тип: Int
- Единица: Секунда
- Изменяемый: Да
- Описание: Интервал для задач ANALYZE, запускаемых запросами, для малых таблиц.
- Введено в: v1.5.2

connector_table_query_trigger_analyze_large_table_interval

- По умолчанию: 12 * 3600
- Тип: Int
- Единица: Секунда
- Изменяемый: Да
- Описание: Интервал для задач ANALYZE, запускаемых запросами, для больших таблиц.
- Введено в: v1.5.2

connector_table_query_trigger_analyze_max_pending_task_num

- По умолчанию: 100
- Тип: Int
- Единица: -
- Изменяемый: Да
- Описание: Максимальное количество задач ANALYZE, запускаемых запросами, которые находятся в состоянии Pending на FE.
- Введено в: v1.5.2

connector_table_query_trigger_analyze_schedule_interval

- По умолчанию: 30
- Тип: Int
- Единица: Секунда
- Изменяемый: Да
- Описание: Интервал, с которым поток Scheduler планирует задачи ANALYZE, запускаемые запросами.
- Введено в: v1.5.2

connector_table_query_trigger_analyze_max_running_task_num

- По умолчанию: 2
- Тип: Int
- Единица: -
- Изменяемый: Да
- Описание: Максимальное количество задач ANALYZE, запускаемых запросами, которые находятся в состоянии Running на FE.
- Введено в: v1.5.2

Ручной сбор

Вы можете создать задачу Analyze по требованию, и задача запускается сразу после ее создания.

Создание задачи ручного сбора

Синтаксис:

```
-- Ручной полный сбор
ANALYZE [FULL] TABLE tbl_name (col_name [,col_name])
[WITH SYNC | ASYNC MODE]
[PROPERTIES(property [,property])]

-- Ручной сбор гистограмм (начиная с версии 3.3.0)
ANALYZE TABLE tbl_name UPDATE HISTOGRAM ON col_name [, col_name]
[WITH SYNC | ASYNC MODE]
[WITH N BUCKETS]
[PROPERTIES (property [,property])]
```

Ниже приведен пример ручного полного сбора:

```
ANALYZE TABLE ex_hive_tbl(k1);
+----------------------------------+---------+----------+----------+
| Table                            | Op      | Msg_type | Msg_text |
+----------------------------------+---------+----------+----------+
| hive_catalog.tn_test.ex_hive_tbl | analyze | status   | OK       |
+----------------------------------+---------+----------+----------+
```

Просмотр статуса задачи

Синтаксис:

```
SHOW ANALYZE STATUS [LIKE | WHERE predicate]
```

Пример:

```
SHOW ANALYZE STATUS where `table` = 'ex_hive_tbl';
+-------+----------------------+-------------+---------+------+----------+-
--------+---------------------+---------------------+------------+--------+
| Id    | Database             | Table       | Columns | Type | Schedule |
Status  | StartTime           | EndTime             | Properties | Reason |
+-------+----------------------+-------------+---------+------+----------+-
--------+---------------------+---------------------+------------+--------+
| 16400 | hive_catalog.tn_test | ex_hive_tbl | k1      | FULL | ONCE     |
SUCCESS | 2023-12-04 16:31:42 | 2023-12-04 16:31:42 | {}         |        |
| 16465 | hive_catalog.tn_test | ex_hive_tbl | k1      | FULL | ONCE     |
SUCCESS | 2023-12-04 16:37:35 | 2023-12-04 16:37:35 | {}         |        |
| 16467 | hive_catalog.tn_test | ex_hive_tbl | k1      | FULL | ONCE     |
SUCCESS | 2023-12-04 16:37:46 | 2023-12-04 16:37:46 | {}         |        |
+-------+----------------------+-------------+---------+------+----------+-
--------+---------------------+---------------------+------------+--------+
```

Просмотр метаданных статистики

Синтаксис:

```
SHOW STATS META [WHERE predicate]
```

Пример:

```
SHOW STATS META where `table` = 'ex_hive_tbl';
+----------------------+-------------+---------+------+-------------------
-+------------+---------+
| Database             | Table       | Columns | Type | UpdateTime
| Properties | Healthy |
+----------------------+-------------+---------+------+-------------------
-+------------+---------+
| hive_catalog.tn_test | ex_hive_tbl | k1      | FULL | 2023-12-04 16:37:46
| {}         |         |
+----------------------+-------------+---------+------+-------------------
-+------------+---------+
```

Отмена задачи сбора

Отменяет выполняющуюся задачу сбора.

Синтаксис:

```
KILL ANALYZE <ID>
```

Вы можете просмотреть ID задачи в выводе SHOW ANALYZE STATUS.

Автоматический сбор

Для автоматического сбора статистики таблиц во внешнем источнике данных системой вы можете создать задачу Analyze. Selena автоматически проверяет, нужно ли запускать задачу, с интервалом проверки по умолчанию 5 минут. Для таблиц Hive и Iceberg Selena запускает задачу сбора только тогда, когда данные в таблицах обновлены.

Однако изменения данных в таблицах Hudi не могут быть восприняты, и Selena периодически собирает статистику на основе интервала проверки и интервала сбора, которые вы указали. Вы можете указать следующие параметры конфигурации FE для управления поведением сбора:

- statistic_collect_interval_sec

  Интервал для проверки обновлений данных во время автоматического сбора. Единица: секунды. По умолчанию: 5 минут.

- statistic_auto_collect_small_table_rows (v1.5.2 и позже)

  Порог для определения, является ли таблица во внешнем источнике данных (Hive, Iceberg, Hudi) малой таблицей во время автоматического сбора. По умолчанию: 10000000.

- statistic_auto_collect_small_table_interval

  Интервал для сбора статистики малых таблиц. Единица: секунды. По умолчанию: 0.

- statistic_auto_collect_large_table_interval

  Интервал для сбора статистики больших таблиц. Единица: секунды. По умолчанию: 43200 (12 часов).

Потоки автоматического сбора проверяют обновления данных с интервалом, указанным в `statistic_collect_interval_sec`. Если количество строк в таблице меньше

`statistic_auto_collect_small_table_rows`, она собирает статистику таких таблиц на основе `statistic_auto_collect_small_table_interval`.

Если количество строк в таблице превышает `statistic_auto_collect_small_table_rows`, она собирает статистику таких таблиц на основе `statistic_auto_collect_large_table_interval`. Она обновляет статистику больших таблиц только тогда, когда `Время последнего обновления таблицы + Интервал сбора > Текущее время`. Это предотвращает частые задачи analyze для больших таблиц.

Создание автоматической задачи сбора

Синтаксис:

```
CREATE ANALYZE TABLE tbl_name (col_name [,col_name])
[PROPERTIES (property [,property])]
```

Вы можете указать свойство `statistic_auto_collect_interval` для установки интервала сбора исключительно для автоматической задачи сбора. Параметры конфигурации FE `statistic_auto_collect_small_table_interval` и `statistic_auto_collect_large_table_interval` не будут действовать на эту задачу.

Пример:

```
CREATE ANALYZE TABLE ex_hive_tbl (k1)
PROPERTIES ("statistic_auto_collect_interval" = "5");

Query OK, 0 rows affected (0.01 sec)
```

Просмотр статуса автоматической задачи сбора

То же, что и для ручного сбора.

Просмотр метаданных статистики

То же, что и для ручного сбора.

Просмотр автоматических задач сбора

Синтаксис:

```
SHOW ANALYZE JOB [WHERE predicate]
```

Пример:

```
SHOW ANALYZE JOB WHERE `id` = '17204';

Empty set (0.00 sec)
```

Отмена задачи сбора

То же, что и для ручного сбора.

Удаление статистики

```
DROP STATS tbl_name
```

6.2.13 Ссылки

- Для запроса параметров конфигурации FE выполните ADMIN SHOW CONFIG.
- Для изменения параметров конфигурации FE выполните ADMIN SET CONFIG.

6.3 SQL Plan Manager

> *Beta* **BETA FEATURE**
>
> Рекомендации по использованию бета-функций

Эта тема знакомит с базовыми концепциями и сценариями использования функции SQL Plan Manager, а также с тем, как использовать SQL Plan Manager для регуляризации планов запросов.

Начиная с версии v1.5.2, Selena поддерживает функцию SQL Plan Manager.

6.3.1 Обзор

SQL Plan Manager позволяет пользователям привязывать план запроса к запросу, тем самым предотвращая изменение плана запроса из-за изменений состояния системы (в

основном обновлений данных и обновлений статистики), стабилизируя производительность запросов.

Рабочий процесс

SQL Plan Manager требует предварительного указания привязанного SQL запроса и плана запроса (Baseline), который будет использоваться. Здесь запрос относится к фактическому SQL запросу, выполняемому пользователем, в то время как план запроса - это SQL запрос, оптимизированный вручную или с добавленными hints.

Рабочий процесс SQL Plan Manager выглядит следующим образом:

1. **Создание Baseline**: Используйте команду `CREATE BASELINE` для привязки плана запроса к указанному SQL запросу.
2. **Переписывание запроса**: Запросы, отправленные в Selena, автоматически сопоставляются с Baselines, хранящимися в SQL Plan Manager. Если они успешно совпадают, план запроса из Baseline используется для запроса.

Примечания по созданию Baseline:

- Вы должны обеспечить логическую согласованность между привязанным SQL и SQL плана выполнения в Baseline. Selena выполняет базовые проверки таблиц и параметров, но не может гарантировать полную проверку логической согласованности. Обеспечение логической корректности плана выполнения - это ответственность самих пользователей.
- По умолчанию привязанный SQL в Baseline хранит свой собственный SQL fingerprint. По умолчанию постоянные значения в SQL будут заменены на переменные параметры (например, изменение `t1.v1 > 1000` на `t1.v1 > ?`) для улучшения сопоставления SQL.
- План выполнения, привязанный в Baseline, может быть настроен путем изменения логики SQL или добавления Hints (Join Hints или `Set_Var`) для обеспечения генерации желаемого плана выполнения.
- Для сложных SQL Selena может не автоматически привязывать SQL и план выполнения в Baseline. В таких случаях можно использовать ручную привязку, как подробно описано в разделе Расширенное использование.

Примечания по переписыванию запросов:

- SQL Plan Manager в первую очередь полагается на сопоставление SQL fingerprint. Он проверяет, совпадает ли SQL fingerprint запроса с fingerprint Baseline. Если запрос совпадает с Baseline, параметры в запросе автоматически подставляются в план выполнения Baseline.
- В процессе сопоставления, если запрос совпадает с несколькими Baselines со статусом `enable`, оптимизатор оценивает и выбирает оптимальный Baseline.
- В процессе сопоставления SQL Plan Manager проверяет, соответствует ли Baseline запросу. Если сопоставление не удается, план запроса Baseline не будет использован.
- Для планов выполнения, переписанных SQL Plan Manager, оператор `EXPLAIN` вернет `Using baseline plan[id]`.

6.3.2 Управление Baseline

Создание Baseline

Синтаксис:

```
CREATE [GLOBAL] BASELINE [ON <BindSQL>] USING <PlanSQL>
[PROPERTIES ("key" = "value"[, ...])]
```

Параметры:

- `GLOBAL`: (Опционально) Создает Baseline глобального уровня.
- `BindSQL`: (Опционально) Конкретный запрос, который будет привязан к запросу Baseline (плану выполнения). Если этот параметр не указан, запрос Baseline привязывается к самому себе и использует свой собственный план запроса.
- `PlanSQL`: Запрос, используемый для определения плана выполнения.

Примеры:

```
-- Создать BASELINE уровня сессии, напрямую привязав SQL Baseline к самому себе и используя его собственный план запроса.
CREATE BASELINE
USING SELECT t1.v2, t2.v3 FROM t1 JOIN t2 ON t1.v2 = t2.v2 WHERE t1.v2 > 100;

-- Создать BASELINE глобального уровня, напрямую привязав SQL Baseline к самому себе и используя его собственный план запроса с указанными Join Hints.
CREATE GLOBAL BASELINE
USING SELECT t1.v2, t2.v3 FROM t1 JOIN[BROADCAST] t2 ON t1.v2 = t2.v2 WHERE t1.v2 > 100;

-- Создать BASELINE уровня сессии, привязав запрос к SQL Baseline и используя план запроса SQL Baseline с указанными Join Hints.
CREATE BASELINE ON SELECT t1.v2, t2.v3 FROM t1, t2 WHERE t1.v2 = t2.v2 AND t1.v2 > 100
USING SELECT t1.v2, t2.v3 FROM t1 JOIN[BROADCAST] t2 on t1.v2 = t2.v2 where t1.v2 > 100;
```

Просмотр Baseline

Синтаксис:

```
SHOW BASELINE [WHERE <condition>]

SHOW BASELINE [ON <query>]
```

Пример:

```
MySQL > show baseline\G;
***************************[ 1. row ]***************************
Id            | 646125
global        | N
enable        | N
bindSQLDigest | SELECT * FROM `td`.`t2` INNER JOIN `td`.`t1` ON
`td`.`t2`.`v2` = `td`.`t1`.`v2` LIMIT 2
bindSQLHash   | 1085294
bindSQL       | SELECT * FROM `td`.`t2` INNER JOIN `td`.`t1` ON
`td`.`t2`.`v2` = `td`.`t1`.`v2` LIMIT 2
planSQL       | SELECT t2.v1 AS c_1, t2.v2 AS c_2, t2.v3 AS c_3, t1.v1 AS
c_4, t1.v2 AS c_5 FROM t2 INNER JOIN[SHUFFLE] t1 ON t2.v2 = t1.v2 LIMIT 2
costs         | 582.0
queryMs       | -1.0
source        | USER
updateTime    | 2025-05-16 14:50:45
***************************[ 2. row ]***************************
Id            | 636134
global        | Y
enable        | Y
bindSQLDigest | SELECT * FROM `td`.`t2` INNER JOIN `td`.`t1` ON
`td`.`t2`.`v2` = `td`.`t1`.`v2` WHERE `td`.`t2`.`v3` = ?
bindSQLHash   | 1085294
bindSQL       | SELECT * FROM `td`.`t2` INNER JOIN `td`.`t1` ON
`td`.`t2`.`v2` = `td`.`t1`.`v2` WHERE `td`.`t2`.`v3` = _spm_const_range(1,
10, 20)
planSQL       | SELECT t_0.v1 AS c_1, t_0.v2 AS c_2, t_0.v3 AS c_3, t1.v1
AS c_4, t1.v2 AS c_5 FROM (SELECT * FROM t2 WHERE v3 = _spm_const_range(1,
10, 20)) t_0 INNER JOIN[SHUFFLE] t1 ON t_0.v2 = t1.v2
costs         | 551.0204081632653
queryMs       | -1.0
source        | USER
updateTime    | 2025-05-13 15:29:04
2 rows in set
Time: 0.019s

MySQL > show baseline where global = true\G;
***************************[ 1. row ]***************************
Id            | 636134
global        | Y
enable        | Y
bindSQLDigest | SELECT * FROM `td`.`t2` INNER JOIN `td`.`t1` ON
`td`.`t2`.`v2` = `td`.`t1`.`v2` WHERE `td`.`t2`.`v3` = ?
bindSQLHash   | 1085294
bindSQL       | SELECT * FROM `td`.`t2` INNER JOIN `td`.`t1` ON
```

```
`td`.`t2`.`v2` = `td`.`t1`.`v2` WHERE `td`.`t2`.`v3` = _spm_const_range(1,
10, 20)
planSQL       | SELECT t_0.v1 AS c_1, t_0.v2 AS c_2, t_0.v3 AS c_3, t1.v1
AS c_4, t1.v2 AS c_5 FROM (SELECT * FROM t2 WHERE v3 = _spm_const_range(1,
10, 20)) t_0 INNER JOIN[SHUFFLE] t1 ON t_0.v2 = t1.v2
costs         | 551.0204081632653
queryMs       | -1.0
source        | USER
updateTime    | 2025-05-13 15:29:04
1 row in set
Time: 0.013s

MySQL > show baseline on SELECT count(1) AS `count(1)` FROM `old`.`t1`
INNER JOIN `old`.`t2` ON `old`.`t1`.`k2` = `old`.`t2`.`k2` LIMIT 10\G;
***************************[ 1. row ]***************************
Id            | 679817
global        | Y
enable        | Y
bindSQLDigest | SELECT count(?) AS `count(1)` FROM `old`.`t1` INNER JOIN
`old`.`t2` ON `old`.`t1`.`k2` = `old`.`t2`.`k2` LIMIT 10
bindSQLHash   | 1085927
bindSQL       | SELECT count(_spm_const_var(1)) AS `count(1)` FROM
`old`.`t1` INNER JOIN `old`.`t2` ON `old`.`t1`.`k2` = `old`.`t2`.`k2` LIMIT
10
planSQL       | SELECT count(_spm_const_var(1)) AS c_7 FROM (SELECT 1 AS
c_9 FROM t1 INNER JOIN[SHUFFLE] t2 ON t1.k2 = t2.k2) t_0 LIMIT 10
costs         | 2532.6
queryMs       | 35.0
source        | CAPTURE
updateTime    | 2025-05-27 11:17:48
1 row in set
Time: 0.026s
```

Удаление Baseline

Синтаксис:

```
DROP BASELINE <id>,<id>...
```

Параметр:

`id`: ID Baseline. Вы можете получить ID Baseline, выполнив `SHOW BASELINE`.

Пример:

```
-- Удалить Baseline с ID 140035.
DROP BASELINE 140035;
```

Включение/Отключение Baseline

Синтаксис:

```
ENABLE BASELINE <id>,<id>...
DISABLE BASELINE <id>,<id>...
```

Параметр:

`id`: ID Baseline. Вы можете получить ID Baseline, выполнив `SHOW BASELINE`.

Пример:

```
-- включить baseline с id 140035
ENABLE BASELINE 140035;
-- отключить baseline с id 140035 140037
DISABLE BASELINE 140035, 140037;
```

6.3.3 Переписывание запросов

Вы можете включить функцию переписывания запросов SQL Plan Manager, установив переменную `enable_spm_rewrite` в `true`.

```
SET enable_spm_rewrite = true;
```

После привязки плана выполнения Selena автоматически перепишет соответствующий запрос в соответствующий план запроса.

Пример:

Проверьте исходный SQL и план выполнения:

```
mysql> EXPLAIN SELECT t1.v2, t2.v3 FROM t1, t2 WHERE t1.v2 = t2.v2 AND
t1.v1 > 20;
+-----------------------------------------+
| Explain String                          |
+-----------------------------------------+
| PLAN FRAGMENT 0                         |
|  OUTPUT EXPRS:2: v2 | 5: v3             |
|   PARTITION: UNPARTITIONED              |
......
|   5:HASH JOIN                           |
|   |  join op: INNER JOIN (PARTITIONED)  |
|   |  colocate: false, reason:           |
|   |  equal join conjunct: 4: v2 = 2: v2 |
|   |                                     |
|   |----4:EXCHANGE                       |
|   |                                     |
|   1:EXCHANGE                            |
......
```

Создайте Baseline для привязки исходного SQL к плану выполнения SQL с Join Hints:

```
MySQL td> create global baseline on select t1.v2, t2.v3 from t1, t2 where
t1.v2 = t2.v2 and t1.v1 > 1000
using select t1.v2, t2.v3 from t1 join[broadcast] t2 on t1.v2 = t2.v2 where
t1.v1 > 1000;
Query OK, 0 rows affected
Time: 0.074s
MySQL td> show baseline\G;
***************************[ 1. row ]***************************
Id            | 647139
global        | Y
enable        | Y
bindSQLDigest | SELECT `td`.`t1`.`v2`, `td`.`t2`.`v3` FROM `td`.`t1` ,
`td`.`t2`  WHERE (`td`.`t1`.`v2` = `td`.`t2`.`v2`) AND (`td`.`t1`.`v1` > ?)
bindSQLHash   | 1085294
bindSQL       | SELECT `td`.`t1`.`v2`, `td`.`t2`.`v3` FROM `td`.`t1` ,
`td`.`t2`  WHERE (`td`.`t1`.`v2` = `td`.`t2`.`v2`) AND (`td`.`t1`.`v1` >
_spm_const_var(1))
planSQL       | SELECT t_0.v2 AS c_2, t2.v3 AS c_5 FROM (SELECT v2 FROM t1
WHERE v1 > _spm_const_var(1)) t_0 INNER JOIN[BROADCAST] t2 ON t_0.v2 =
t2.v2
costs         | 1193.0
queryMs       | -1.0
source        | USER
updateTime    | 2025-05-16 15:51:36
1 rows in set
Time: 0.016s
```

Включите переписывание запросов SQL Plan Manager и проверьте, переписан ли исходный запрос с помощью Baseline:

```
MySQL td> show variables like '%enable_spm_re%'
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| enable_spm_rewrite | false |
+--------------------+-------+
1 row in set
Time: 0.007s
MySQL td> set enable_spm_rewrite=true
Query OK, 0 rows affected
Time: 0.001s
MySQL td> explain select t1.v2, t2.v3 from t1, t2 where t1.v2 = t2.v2 and
t1.v1 > 20;
+-----------------------------------------+
| Explain String                          |
+-----------------------------------------+
| Using baseline plan[647139]             |
|                                         |
| PLAN FRAGMENT 0                         |
|  OUTPUT EXPRS:2: v2 | 5: v3             |
.............
|   4:HASH JOIN                           |
|   |  join op: INNER JOIN (BROADCAST)    |
|   |  colocate: false, reason:           |
|   |  equal join conjunct: 2: v2 = 4: v2 |
|   |                                     |
|   |----3:EXCHANGE                       |
|   |                                     |
|   1:Project                             |
|   |  <slot 2> : 2: v2                   |
.............
```

6.3.4 Расширенное использование

Для следующих сценариев вы можете попробовать использовать ручную привязку плана запроса:

- Для сложных SQL, SQL Plan Manager не может автоматически привязать SQL и план запроса.
- Для конкретных сценариев (например, фиксированные параметры или условные параметры) автоматическая привязка не может удовлетворить требования.

По сравнению с автоматической привязкой, ручная привязка предлагает большую гибкость, но требует понимания некоторых механизмов выполнения SQL Plan Manager.

Логика выполнения

Функции SQL Plan Manager

Функции SQL Plan Manager - это функции-заполнители в SQL Plan Manager с двумя основными целями:

- Пометить выражения в параметризованном SQL для последующего извлечения и замены параметров в процессе.
- Проверить условия параметров, сопоставляя SQL с различными параметрами с различными планами запросов через условия параметров.

В настоящее время Selena поддерживает следующие функции SQL Plan Manager:

- `_spm_const_var(placeholdID)`: Используется для пометки одного постоянного значения.
- `_spm_const_list(placeholdID)`: Используется для пометки нескольких постоянных значений, обычно используется для пометки нескольких постоянных значений в условии IN.
- `_spm_const_range(placeholdID, min, max)`: Используется для пометки одного постоянного значения, но требует, чтобы постоянное значение находилось в указанном диапазоне '[min, max]'.
- `_spm_const_num(placeholdID, value...)`: Используется для пометки одного постоянного значения, но требует, чтобы постоянное значение было значением в указанном перечислении 'value...'.

`placeholdID` - это целое число, которое используется как уникальный идентификатор для параметра, который используется при привязке Baseline и генерации плана.

Процесс создания Baseline

1. Выполните `CREATE BASELINE` для получения привязываемого `BindSQL` и запроса плана выполнения `PlanSQL`.
2. Параметризуйте `BindSQL`: Замените литеральные значения или выражения функциями SQL Plan Manager. Например, замените `id > 200` на `id > _spm_const_var(0)`, где параметр `0` - это ID заполнителя, используемый для

подтверждения позиции выражения в `BindSQL` и `PlanSQL`. Для получения дополнительных функций SQL Plan Manager см. Функции SQL Plan Manager.
3. Привяжите заполнители в `PlanSQL`: Найдите позицию заполнителей в `PlanSQL` и замените их исходными выражениями.
4. Используйте оптимизатор для оптимизации `PlanSQL` и получения плана запроса.
5. Сериализуйте план запроса в SQL с Hints.
6. Сохраните Baseline (SQL fingerprint BindSQL, оптимизированный SQL плана выполнения).

Процесс переписывания запроса

Логика переписывания запроса похожа на оператор `PREPARE`.

1. Выполните запрос.
2. Нормализуйте запрос в SQL fingerprint.
3. Используйте SQL fingerprint для поиска Baseline (сопоставление с `BindSQL` Baseline).
4. Привяжите запрос к Baseline, проверьте, соответствует ли запрос `BindSQL` Baseline, и извлеките соответствующие значения параметров из запроса, используя функции SQL Plan Manager в `BindSQL`. Например, `id > 1000` в запросе привязывается к `id > _spm_const_var(0)` в BindSQL, извлекая `_spm_const_var(0) = 1000`.
5. Замените параметры SQL Plan Manager в `PlanSQL` Baseline.
6. Верните `PlanSQL` для замены исходного запроса.

Ручная привязка запроса

Вы можете использовать функции SQL Plan Manager для привязки более сложных SQL к Baselines.

Пример 1

Например, SQL для привязки выглядит следующим образом:

```
create global baseline using
with ss as (
    select i_item_id, sum(ss_ext_sales_price) total_sales
    from store_sales, item
    where i_color in ('slate', 'blanched', 'burnished') and ss_item_sk =
i_item_sk
    group by i_item_id
),
cs as (
    select i_item_id, sum(cs_ext_sales_price) total_sales
    from catalog_sales, item
    where i_color in ('slate', 'blanched', 'burnished') and cs_item_sk =
i_item_sk
    group by i_item_id
)
select i_item_id, sum(total_sales) total_sales
from (  select * from ss
        union all
        select * from cs) tmp1
group by i_item_id;
```

Поскольку постоянные значения в `i_color in ('slate', 'blanched', 'burnished')` одинаковы, SQL будет распознан с SQL Plan Manager как:

```
with ss as (
    select i_item_id, sum(ss_ext_sales_price) total_sales
    from store_sales, item
    where i_color IN (_spm_const_list(1)) and ss_item_sk = i_item_sk
    group by i_item_id
),
cs as (
    select i_item_id, sum(cs_ext_sales_price) total_sales
    from catalog_sales, item
    where i_color IN (_spm_const_list(1)) and cs_item_sk = i_item_sk
    group by i_item_id
)
select i_item_id, sum(total_sales) total_sales
from (  select * from ss
        union all
        select * from cs) tmp1
group by i_item_id;
```

Это означает, что оба экземпляра `i_color in ('xxx', 'xxx')` распознаются как один и тот же параметр, что делает невозможным для SQL Plan Manager их различение, когда в SQL используются разные параметры.

```
-- can be bind baseline
MySQL tpcds> explain with ss as (
    select i_item_id, sum(ss_ext_sales_price) total_sales
    from store_sales, item
    where i_color in ('A', 'B', 'C') and ss_item_sk = i_item_sk
    group by i_item_id
),
cs as (
    select i_item_id, sum(cs_ext_sales_price) total_sales
    from catalog_sales, item
    where i_color in ('A', 'B', 'C') and cs_item_sk = i_item_sk
    group by i_item_id
)
select i_item_id, sum(total_sales) total_sales
from (  select * from ss
        union all
        select * from cs) tmp1
group by i_item_id;
+-------------------------------------------------------------------------------
-----------------+
| Explain String
|
+-------------------------------------------------------------------------------
-----------------+
| Using baseline plan[646215]
|
|
|
| PLAN FRAGMENT 0
|
|  OUTPUT EXPRS:104: i_item_id | 106: sum
|
|   PARTITION: UNPARTITIONED
|
................................
|
|      avgRowSize=3.0
|
+-------------------------------------------------------------------------------
-----------------+
184 rows in set
Time: 0.095s

-- can't bind basline
MySQL tpcds> explain with ss as (
```

```
    select i_item_id, sum(ss_ext_sales_price) total_sales
    from store_sales, item
    where i_color in ('A', 'B', 'C') and ss_item_sk = i_item_sk
    group by i_item_id
),
cs as (
    select i_item_id, sum(cs_ext_sales_price) total_sales
    from catalog_sales, item
    where i_color in ('E', 'F', 'G') and cs_item_sk = i_item_sk
    group by i_item_id
)
select i_item_id, sum(total_sales) total_sales
from (  select * from ss
        union all
        select * from cs) tmp1
group by i_item_id;
+-------------------------------------------------------------------------
-----------------+
| Explain String
|
+-------------------------------------------------------------------------
-----------------+
| PLAN FRAGMENT 0
|
|  OUTPUT EXPRS:104: i_item_id | 106: sum
|
|   PARTITION: UNPARTITIONED
|
................................
|
|      avgRowSize=3.0
|
+-------------------------------------------------------------------------
-----------------+
182 rows in set
Time: 0.040s
```

В таких случаях вы можете вручную указать параметры в `BindSQL` и `PlanSQL`:

```
create global baseline using
with ss as (
    select i_item_id, sum(ss_ext_sales_price) total_sales
    from store_sales, item
    where i_color IN (_spm_const_list(1)) and ss_item_sk = i_item_sk
    group by i_item_id
),
cs as (
    select i_item_id, sum(cs_ext_sales_price) total_sales
    from catalog_sales, item
    where i_color IN (_spm_const_list(2)) and cs_item_sk = i_item_sk
    group by i_item_id
)
select i_item_id, sum(total_sales) total_sales
from (  select * from ss
        union all
        select * from cs) tmp1
group by i_item_id;
```

Проверьте, переписан ли запрос с помощью Baseline:

```
MySQL td> show baseline\G;
***************************[ 1. row ]***************************
Id            | 646215
global        | Y
enable        | Y
bindSQLDigest | WITH `ss` (`i_item_id`, `total_sales`) AS (SELECT
`tpcds`.`item`.`i_item_id`, sum(`tpcds`.`store_sales`.`ss_ext_sales_price`)
AS `total_sales` FROM `tpcds`.`store_sales` , `tpcds`.`item`  WHERE
(`tpcds`.`item`.`i_color` IN (?)) AND (`tpcds`.`store_sales`.`ss_item_sk` =
`tpcds`.`item`.`i_item_sk`) GROUP BY `tpcds`.`item`.`i_item_id`) , `cs`
(`i_item_id`, `total_sales`) AS (SELECT `tpcds`.`item`.`i_item_id`,
sum(`tpcds`.`catalog_sales`.`cs_ext_sales_price`) AS `total_sales` FROM
`tpcds`.`catalog_sales` , `tpcds`.`item`  WHERE (`tpcds`.`item`.`i_color`
IN (?)) AND (`tpcds`.`catalog_sales`.`cs_item_sk` =
`tpcds`.`item`.`i_item_sk`) GROUP BY `tpcds`.`item`.`i_item_id`) SELECT
`tmp1`.`i_item_id`, sum(`tmp1`.`total_sales`) AS `total_sales` FROM (SELECT
* FROM `ss` UNION ALL SELECT * FROM `cs`) `tmp1` GROUP BY
`tmp1`.`i_item_id`
bindSQLHash   | 203487418
bindSQL       | WITH `ss` (`i_item_id`, `total_sales`) AS (SELECT
`tpcds`.`item`.`i_item_id`, sum(`tpcds`.`store_sales`.`ss_ext_sales_price`)
AS `total_sales` FROM `tpcds`.`store_sales` , `tpcds`.`item`  WHERE
(`tpcds`.`item`.`i_color` IN (_spm_const_list(1))) AND
(`tpcds`.`store_sales`.`ss_item_sk` = `tpcds`.`item`.`i_item_sk`) GROUP BY
`tpcds`.`item`.`i_item_id`) , `cs` (`i_item_id`, `total_sales`) AS (SELECT
`tpcds`.`item`.`i_item_id`,
sum(`tpcds`.`catalog_sales`.`cs_ext_sales_price`) AS `total_sales` FROM
`tpcds`.`catalog_sales` , `tpcds`.`item`  WHERE (`tpcds`.`item`.`i_color`
IN (_spm_const_list(1))) AND (`tpcds`.`catalog_sales`.`cs_item_sk` =
`tpcds`.`item`.`i_item_sk`) GROUP BY `tpcds`.`item`.`i_item_id`) SELECT
`tmp1`.`i_item_id`, sum(`tmp1`.`total_sales`) AS `total_sales` FROM (SELECT
* FROM `ss` UNION ALL SELECT * FROM `cs`) `tmp1` GROUP BY
`tmp1`.`i_item_id`
planSQL       | SELECT c_104, sum(c_105) AS c_106 FROM (SELECT * FROM
(SELECT i_item_id AS c_104, c_46 AS c_105 FROM (SELECT i_item_id,
sum(ss_ext_sales_price) AS c_46 FROM (SELECT i_item_id, ss_ext_sales_price
FROM store_sales INNER JOIN[BROADCAST] (SELECT i_item_sk, i_item_id FROM
item WHERE i_color IN (_spm_const_list(1))) t_0 ON ss_item_sk = i_item_sk)
t_1 GROUP BY i_item_id) t_2 UNION ALL SELECT i_item_id AS c_104, c_103 AS
c_105 FROM (SELECT i_item_id, sum(cs_ext_sales_price) AS c_103 FROM (SELECT
i_item_id, cs_ext_sales_price FROM catalog_sales INNER JOIN[BROADCAST]
(SELECT i_item_sk, i_item_id FROM item WHERE i_color IN
(_spm_const_list(1))) t_3 ON cs_item_sk = i_item_sk) t_4 GROUP BY
i_item_id) t_5) t_6) t_7 GROUP BY c_104
costs         | 2.608997082E8
```

```
queryMs       | -1.0
source        | USER
updateTime    | 2025-05-16 15:30:29
***************************[ 2. row ]***************************
Id            | 646237
global        | Y
enable        | Y
bindSQLDigest | WITH `ss` (`i_item_id`, `total_sales`) AS (SELECT
`tpcds`.`item`.`i_item_id`, sum(`tpcds`.`store_sales`.`ss_ext_sales_price`)
AS `total_sales` FROM `tpcds`.`store_sales` , `tpcds`.`item`  WHERE
(`tpcds`.`item`.`i_color` IN (?)) AND (`tpcds`.`store_sales`.`ss_item_sk` =
`tpcds`.`item`.`i_item_sk`) GROUP BY `tpcds`.`item`.`i_item_id`) , `cs`
(`i_item_id`, `total_sales`) AS (SELECT `tpcds`.`item`.`i_item_id`,
sum(`tpcds`.`catalog_sales`.`cs_ext_sales_price`) AS `total_sales` FROM
`tpcds`.`catalog_sales` , `tpcds`.`item`  WHERE (`tpcds`.`item`.`i_color`
IN (?)) AND (`tpcds`.`catalog_sales`.`cs_item_sk` =
`tpcds`.`item`.`i_item_sk`) GROUP BY `tpcds`.`item`.`i_item_id`) SELECT
`tmp1`.`i_item_id`, sum(`tmp1`.`total_sales`) AS `total_sales` FROM (SELECT
* FROM `ss` UNION ALL SELECT * FROM `cs`) `tmp1` GROUP BY
`tmp1`.`i_item_id`
bindSQLHash   | 203487418
bindSQL       | WITH `ss` (`i_item_id`, `total_sales`) AS (SELECT
`tpcds`.`item`.`i_item_id`, sum(`tpcds`.`store_sales`.`ss_ext_sales_price`)
AS `total_sales` FROM `tpcds`.`store_sales` , `tpcds`.`item`  WHERE
(`tpcds`.`item`.`i_color` IN (_spm_const_list(1))) AND
(`tpcds`.`store_sales`.`ss_item_sk` = `tpcds`.`item`.`i_item_sk`) GROUP BY
`tpcds`.`item`.`i_item_id`) , `cs` (`i_item_id`, `total_sales`) AS (SELECT
`tpcds`.`item`.`i_item_id`,
sum(`tpcds`.`catalog_sales`.`cs_ext_sales_price`) AS `total_sales` FROM
`tpcds`.`catalog_sales` , `tpcds`.`item`  WHERE (`tpcds`.`item`.`i_color`
IN (_spm_const_list(2))) AND (`tpcds`.`catalog_sales`.`cs_item_sk` =
`tpcds`.`item`.`i_item_sk`) GROUP BY `tpcds`.`item`.`i_item_id`) SELECT
`tmp1`.`i_item_id`, sum(`tmp1`.`total_sales`) AS `total_sales` FROM (SELECT
* FROM `ss` UNION ALL SELECT * FROM `cs`) `tmp1` GROUP BY
`tmp1`.`i_item_id`
planSQL       | SELECT c_104, sum(c_105) AS c_106 FROM (SELECT * FROM
(SELECT i_item_id AS c_104, c_46 AS c_105 FROM (SELECT i_item_id,
sum(ss_ext_sales_price) AS c_46 FROM (SELECT i_item_id, ss_ext_sales_price
FROM store_sales INNER JOIN[BROADCAST] (SELECT i_item_sk, i_item_id FROM
item WHERE i_color IN (_spm_const_list(1))) t_0 ON ss_item_sk = i_item_sk)
t_1 GROUP BY i_item_id) t_2 UNION ALL SELECT i_item_id AS c_104, c_103 AS
c_105 FROM (SELECT i_item_id, sum(cs_ext_sales_price) AS c_103 FROM (SELECT
i_item_id, cs_ext_sales_price FROM catalog_sales INNER JOIN[BROADCAST]
(SELECT i_item_sk, i_item_id FROM item WHERE i_color IN
(_spm_const_list(2))) t_3 ON cs_item_sk = i_item_sk) t_4 GROUP BY
i_item_id) t_5) t_6) t_7 GROUP BY c_104
```

```
costs          | 2.635637082E8
queryMs        | -1.0
source         | USER
updateTime     | 2025-05-16 15:37:35
2 rows in set
Time: 0.013s
MySQL td> explain with ss as (
    select i_item_id, sum(ss_ext_sales_price) total_sales
    from store_sales, item
    where i_color in ('A', 'B', 'C') and ss_item_sk = i_item_sk
    group by i_item_id
),
cs as (
    select i_item_id, sum(cs_ext_sales_price) total_sales
    from catalog_sales, item
    where i_color in ('E', 'F', 'G') and cs_item_sk = i_item_sk
    group by i_item_id
)
select i_item_id, sum(total_sales) total_sales
from (  select * from ss
        union all
        select * from cs) tmp1
group by i_item_id;
+-------------------------------------------------------------------------------
-----------------+
| Explain String
|
+-------------------------------------------------------------------------------
-----------------+
| Using baseline plan[646237]
|
|
|
| PLAN FRAGMENT 0
|
|  OUTPUT EXPRS:104: i_item_id | 106: sum
|
|   PARTITION: UNPARTITIONED
|
......
```

Из приведенного выше вывода видно, что запрос переписан с помощью Baseline с различными параметрами в функциях SQL Plan Manager.

Пример 2

Для следующего запроса, если вы хотите использовать различный baseline для разных `i_color`

```
select i_item_id, sum(ss_ext_sales_price) total_sales
from store_sales join item on ss_item_sk = i_item_sk
where i_color = 25
group by i_item_id
```

Вы можете использовать '_spm_const_range' для:

```
-- 10 <= i_color <= 50 время, используется SHUFFLE JOIN
MySQL tpcds> create baseline  using select i_item_id,
sum(ss_ext_sales_price) total_sales
from store_sales join[SHUFFLE] item on ss_item_sk = i_item_sk
where i_color = _spm_const_range(1, 10, 50)
group by i_item_id
Query OK, 0 rows affected
Time: 0.017s
-- i_color для 60 ,70 ,80 время, используется BROADCAST JOIN
MySQL tpcds> create baseline  using select i_item_id,
sum(ss_ext_sales_price) total_sales
from store_sales join[BROADCAST] item on ss_item_sk = i_item_sk
where i_color = _spm_const_enum(1, 60, 70, 80)
group by i_item_id
Query OK, 0 rows affected
Time: 0.009s
MySQL tpcds> show baseline\G;
***************************[ 1. row ]***************************
Id            | 647167
global        | N
enable        | Y
bindSQLDigest | SELECT `tpcds`.`item`.`i_item_id`,
sum(`tpcds`.`store_sales`.`ss_ext_sales_price`) AS `total_sales` FROM
`tpcds`.`store_sales` INNER JOIN `tpcds`.`item` ON
`tpcds`.`store_sales`.`ss_item_sk` = `tpcds`.`item`.`i_item_sk` WHERE
`tpcds`.`item`.`i_color` = ? GROUP BY `tpcds`.`item`.`i_item_id`
bindSQLHash   | 68196091
bindSQL       | SELECT `tpcds`.`item`.`i_item_id`,
sum(`tpcds`.`store_sales`.`ss_ext_sales_price`) AS `total_sales` FROM
`tpcds`.`store_sales` INNER JOIN `tpcds`.`item` ON
`tpcds`.`store_sales`.`ss_item_sk` = `tpcds`.`item`.`i_item_sk` WHERE
`tpcds`.`item`.`i_color` = _spm_const_range(1, 10, 50) GROUP BY
`tpcds`.`item`.`i_item_id`
planSQL       | SELECT i_item_id, sum(ss_ext_sales_price) AS c_46 FROM
(SELECT i_item_id, ss_ext_sales_price FROM store_sales INNER JOIN[SHUFFLE]
(SELECT i_item_sk, i_item_id FROM item WHERE i_color = _spm_const_range(1,
'10', '50')) t_0 ON ss_item_sk = i_item_sk) t_1 GROUP BY i_item_id
costs         | 1.612502146E8
queryMs       | -1.0
source        | USER
updateTime    | 2025-05-16 16:02:46
***************************[ 2. row ]***************************
Id            | 647171
global        | N
enable        | Y
```

```
bindSQLDigest | SELECT `tpcds`.`item`.`i_item_id`,
sum(`tpcds`.`store_sales`.`ss_ext_sales_price`) AS `total_sales` FROM
`tpcds`.`store_sales` INNER JOIN `tpcds`.`item` ON
`tpcds`.`store_sales`.`ss_item_sk` = `tpcds`.`item`.`i_item_sk` WHERE
`tpcds`.`item`.`i_color` = ? GROUP BY `tpcds`.`item`.`i_item_id`
bindSQLHash   | 68196091
bindSQL       | SELECT `tpcds`.`item`.`i_item_id`,
sum(`tpcds`.`store_sales`.`ss_ext_sales_price`) AS `total_sales` FROM
`tpcds`.`store_sales` INNER JOIN `tpcds`.`item` ON
`tpcds`.`store_sales`.`ss_item_sk` = `tpcds`.`item`.`i_item_sk` WHERE
`tpcds`.`item`.`i_color` = _spm_const_enum(1, 60, 70, 80) GROUP BY
`tpcds`.`item`.`i_item_id`
planSQL       | SELECT i_item_id, sum(ss_ext_sales_price) AS c_46 FROM
(SELECT i_item_id, ss_ext_sales_price FROM store_sales INNER
JOIN[BROADCAST] (SELECT i_item_sk, i_item_id FROM item WHERE i_color =
_spm_const_enum(1, '60', '70', '80')) t_0 ON ss_item_sk = i_item_sk) t_1
GROUP BY i_item_id
costs         | 1.457490986E8
queryMs       | -1.0
source        | USER
updateTime    | 2025-05-16 16:03:23
2 rows in set
Time: 0.011s
MySQL tpcds>
MySQL tpcds> explain select i_item_id, sum(ss_ext_sales_price) total_sales
from store_sales join item on ss_item_sk = i_item_sk
where i_color = 40 -- hit SHUFFLE JOIN
group by i_item_id
+-------------------------------------------------------------------------
-----------------+
| Explain String
|
+-------------------------------------------------------------------------
-----------------+
| Using baseline plan[647167]
|
|
|
| PLAN FRAGMENT 0
|
................
|   |
|
|   5:HASH JOIN
|
|   |  join op: INNER JOIN (PARTITIONED)
```

```
|
|   |  colocate: false, reason:
|
|   |  equal join conjunct: 2: ss_item_sk = 24: i_item_sk
|
|   |
|
|   |----4:EXCHANGE
|
|   |
|
|   1:EXCHANGE
|
.................
MySQL tpcds> explain select i_item_id, sum(ss_ext_sales_price) total_sales
from store_sales join item on ss_item_sk = i_item_sk
where i_color = 70 -- hit BROADCAST JOIN
group by i_item_id
+-------------------------------------------------------------------------
-----------------+
| Explain String
|
+-------------------------------------------------------------------------
-----------------+
| Using baseline plan[647171]
|
|
|
| PLAN FRAGMENT 0
|
.................
|   4:HASH JOIN
|
|   |  join op: INNER JOIN (BROADCAST)
|
|   |  colocate: false, reason:
|
|   |  equal join conjunct: 2: ss_item_sk = 24: i_item_sk
|
|   |
|
|   |----3:EXCHANGE
|
|   |
|
|   0:OlapScanNode
```

```
|
|      TABLE: store_sales
|
.................
MySQL tpcds> explain select i_item_id, sum(ss_ext_sales_price) total_sales
from store_sales join item on ss_item_sk = i_item_sk
where i_color = 100  -- don't use any Baseline
group by i_item_id
+-----------------------------------------------------------------------------
-----------------+
| Explain String
|
+-----------------------------------------------------------------------------
-----------------+
| PLAN FRAGMENT 0
|
|  OUTPUT EXPRS:25: i_item_id | 46: sum
|
|   PARTITION: UNPARTITIONED
|
|
|
|   RESULT SINK
|
.................
```

Auto-Capture

Auto-Capture запрашивает SQL запросы за прошедший период времени (по умолчанию 3 часа), генерирует и сохраняет baseline на основе этих запросов, и сгенерированный baseline находится в состоянии 'disable' по умолчанию и не вступает в силу немедленно. В следующих сценариях:

- После обновления план выполнения изменяется, что приводит к увеличению времени запроса
- После изменения данных и изменения статистики, что приводит к изменению плана выполнения

Вы можете найти исторический baseline с помощью `show baseline` и вручную откатить план с помощью `enable baseline`.

Функция Auto-Capture зависит от функции сохранения истории запросов и требует следующих настроек:

```
set global enable_query_history=true;
```

История запросов хранится в таблице '*statistics*.query_history'.

Чтобы включить автоматический захват:

```
set global enable_plan_capture=true;
```

Другие конфигурации:

```
-- Историческая длительность хранения истории запросов, единица: секунды,
по умолчанию 3 дня
set global query_history_keep_seconds = 259200;
-- Рабочий интервал Auto-Capture, единица: секунды, по умолчанию 3 часа
set global plan_capture_interval=10800;
-- Захватывает регулярные проверки таблиц SQL, захватывает SQL только когда
имена таблиц (db.table) могут соответствовать plan_capture_include_pattern,
по умолчанию .*, что представляет все таблицы
set global plan_capture_include_pattern=".*";
```

ПРИМЕЧАНИЕ

1. Сохранение истории запросов и Auto-Capture потребуют некоторых ресурсов хранения и вычислений, поэтому установите это разумно в соответствии с вашими собственными сценариями.
2. После привязки baselines вновь добавленные оптимизации после обновлений могут стать неэффективными, поэтому автоматически захваченные baselines отключены по умолчанию.

Пример использования

Используйте функцию Auto-Capture во время обновлений, чтобы избежать проблем с регрессией плана после обновления:

1. Включите функциональность Auto-Capture за 1-2 дня до обновления:

```
set global enable_query_history=true;
set global enable_plan_capture=true;
```

2. Selena начинает периодически записывать планы запросов, которые можно просмотреть с помощью `show baseline`

3. Обновите Selena

4. После обновления проверьте время выполнения запроса или используйте следующий SQL для идентификации запросов с изменениями плана:

```
WITH recent_queries AS (
    -- Используйте время выполнения запроса в течение 3 дней как среднее
время выполнения
    SELECT
        dt,                     -- Время выполнения запроса
        sql_digest,             -- SQL fingerprint запроса
        `sql`,                  -- SQL запроса
        query_ms,               -- Время выполнения
        plan,                   -- Используемый план запроса
        AVG(query_ms) OVER (PARTITION BY sql_digest) AS avg_ms, -- Среднее
время выполнения в группе SQL fingerprint
        RANK() OVER (PARTITION BY sql_digest ORDER BY plan) != 1 AS
is_changed -- Подсчитайте различные форматы плана как индикатор изменения
    FROM _statistics_.query_history
    WHERE dt >= NOW() - INTERVAL 3 DAY
)
-- Запросы со временем выполнения выше 1.5 раз среднего за последние 12
часов
SELECT *, RANK() OVER (PARTITION BY sql_digest ORDER BY query_ms DESC) AS
rnk
FROM recent_queries
WHERE query_ms > avg_ms * 1.5 and dt >= now() - INTERVAL 12 HOUR
```

5. На основе информации об изменении плана или информации о времени выполнения запроса определите, необходим ли откат плана

6. Найдите соответствующий baseline для SQL и временной точки:

```
show baseline on <query>
```

7. Используйте Enable baseline для отката:

```
enable baseline <id>
```

8. Включите переключатель переписывания baseline:

```
set enable_spm_rewrite = true;
```

6.3.5 Будущие планы

В будущем Selena предоставит более продвинутые функции на основе SQL Plan Manager, включая:

- Улучшенные проверки стабильности для SQL планов.
- Автоматическую оптимизацию фиксированных планов запросов.
- Поддержку большего количества методов привязки условных параметров.

6.4 Query Feedback

> **EXPERIMENTAL FEATURE**
>
> Рекомендации по использованию экспериментальных функций

В этой теме представлена функция Query Feedback, её сценарии применения и способы оптимизации планов запросов на основе обратной связи от статистики выполнения с использованием Query Plan Advisor.

Selena поддерживает функцию Query Feedback начиная с версии v1.5.2.

6.4.1 Обзор

Query Feedback — это фреймворк и критически важный компонент Cost-based Optimizer (CBO). Он записывает статистику выполнения во время выполнения запроса и повторно использует её в последующих запросах с похожими планами запросов, чтобы помочь CBO генерировать оптимизированные планы запросов. CBO оптимизирует планы запросов на основе оценочной статистики, поэтому когда статистическая информация устарела или неточна, он может выбирать неэффективные планы запросов (плохие планы), такие как broadcast-передача большой таблицы или неправильный порядок левой и правой таблиц.

Эти плохие планы могут вызывать таймауты выполнения запросов, чрезмерное потребление ресурсов или даже сбои системы.

6.4.2 Рабочий процесс

Рабочий процесс оптимизации планов на основе Query Feedback состоит из трёх этапов:

1. **Наблюдение**: BE или CN записывает основные метрики (включая `InputRows` и `OutputRows`) PlanNode в каждом плане запроса.
2. **Анализ**: Для медленных запросов, превышающих настроенный порог, и запросов, помеченных для анализа вручную, система анализирует детали выполнения в критических узлах для выявления возможностей оптимизации в текущем плане запроса после завершения запроса и до возврата результата. FE сравнивает план запроса со статистикой выполнения и проверяет, является ли запрос медленным из-за аномального плана запроса. При анализе неточной статистики FE генерирует руководство по настройке SQL для каждого запроса, инструктирует CBO динамически оптимизировать запрос и рекомендует стратегии для улучшения производительности.
3. **Оптимизация**: После того как CBO генерирует физический план, он ищет любое существующее руководство по настройке, применимое к плану. Если такое есть, CBO динамически оптимизирует план в соответствии с руководством и стратегиями, исправляя проблемные участки, тем самым устраняя влияние на производительность запроса из-за повторного использования плохих планов запросов. Время выполнения оптимизированного плана сравнивается с исходным планом для оценки эффективности настройки.

6.4.3 Использование

Управляемый системной переменной `enable_plan_advisor` (по умолчанию: `true`), Query Plan Advisor включён по умолчанию для медленных запросов, то есть запросов со временем выполнения, превышающим порог, определённый в параметре конфигурации FE `slow_query_analyze_threshold` (по умолчанию: `5` секунд).

Кроме того, вы можете вручную анализировать конкретный запрос или включить автоматический анализ для всех выполняемых запросов.

Ручной анализ конкретного запроса

Вы можете вручную проанализировать конкретный оператор запроса, даже если время его выполнения не превышает `slow_query_analyze_threshold`.

```
ALTER PLAN ADVISOR ADD <query_statement>
```

Пример:

```
ALTER PLAN ADVISOR ADD SELECT COUNT(*) FROM (
    SLECT * FROM c1_skew_left_over t1
    JOIN (SELECT * FROM c1_skew_left_over WHERE c1 = 'c') t2
    ON t1.c2 = t2.c2 WHERE t1.c1 > 'c' ) t;
```

Включение автоматического анализа для всех запросов

Чтобы включить автоматический анализ для всех запросов, необходимо установить системную переменную `enable_plan_analyzer` (по умолчанию: `false`) в значение `true`.

```
SET enable_plan_analyzer = true;
```

Просмотр руководств по настройке на текущем FE

Каждый FE ведёт собственную запись руководств по настройке. Вы можете использовать следующий оператор для просмотра руководств по настройке, сгенерированных для соответствующих запросов на текущем FE:

```
SHOW PLAN ADVISOR
```

Проверка применения руководства по настройке

Выполните EXPLAIN для оператора запроса. В строке EXPLAIN сообщение `Plan had been tuned by Plan Advisor` указывает на то, что руководство по настройке было применено к соответствующему запросу.

Пример:

```
EXPLAIN SELECT COUNT(*) FROM (
    SLECT * FROM c1_skew_left_over t1
    JOIN (SELECT * FROM c1_skew_left_over WHERE c1 = 'c') t2
    ON t1.c2 = t2.c2 WHERE t1.c1 > 'c' ) t;
+-----------------------------------------------------------------------
---------------------+
| Explain String
|
+-----------------------------------------------------------------------
---------------------+
| Plan had been tuned by Plan Advisor.
|
| Original query id:8e010cf4-b178-11ef-8aa4-8a5075cec65e
|
| Original time cost: 148 ms
|
| 1: LeftChildEstimationErrorTuningGuide
|
| Reason: left child statistics of JoinNode 5 had been overestimated.
|
| Advice: Adjust the distribution join execution type and join plan to
improve the performance. |
|
|
| PLAN FRAGMENT 0
|
|  OUTPUT EXPRS:9: count
|
|   PARTITION: UNPARTITIONED
```

Удаление руководства по настройке для конкретного запроса

Вы можете удалить руководство по настройке для конкретного запроса на основе идентификатора запроса, возвращённого из `SHOW PLAN ADVISOR`.

```
ALTER PLAN ADVISOR DROP <query_id>
```

Пример:

```
ALTER PLAN ADVISOR DROP "8e010cf4-b178-11ef-8aa4-8a5075cec65e";
```

Очистка всех руководств по настройке на текущем FE

Чтобы очистить все руководства по настройке на текущем FE, выполните следующий оператор:

```
TRUNCATE PLAN ADVISOR
```

6.4.4 Сценарии использования

В настоящее время Query Feedback в основном используется для оптимизации следующих сценариев:

- Оптимизация порядка левой и правой сторон в локальных узлах Join
- Оптимизация метода выполнения локальных узлов Join (например, переключение с Broadcast на Shuffle)
- Для случаев со значительным потенциалом агрегации принудительное использование режима `pre_aggregation` для максимизации агрегации данных на первой фазе

Руководства по настройке в основном основаны на метриках `Runtime Exec Node Input/Output Rows` и `statistics estimated rows` FE. Поскольку текущие пороговые значения настройки относительно консервативны, рекомендуется использовать Query Feedback для проверки потенциальных улучшений производительности при обнаружении проблем в Query Profile или строке EXPLAIN.

Ниже представлены три распространённых сценария использования.

Случай 1: Неправильный порядок Join

Исходный плохой план:

```
small left table inner join large table (broadcast)
```

Оптимизированный план:

```
large left table inner join small right table (broadcast)
```

Причина Проблема может быть вызвана устаревшей или отсутствующей статистикой, что приводит к тому, что Cost-Based Optimizer (CBO) генерирует неправильный план на основе неточных данных.

Решение Во время выполнения запроса система сравнивает `input/output rows` и `statistics estimated rows` Left Child и Right Child, генерируя руководства по настройке. При повторном выполнении система автоматически корректирует порядок Join.

Случай 2: Неправильный метод выполнения Join

Исходный плохой план:

```
large left table1 inner join large right table2 (broadcast)
```

Оптимизированный план:

```
large left table1 (shuffle) inner join large right table2 (shuffle)
```

Причина Проблема может быть результатом перекоса данных. Когда правая таблица имеет много partition и один из них содержит непропорционально большой объём данных, система может неправильно оценить количество строк правой таблицы после применения предикатов.

Решение Во время выполнения запроса система сравнивает `input/output rows` и `statistics estimated rows` Left Child и Right Child, генерируя руководства по настройке. После оптимизации метод Join изменяется с Broadcast Join на Shuffle Join.

Случай 3: Неэффективный режим предварительной агрегации первой фазы

Симптом Для данных с хорошим потенциалом агрегации режим `auto` агрегации первой фазы может агрегировать только небольшой объём локальных данных, упуская возможность повышения производительности.

Решение Во время выполнения запроса система собирает `Input/Output Rows` для локальных и глобальных агрегаций. На основе исторических данных она оценивает потенциал столбцов агрегации. Если потенциал значительный, система принудительно

использует режим `pre_aggregation` в локальных агрегациях, максимизируя агрегацию данных на первой фазе и улучшая общую производительность запроса.

6.4.5 Ограничения

- Руководство по настройке может использоваться только для точно такого же запроса, для которого оно было сгенерировано. Оно не применимо к запросам с тем же шаблоном, но с другими параметрами.
- Каждый FE управляет своим Query Plan Advisor независимо, и синхронизация между узлами FE не поддерживается. Если один и тот же запрос отправляется на разные узлы FE, результаты настройки могут отличаться.
- Query Plan Advisor использует структуру кэша в памяти:
  - Когда количество руководств по настройке превышает лимит, просроченные руководства автоматически удаляются.
  - Лимит руководств по настройке по умолчанию составляет 300, и сохранение исторических руководств по настройке не поддерживается.

6.5 Синхронное материализованное представление

Эта тема описывает, как создавать, использовать и управлять **синхронным материализованным представлением (Rollup)**.

Для синхронного материализованного представления все изменения в базовой таблице одновременно обновляются в соответствующих синхронных материализованных представлениях. Обновление синхронного материализованного представления запускается автоматически. Синхронные материализованные представления значительно недороги в обслуживании и обновлении, что делает их подходящими для прозрачного ускорения запросов в реальном времени с агрегацией по одной таблице.

Синхронные материализованные представления в Selena могут быть созданы только на одной базовой таблице из каталога по умолчанию. По сути, они представляют собой специальный индекс для ускорения запросов, а не физическую таблицу, как асинхронные материализованные представления.

Начиная с версии v1.5.2, Selena предоставляет асинхронные материализованные представления, которые поддерживают создание на нескольких таблицах и большее количество операторов агрегации. Для использования **асинхронных**

материализованных представлений см. Асинхронное материализованное представление.

> **ПРИМЕЧАНИЕ**
>
> - Синхронные материализованные представления поддерживают условия WHERE начиная с версии v1.5.2.
> - Синхронные материализованные представления поддерживаются в shared-data кластерах начиная с версии v1.5.2.

Следующая таблица сравнивает асинхронные материализованные представления (ASYNC MV) в Selena v1.5.2 и синхронное материализованное представление (SYNC MV) с точки зрения поддерживаемых функций:

	Агрегация по одной таблице	Соединение нескольких таблиц	Переписывание запросов	Стратегия обновления	Базовая таблица
ASYNC MV	Да	Да	Да	• Асинхронное обновление • Ручное обновление	Несколько таблиц из: • Каталога по умолчанию • Внешних каталогов (v1.5.2) • Существующих материализованных представлений (v1.5.2) • Существующих представлений (v1.5.2)
SYNC MV (Rollup)	Ограниченный выбор агрегатных функций	Нет	Да	Синхронное обновление во время загрузки данных	Одна таблица в каталоге по умолчанию

6.5.1 Базовые концепции

- **Базовая таблица**

  Базовые таблицы являются исходными таблицами материализованного представления.

Для синхронных материализованных представлений Selena базовые таблицы должны быть одной нативной таблицей из каталога по умолчанию. Selena поддерживает создание синхронных материализованных представлений на таблицах Duplicate Key и Aggregate.

- **Обновление**

  Синхронное материализованное представление обновляется автоматически при каждом изменении данных в базовой таблице. Вам не нужно запускать обновление вручную.

- **Переписывание запросов**

  Переписывание запросов означает, что при выполнении запроса к базовым таблицам с построенными на них материализованными представлениями система автоматически определяет, можно ли использовать предварительно вычисленные результаты в материализованном представлении для запроса. Если они могут быть использованы, система загрузит данные непосредственно из соответствующего материализованного представления, чтобы избежать затратных по времени и ресурсам вычислений или соединений.

  Синхронные материализованные представления поддерживают переписывание запросов на основе некоторых агрегатных операторов. Для получения дополнительной информации см. Соответствие агрегатных функций.

6.5.2 Подготовка

Перед созданием синхронного материализованного представления проверьте, подходит ли ваше хранилище данных для ускорения запросов с помощью синхронных материализованных представлений. Например, проверьте, используют ли запросы определенные подзапросы повторно.

Следующий пример основан на таблице `sales_records`, которая содержит идентификатор транзакции `record_id`, идентификатор продавца `seller_id`, идентификатор магазина `store_id`, дату `sale_date` и сумму продажи `sale_amt` для каждой транзакции. Выполните следующие шаги для создания таблицы и вставки данных в нее:

```
CREATE TABLE sales_records(
    record_id INT,
    seller_id INT,
    store_id INT,
    sale_date DATE,
    sale_amt BIGINT
) DISTRIBUTED BY HASH(record_id);

INSERT INTO sales_records
VALUES
    (001,01,1,"2022-03-13",8573),
    (002,02,2,"2022-03-14",6948),
    (003,01,1,"2022-03-14",4319),
    (004,03,3,"2022-03-15",8734),
    (005,03,3,"2022-03-16",4212),
    (006,02,2,"2022-03-17",9515);
```

Бизнес-сценарий этого примера требует частого анализа сумм продаж различных магазинов. В результате функция `sum()` используется в каждом запросе, потребляя огромное количество вычислительных ресурсов. Вы можете выполнить запрос, чтобы записать его время, и просмотреть его профиль запроса с помощью команды EXPLAIN.

```
MySQL > SELECT store_id, SUM(sale_amt)
FROM sales_records
GROUP BY store_id;
+----------+-----------------+
| store_id | sum(`sale_amt`) |
+----------+-----------------+
|        2 |           16463 |
|        3 |           12946 |
|        1 |           12892 |
+----------+-----------------+
3 rows in set (0.02 sec)

MySQL > EXPLAIN SELECT store_id, SUM(sale_amt)
FROM sales_records
GROUP BY store_id;
+-------------------------------------------------------------------------------
---+
| Explain String
|
+-------------------------------------------------------------------------------
---+
| PLAN FRAGMENT 0
|
|  OUTPUT EXPRS:3: store_id | 6: sum
|
|   PARTITION: UNPARTITIONED
|
|
|
|   RESULT SINK
|
|
|
|   4:EXCHANGE
|
|
|
| PLAN FRAGMENT 1
|
|  OUTPUT EXPRS:
|
|   PARTITION: HASH_PARTITIONED: 3: store_id
|
|
|
```

```
|   STREAM DATA SINK
|
|     EXCHANGE ID: 04
|
|     UNPARTITIONED
|
|
|
|   3:AGGREGATE (merge finalize)
|
|   |  output: sum(6: sum)
|
|   |  group by: 3: store_id
|
|   |
|
|   2:EXCHANGE
|
|
|
| PLAN FRAGMENT 2
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 02
|
|     HASH_PARTITIONED: 3: store_id
|
|
|
|   1:AGGREGATE (update serialize)
|
|   |  STREAMING
|
|   |  output: sum(5: sale_amt)
|
|   |  group by: 3: store_id
|
|   |
|
```

```
|   0:OlapScanNode
|
|      TABLE: sales_records
|
|      PREAGGREGATION: ON
|
|      partitions=1/1
|
|      rollup: sales_records
|
|      tabletRatio=10/10
|
|
tabletList=12049,12053,12057,12061,12065,12069,12073,12077,12081,12085 |
|      cardinality=1
|
|      avgRowSize=2.0
|
|      numNodes=0
|
+-----------------------------------------------------------------------
---+
45 rows in set (0.00 sec)
```

Можно заметить, что запрос занимает около 0,02 секунд, и синхронное материализованное представление не используется для ускорения запроса, потому что значение поля `rollup` в профиле запроса равно `sales_records`, что является базовой таблицей.

6.5.3 Создание синхронного материализованного представления

Вы можете создать синхронное материализованное представление на основе конкретного запроса, используя CREATE MATERIALIZED VIEW.

На основе таблицы `sales_records` и запроса, упомянутого выше, следующий пример создает синхронное материализованное представление `store_amt` для анализа суммы продаж в каждом магазине.

```
CREATE MATERIALIZED VIEW store_amt AS
SELECT store_id, SUM(sale_amt)
FROM sales_records
GROUP BY store_id;
```

ВНИМАНИЕ

- При использовании агрегатных функций в синхронных материализованных представлениях вы должны использовать условие GROUP BY и указать хотя бы один столбец GROUP BY в вашем списке SELECT.
- Синхронные материализованные представления не поддерживают использование одной агрегатной функции для нескольких столбцов. Запросы в форме `sum(a+b)` не поддерживаются.
- Синхронные материализованные представления не поддерживают использование нескольких агрегатных функций для одного столбца. Запросы в форме `select sum(a), min(a) from table` не поддерживаются.
- JOIN не поддерживается при создании синхронного материализованного представления.
- При использовании ALTER TABLE DROP COLUMN для удаления определенного столбца в базовой таблице необходимо убедиться, что все синхронные материализованные представления базовой таблицы не содержат удаляемого столбца, в противном случае операция удаления не может быть выполнена. Чтобы удалить столбец, используемый в синхронных материализованных представлениях, сначала необходимо удалить все синхронные материализованные представления, содержащие этот столбец, а затем удалить столбец.
- Создание слишком большого количества синхронных материализованных представлений для таблицы повлияет на эффективность загрузки данных. При загрузке данных в базовую таблицу данные в синхронных материализованных представлениях и базовой таблице обновляются синхронно. Если базовая таблица содержит `n` синхронных материализованных представлений, эффективность загрузки данных в базовую таблицу примерно такая же, как и загрузка данных в `n` таблиц.
- В настоящее время Selena не поддерживает создание нескольких синхронных материализованных представлений одновременно. Новое синхронное

> материализованное представление может быть создано только после завершения предыдущего.
> - Материализованное представление может быть создано только в default_catalog. Вы можете либо создать его с помощью default_catalog.database.mv, либо переключиться на default_catalog с помощью оператора `set catalog <default_catalog>`.

6.5.4 Проверка статуса построения синхронного материализованного представления

Создание синхронного материализованного представления является асинхронной операцией. Успешное выполнение CREATE MATERIALIZED VIEW указывает на то, что задача по созданию материализованного представления успешно отправлена. Вы можете просмотреть статус построения синхронного материализованного представления в базе данных через SHOW ALTER MATERIALIZED VIEW.

```
MySQL > SHOW ALTER MATERIALIZED VIEW\G
*************************** 1. row ***************************
          JobId: 12090
      TableName: sales_records
     CreateTime: 2022-08-25 19:41:10
   FinishedTime: 2022-08-25 19:41:39
  BaseIndexName: sales_records
RollupIndexName: store_amt
       RollupId: 12091
  TransactionId: 10
          State: FINISHED
            Msg:
       Progress: NULL
        Timeout: 86400
1 row in set (0.00 sec)
```

Раздел `RollupIndexName` указывает имя синхронного материализованного представления, а раздел `State` указывает, завершено ли построение.

6.5.5 Прямой запрос синхронного материализованного представления

Поскольку синхронное материализованное представление по сути является индексом базовой таблицы, а не физической таблицей, вы можете запросить синхронное

материализованное представление только с помощью подсказки `[_SYNC_MV_]`:

```
-- Не опускайте квадратные скобки [] в подсказке.
MySQL > SELECT * FROM store_amt [_SYNC_MV_];
+----------+----------+
| store_id | sale_amt |
+----------+----------+
|        2 |     6948 |
|        3 |     8734 |
|        1 |     4319 |
|        2 |     9515 |
|        3 |     4212 |
|        1 |     8573 |
+----------+----------+
```

> **ВНИМАНИЕ**
>
> В настоящее время Selena автоматически генерирует имена для столбцов в синхронном материализованном представлении, даже если вы указали для них псевдонимы.

6.5.6 Переписывание и ускорение запросов с помощью синхронного материализованного представления

Синхронное материализованное представление, которое вы создали, содержит полный набор предварительно вычисленных результатов в соответствии с запросом. Последующие запросы используют данные из него. Вы можете выполнить тот же запрос, чтобы протестировать время запроса, как вы делали при подготовке.

```
MySQL > SELECT store_id, SUM(sale_amt)
FROM sales_records
GROUP BY store_id;
+----------+-----------------+
| store_id | sum(`sale_amt`) |
+----------+-----------------+
|        2 |           16463 |
|        3 |           12946 |
|        1 |           12892 |
+----------+-----------------+
3 rows in set (0.01 sec)
```

Можно заметить, что время запроса сократилось до 0,01 секунд.

6.5.7 Проверка использования синхронного материализованного представления запросом

Выполните команду EXPLAIN снова, чтобы проверить, использует ли запрос синхронное материализованное представление.

```
MySQL > EXPLAIN SELECT store_id, SUM(sale_amt) FROM sales_records GROUP BY
store_id;
+-----------------------------------------------------------------------------
---+
| Explain String
|
+-----------------------------------------------------------------------------
---+
| PLAN FRAGMENT 0
|
|  OUTPUT EXPRS:3: store_id | 6: sum
|
|   PARTITION: UNPARTITIONED
|
|
|
|   RESULT SINK
|
|
|
|   4:EXCHANGE
|
|
|
| PLAN FRAGMENT 1
|
|  OUTPUT EXPRS:
|
|   PARTITION: HASH_PARTITIONED: 3: store_id
|
|
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 04
|
|     UNPARTITIONED
|
|
|
|   3:AGGREGATE (merge finalize)
|
|   |  output: sum(6: sum)
|
|   |  group by: 3: store_id
```

```
|
|   |
|
|   2:EXCHANGE
|
|
|
| PLAN FRAGMENT 2
|
|  OUTPUT EXPRS:
|
|   PARTITION: RANDOM
|
|
|
|   STREAM DATA SINK
|
|     EXCHANGE ID: 02
|
|     HASH_PARTITIONED: 3: store_id
|
|
|
|   1:AGGREGATE (update serialize)
|
|   |  STREAMING
|
|   |  output: sum(5: sale_amt)
|
|   |  group by: 3: store_id
|
|   |
|
|   0:OlapScanNode
|
|      TABLE: sales_records
|
|      PREAGGREGATION: ON
|
|      partitions=1/1
|
|      rollup: store_amt
|
|      tabletRatio=10/10
|
|
```

```
tabletList=12092,12096,12100,12104,12108,12112,12116,12120,12124,12128 |
|      cardinality=6
|
|      avgRowSize=2.0
|
|      numNodes=0
|
+-----------------------------------------------------------------------
---+
45 rows in set (0.00 sec)
```

Можно заметить, что значение раздела `rollup` в профиле запроса теперь `store_amt`, что является синхронным материализованным представлением, которое вы создали. Это означает, что этот запрос использовал синхронное материализованное представление.

6.5.8 Отображение синхронных материализованных представлений

Вы можете выполнить DESC <tbl_name> ALL, чтобы проверить схему таблицы и ее подчиненных синхронных материализованных представлений.

```
MySQL > DESC sales_records ALL;
+---------------+---------------+-----------+--------+------+-------+------
---+-------+
| IndexName     | IndexKeysType | Field     | Type   | Null | Key   |
Default | Extra |
+---------------+---------------+-----------+--------+------+-------+------
---+-------+
| sales_records | DUP_KEYS      | record_id | INT    | Yes  | true  | NULL
|       |
|               |               | seller_id | INT    | Yes  | true  | NULL
|       |
|               |               | store_id  | INT    | Yes  | true  | NULL
|       |
|               |               | sale_date | DATE   | Yes  | false | NULL
| NONE  |
|               |               | sale_amt  | BIGINT | Yes  | false | NULL
| NONE  |
|               |               |           |        |      |       |
|       |
| store_amt     | AGG_KEYS      | store_id  | INT    | Yes  | true  | NULL
|       |
|               |               | sale_amt  | BIGINT | Yes  | false | NULL
| SUM   |
+---------------+---------------+-----------+--------+------+-------+------
---+-------+
8 rows in set (0.00 sec)
```

6.5.9 Удаление синхронного материализованного представления

При следующих обстоятельствах вам необходимо удалить синхронное материализованное представление:

- Вы создали неправильное материализованное представление и вам нужно удалить его до завершения построения.
- Вы создали слишком много материализованных представлений, что привело к значительному снижению производительности загрузки, и некоторые из материализованных представлений являются дубликатами.
- Частота использования соответствующих запросов низкая, и вы можете терпеть относительно высокую задержку запроса.

Удаление незавершенного синхронного материализованного представления

Вы можете удалить создаваемое синхронное материализованное представление, отменив выполняющуюся задачу создания. Сначала вам нужно получить идентификатор задачи `JobID` задачи создания материализованного представления, проверив статус построения материализованного представления. После получения идентификатора задачи вам необходимо отменить задачу создания с помощью команды CANCEL ALTER.

```
CANCEL ALTER TABLE ROLLUP FROM sales_records (12090);
```

Удаление существующего синхронного материализованного представления

Вы можете удалить существующее синхронное материализованное представление с помощью команды DROP MATERIALIZED VIEW.

```
DROP MATERIALIZED VIEW store_amt;
```

6.5.10 Лучшие практики

Точный подсчет уникальных значений

Следующий пример основан на таблице анализа рекламного бизнеса `advertiser_view_record`, которая записывает дату просмотра рекламы `click_time`, название рекламы `advertiser`, канал рекламы `channel` и идентификатор пользователя, просмотревшего рекламу `user_id`.

```
CREATE TABLE advertiser_view_record(
    click_time DATE,
    advertiser VARCHAR(10),
    channel VARCHAR(10),
    user_id INT
) distributed BY hash(click_time);
```

Анализ в основном сосредоточен на UV рекламы.

```
SELECT advertiser, channel, count(distinct user_id)
FROM advertiser_view_record
GROUP BY advertiser, channel;
```

Чтобы ускорить точный подсчет уникальных значений, вы можете создать синхронное материализованное представление на основе этой таблицы и использовать функцию bitmap_union для предварительной агрегации данных.

```
CREATE MATERIALIZED VIEW advertiser_uv AS
SELECT advertiser, channel, bitmap_union(to_bitmap(user_id))
FROM advertiser_view_record
GROUP BY advertiser, channel;
```

После создания синхронного материализованного представления подзапрос `count(distinct user_id)` в последующих запросах будет автоматически переписан как `bitmap_union_count (to_bitmap(user_id))`, чтобы они могли использовать синхронное материализованное представление.

Приблизительный подсчет уникальных значений

Используя таблицу `advertiser_view_record` из примера выше. Чтобы ускорить приблизительный подсчет уникальных значений, вы можете создать синхронное материализованное представление на основе этой таблицы и использовать функцию hll_union() для предварительной агрегации данных.

```
CREATE MATERIALIZED VIEW advertiser_uv2 AS
SELECT advertiser, channel, hll_union(hll_hash(user_id))
FROM advertiser_view_record
GROUP BY advertiser, channel;
```

Установка дополнительных ключей сортировки

Предположим, что базовая таблица `tableA` содержит столбцы `k1`, `k2` и `k3`, где только `k1` и `k2` являются ключами сортировки. Если запрос, включающий подзапрос `where k3=x`, должен быть ускорен, вы можете создать синхронное материализованное представление с `k3` в качестве первого столбца.

```
CREATE MATERIALIZED VIEW k3_as_key AS
SELECT k3, k2, k1
FROM tableA
```

6.5.11 Соответствие агрегатных функций

Когда запрос выполняется с синхронным материализованным представлением, исходный запрос автоматически переписывается и используется для запроса промежуточных результатов, хранящихся в синхронном материализованном представлении. Следующая таблица показывает соответствие между агрегатной функцией в исходном запросе и агрегатной функцией, используемой для построения синхронного материализованного представления. Вы можете выбрать соответствующую агрегатную функцию для построения синхронного материализованного представления в соответствии с вашим бизнес-сценарием.

агрегатная функция в исходном запросе	агрегатная функция материализованного представления
sum	sum
min	min
max	max
count	count
bitmap_union, bitmap_union_count, count(distinct)	bitmap_union
hll_raw_agg, hll_union_agg, ndv, approx_count_distinct	hll_union

6.6 Asynchronous materialized views

6.6.1 Асинхронные материализованные представления

В этом разделе описывается, как понимать, создавать, использовать и управлять асинхронным материализованным представлением. Асинхронные материализованные представления поддерживаются начиная с Selena v1.5.2.

По сравнению с синхронными материализованными представлениями, асинхронные материализованные представления поддерживают соединение нескольких таблиц и больше агрегатных функций. Обновление асинхронных материализованных представлений может быть запущено вручную, регулярно через заданные интервалы или автоматически при изменении данных в базовой таблице. Вы также можете обновлять

некоторые партиции вместо всего материализованного представления, что значительно снижает стоимость обновления. Кроме того, асинхронные материализованные представления поддерживают различные сценарии перезаписи запросов, позволяя автоматическое и прозрачное ускорение запросов.

Для сценария и использования синхронных материализованных представлений (Rollup) см. Синхронное материализованное представление (Rollup).

Обзор

Приложения в базах данных часто выполняют сложные запросы к большим таблицам. Такие запросы включают соединения нескольких таблиц и агрегации таблиц, содержащих миллиарды строк. Обработка этих запросов может быть дорогостоящей с точки зрения системных ресурсов и времени, необходимого для вычисления результатов.

Асинхронные материализованные представления в Selena предназначены для решения этих проблем. Асинхронное материализованное представление — это специальная физическая таблица, которая хранит предварительно вычисленные результаты запроса из одной или нескольких базовых таблиц. Когда вы выполняете сложные запросы к базовой таблице, Selena возвращает предварительно вычисленные результаты из соответствующих материализованных представлений для обработки этих запросов. Таким образом, производительность запросов может быть улучшена, поскольку избегаются повторяющиеся сложные вычисления. Эта разница в производительности может быть значительной, когда запрос выполняется часто или достаточно сложен.

Кроме того, асинхронные материализованные представления особенно полезны для построения математических моделей на основе вашего хранилища данных. Таким образом, вы можете предоставить единую спецификацию данных для приложений верхнего уровня, защитить базовую реализацию или обеспечить безопасность необработанных данных базовых таблиц.

Понимание материализованных представлений в Selena

Selena v1.5.2 и более ранние версии предоставляли синхронное материализованное представление, которое можно было построить только на одной таблице. Синхронные материализованные представления, или Rollup, сохраняют большую свежесть данных и более низкие затраты на обновление. Однако по сравнению с асинхронными материализованными представлениями, поддерживаемыми начиная с v1.5.2, синхронные материализованные представления имеют много ограничений. У вас есть ограниченный

выбор операторов агрегации, когда вы хотите построить синхронное материализованное представление для ускорения или перезаписи ваших запросов.

В следующей таблице сравниваются асинхронные материализованные представления (ASYNC MV) и синхронные материализованные представления (SYNC MV) в Selena с точки зрения поддерживаемых функций:

	Агрегация одной таблицы	Соединение нескольких таблиц	Перезапись запросов	Стратегия обновления	Базовая таблица
ASYNC MV	Да	Да	Да	• Асинхронное обновление • Ручное обновление	Несколько таблиц из: • Default catalog • Внешние каталоги (v1.5.2) • Существующие материализованные представления (v1.5.2) • Существующие представления (v1.5.2)
SYNC MV (Rollup)	Ограниченный выбор агрегатных функций	Нет	Да	Синхронное обновление при загрузке данных	Одна таблица в default catalog

Основные понятия

- **Базовая таблица**

  Базовые таблицы являются управляющими таблицами материализованного представления.

  Для асинхронных материализованных представлений Selena базовыми таблицами могут быть собственные таблицы Selena в default catalog, таблицы во внешних каталогах (поддерживается с v1.5.2) или даже существующие асинхронные материализованные представления (поддерживается с v1.5.2) и представления (поддерживается с v1.5.2). Selena поддерживает создание асинхронных материализованных представлений на всех типах таблиц Selena.

- **Обновление**

При создании асинхронного материализованного представления его данные отражают только состояние базовых таблиц на тот момент. Когда данные в базовых таблицах изменяются, вам нужно обновить материализованное представление, чтобы синхронизировать изменения.

В настоящее время Selena поддерживает две общие стратегии обновления:

- ASYNC: Режим асинхронного обновления. Материализованные представления могут обновляться автоматически при изменении данных в базовой таблице или через регулярные интервалы на основе указанных интервалов.
- MANUAL: Режим ручного обновления. Материализованное представление не будет автоматически обновляться. Задачи обновления могут быть запущены только вручную пользователями.

- **Перезапись запросов**

  Перезапись запросов означает, что при выполнении запроса к базовым таблицам с построенными на них материализованными представлениями система автоматически определяет, могут ли быть повторно использованы предварительно вычисленные результаты в материализованном представлении для запроса. Если они могут быть повторно использованы, система загрузит данные напрямую из соответствующего материализованного представления, чтобы избежать трудоемких и ресурсоемких вычислений или соединений.

  Начиная с v1.5.2, Selena поддерживает автоматическую, прозрачную перезапись запросов на основе асинхронных материализованных представлений типа SPJG. Материализованные представления типа SPJG относятся к материализованным представлениям, план которых включает только операторы типа Scan, Filter, Project и Aggregate.

  > **ПРИМЕЧАНИЕ**
  >
  > Асинхронные материализованные представления, созданные на базовых таблицах в JDBC catalog или Hudi catalog, не поддерживают перезапись запросов.

Определение, когда создавать материализованное представление

Вы можете создать асинхронное материализованное представление, если у вас есть следующие требования в вашей среде хранилища данных:

- **Ускорение запросов с повторяющимися агрегатными функциями**

  Предположим, что большинство запросов в вашем хранилище данных включают один и тот же подзапрос с агрегатной функцией, и эти запросы потребляли огромную долю ваших вычислительных ресурсов. На основе этого подзапроса вы можете создать асинхронное материализованное представление, которое будет вычислять и хранить все результаты подзапроса. После создания материализованного представления Selena переписывает все запросы, содержащие подзапрос, загружает промежуточные результаты, хранящиеся в материализованном представлении, и таким образом ускоряет эти запросы.

- **Регулярное соединение нескольких таблиц**

  Предположим, что вам нужно регулярно соединять несколько таблиц в вашем хранилище данных, чтобы создать новую широкую таблицу. Вы можете построить асинхронное материализованное представление для этих таблиц и установить стратегию обновления ASYNC, которая запускает задачи обновления через фиксированный временной интервал. После создания материализованного представления результаты запроса возвращаются напрямую из материализованного представления, и таким образом избегается задержка, вызванная операциями JOIN.

- **Послойное хранилище данных**

  Предположим, что ваше хранилище данных содержит массу необработанных данных, и запросы в нем требуют сложного набора операций ETL. Вы можете построить несколько уровней асинхронных материализованных представлений для стратификации данных в вашем хранилище данных и таким образом разбить запрос на серию простых подзапросов. Это может значительно сократить повторяющиеся вычисления и, что более важно, помочь вашему администратору баз данных легко и эффективно выявлять проблемы. Кроме того, послойное хранилище данных помогает разделить необработанные данные и статистические данные, защищая безопасность конфиденциальных необработанных данных.

- **Ускорение запросов в озерах данных**

  Запросы к озеру данных могут быть медленными из-за задержки сети и пропускной способности объектного хранилища. Вы можете повысить производительность запросов, построив асинхронное материализованное представление поверх озера

данных. Более того, Selena может интеллектуально переписывать запросы для использования существующих материализованных представлений, избавляя вас от необходимости изменять запросы вручную.

Для конкретных случаев использования асинхронных материализованных представлений обратитесь к следующему содержанию:

- Моделирование данных
- Перезапись запросов
- Ускорение запросов к озерам данных

Создание асинхронного материализованного представления

Асинхронные материализованные представления Selena могут быть созданы на следующих базовых таблицах:

- Собственные таблицы Selena (поддерживаются все типы таблиц Selena)
- Таблицы во внешних каталогах
- Существующие асинхронные материализованные представления (с v1.5.2)
- Существующие представления (с v1.5.2)

Прежде чем начать

Следующие примеры включают две базовые таблицы в default catalog:

- Таблица `goods` записывает идентификатор товара `item_id1`, название товара `item_name` и цену товара `price`.
- Таблица `order_list` записывает идентификатор заказа `order_id`, идентификатор клиента `client_id`, идентификатор товара `item_id2` и дату заказа `order_date`.

Столбец `goods.item_id1` эквивалентен столбцу `order_list.item_id2`.

Выполните следующие инструкции для создания таблиц и вставки данных в них:

```
CREATE TABLE goods(
    item_id1          INT,
    item_name         STRING,
    price             FLOAT
) DISTRIBUTED BY HASH(item_id1);

INSERT INTO goods
VALUES
    (1001,"apple",6.5),
    (1002,"pear",8.0),
    (1003,"potato",2.2);

CREATE TABLE order_list(
    order_id          INT,
    client_id         INT,
    item_id2          INT,
    order_date        DATE
) DISTRIBUTED BY HASH(order_id);

INSERT INTO order_list
VALUES
    (10001,101,1001,"2022-03-13"),
    (10001,101,1002,"2022-03-13"),
    (10002,103,1002,"2022-03-13"),
    (10002,103,1003,"2022-03-14"),
    (10003,102,1003,"2022-03-14"),
    (10003,102,1001,"2022-03-14");
```

Сценарий в следующем примере требует частых вычислений суммы каждого заказа. Он требует частых соединений двух базовых таблиц и интенсивного использования агрегатной функции `sum()`. Кроме того, бизнес-сценарий требует обновления данных с интервалом в один день.

Инструкция запроса следующая:

```
SELECT
    order_id,
    sum(goods.price) as total
FROM order_list INNER JOIN goods ON goods.item_id1 = order_list.item_id2
GROUP BY order_id;
```

Создание материализованного представления

Вы можете создать материализованное представление на основе определенной инструкции запроса, используя CREATE MATERIALIZED VIEW.

На основе таблиц `goods`, `order_list` и приведенной выше инструкции запроса следующий пример создает материализованное представление `order_mv` для анализа суммы каждого заказа. Материализованное представление настроено на обновление с интервалом в один день.

```
CREATE MATERIALIZED VIEW order_mv
DISTRIBUTED BY HASH(`order_id`)
REFRESH ASYNC START('2022-09-01 10:00:00') EVERY (interval 1 day)
AS SELECT
    order_list.order_id,
    sum(goods.price) as total
FROM order_list INNER JOIN goods ON goods.item_id1 = order_list.item_id2
GROUP BY order_id;
```

ПРИМЕЧАНИЕ

- При создании асинхронного материализованного представления вы должны указать либо стратегию распределения данных, либо стратегию обновления материализованного представления, либо и то, и другое.
- Вы можете установить различные стратегии партиционирования и баскетинга для асинхронного материализованного представления, отличные от базовых таблиц, но вы должны включить ключи партиций и ключи баскетов материализованных представлений в инструкцию запроса, используемую для создания материализованного представления.
- Асинхронные материализованные представления поддерживают стратегию динамического партиционирования с более длинным интервалом. Например, если базовая таблица партиционирована с интервалом в один день, вы можете установить материализованное представление для партиционирования с интервалом в один месяц.
- Начиная с v1.5.2, Selena поддерживает создание асинхронных материализованных представлений со стратегией List Partitioning.
    - Вы можете создавать материализованные представления с list-партиционированием на основе таблиц, созданных со стратегией List

Partitioning или Expression partitioning.
  - В настоящее время вы можете указать только один Partition Key при создании материализованных представлений со стратегией List Partitioning. Вы должны выбрать один Partition Key, если базовая таблица имеет более одного Partition Key.
  - Поведение обновления и логика перезаписи запросов материализованных представлений со стратегией List Partitioning согласуются с теми, которые имеют стратегию Range Partitioning.
- Инструкция запроса, используемая для создания материализованного представления, не поддерживает случайные функции, включая rand(), random(), uuid() и sleep().
- Асинхронные материализованные представления поддерживают различные типы данных. Для получения дополнительной информации см. CREATE MATERIALIZED VIEW - Поддерживаемые типы данных.
- По умолчанию выполнение инструкции CREATE MATERIALIZED VIEW немедленно запускает задачу обновления, которая может потреблять определенную долю системных ресурсов. Если вы хотите отложить задачу обновления, вы можете добавить параметр REFRESH DEFERRED в свою инструкцию CREATE MATERIALIZED VIEW.
- Материализованное представление может быть создано только в default_catalog. Вы можете создать его с помощью default_catalog.database.mv или переключиться на default_catalog с помощью инструкции `set catalog <default_catalog>`.

- **О механизмах обновления асинхронных материализованных представлений**

  В настоящее время Selena поддерживает две стратегии обновления ON DEMAND: MANUAL refresh и ASYNC refresh.

  В Selena v1.5.2 асинхронные материализованные представления дополнительно поддерживают различные механизмы асинхронного обновления для контроля стоимости обновления и повышения скорости успеха:

  - Если MV имеет много больших партиций, каждое обновление может потреблять большое количество ресурсов. В v1.5.2 Selena поддерживает разделение задач обновления. Вы можете указать максимальное количество партиций для обновления, и Selena выполняет обновление пакетами, с размером пакета

меньшим или равным указанному максимальному количеству партиций. Эта функция обеспечивает стабильное обновление больших асинхронных материализованных представлений, повышая стабильность и надежность моделирования данных.
  - Вы можете указать срок жизни (TTL) для партиций асинхронного материализованного представления, уменьшая размер хранилища, занимаемый материализованным представлением.
  - Вы можете указать диапазон обновления для обновления только последних нескольких партиций, уменьшая накладные расходы на обновление.
  - Вы можете указать базовые таблицы, где изменения данных не будут автоматически запускать обновление соответствующего материализованного представления.
  - Вы можете назначить группу ресурсов задаче обновления.

Для получения дополнительной информации см. раздел **PROPERTIES** в CREATE MATERIALIZED VIEW - Параметры. Вы также можете изменить механизмы существующего асинхронного материализованного представления, используя ALTER MATERIALIZED VIEW.

> **ВНИМАНИЕ**
>
> Чтобы предотвратить полное обновление операций от исчерпания системных ресурсов и вызова сбоев задач, рекомендуется создавать партиционированные материализованные представления на основе партиционированных базовых таблиц. Это гарантирует, что когда происходят обновления данных в партиции базовой таблицы, обновляется только соответствующая партиция материализованного представления, а не все материализованное представление. Для получения дополнительной информации см. Моделирование данных с материализованными представлениями - Партиционированное моделирование.

- **О вложенных материализованных представлениях**

  Selena v1.5.2 поддерживает создание вложенных асинхронных материализованных представлений. Вы можете строить асинхронные материализованные представления на основе существующих асинхронных материализованных представлений. Стратегия обновления для каждого материализованного представления не влияет на материализованные представления на верхних или нижних уровнях. В настоящее

время Selena не ограничивает количество уровней вложенности. В производственной среде мы рекомендуем, чтобы количество уровней вложенности не превышало TPEX.

- **О материализованных представлениях внешнего каталога**

  Selena поддерживает построение асинхронных материализованных представлений на основе Hive Catalog (с v1.5.2), Hudi Catalog (с v1.5.2), Iceberg Catalog (с v1.5.2) и JDBC Catalog (с v1.5.2). Создание материализованного представления во внешних каталогах похоже на создание асинхронного материализованного представления в default catalog, но с некоторыми ограничениями использования. Для получения дополнительной информации см. Ускорение запросов к озерам данных с материализованными представлениями.

Ручное обновление асинхронного материализованного представления

Вы можете обновить асинхронное материализованное представление независимо от его стратегии обновления с помощью REFRESH MATERIALIZED VIEW. Selena v1.5.2 поддерживает обновление определенных партиций асинхронного материализованного представления путем указания имен партиций. Selena v1.5.2 поддерживает синхронный вызов задачи обновления, и инструкция SQL возвращается только тогда, когда задача успешно выполнена или не удалась.

```
-- Обновление материализованного представления через асинхронный вызов (по умолчанию).
REFRESH MATERIALIZED VIEW order_mv;
-- Обновление материализованного представления через синхронный вызов.
REFRESH MATERIALIZED VIEW order_mv WITH SYNC MODE;
```

Вы можете отменить задачу обновления, отправленную через асинхронный вызов, используя CANCEL REFRESH MATERIALIZED VIEW.

Прямой запрос к асинхронному материализованному представлению

Созданное вами асинхронное материализованное представление по сути является физической таблицей, которая содержит полный набор предварительно вычисленных результатов в соответствии с инструкцией запроса. Следовательно, вы можете напрямую

запрашивать материализованное представление после того, как материализованное представление будет обновлено в первый раз.

```
MySQL > SELECT * FROM order_mv;
+----------+--------------------+
| order_id | total              |
+----------+--------------------+
|    10001 |               14.5 |
|    10002 | 10.200000047683716 |
|    10003 |  8.700000047683716 |
+----------+--------------------+
3 rows in set (0.01 sec)
```

> **ПРИМЕЧАНИЕ**
>
> Вы можете напрямую запрашивать асинхронное материализованное представление, но результаты могут быть несогласованными с тем, что вы получаете из запроса к его базовым таблицам.

Перезапись и ускорение запросов с помощью асинхронного материализованного представления

Selena v1.5.2 поддерживает автоматическую и прозрачную перезапись запросов на основе асинхронных материализованных представлений типа SPJG. Перезапись запросов материализованных представлений типа SPJG включает перезапись запросов к одной таблице, перезапись запросов Join, перезапись запросов агрегации, перезапись запросов Union и перезапись запросов на основе вложенных материализованных представлений. Начиная с v1.5.2, Selena дополнительно поддерживает текстовую перезапись материализованных представлений. Для получения дополнительной информации см. Перезапись запросов с материализованными представлениями.

В настоящее время Selena поддерживает перезапись запросов к асинхронным материализованным представлениям, которые созданы в default catalog или внешнем каталоге, таком как Hive catalog, Hudi catalog или Iceberg catalog. При запросе данных в default catalog Selena обеспечивает строгую согласованность результатов между переписанным запросом и исходным запросом, исключая материализованные представления, данные которых несовместимы с базовой таблицей. Когда данные в материализованном представлении устаревают, материализованное представление не будет использоваться в качестве кандидата материализованного представления. При

запросе данных во внешних каталогах Selena не обеспечивает строгую согласованность результатов, поскольку Selena не может воспринимать изменения данных во внешних каталогах. Для получения дополнительной информации об асинхронных материализованных представлениях, созданных на основе внешнего каталога, см. Ускорение запросов к озерам данных с материализованными представлениями.

> **ПРИМЕЧАНИЕ**
>
> Асинхронные материализованные представления, созданные на базовых таблицах в JDBC catalog, не поддерживают перезапись запросов.

Управление асинхронным материализованным представлением

Изменение асинхронного материализованного представления

Вы можете изменить свойство асинхронного материализованного представления, используя ALTER MATERIALIZED VIEW.

- Активировать неактивное материализованное представление.

```
ALTER MATERIALIZED VIEW order_mv ACTIVE;
```

- Переименовать асинхронное материализованное представление.

```
ALTER MATERIALIZED VIEW order_mv RENAME order_total;
```

- Изменить интервал обновления асинхронного материализованного представления на 2 дня.

```
ALTER MATERIALIZED VIEW order_mv REFRESH ASYNC EVERY(INTERVAL 2 DAY);
```

Показать асинхронные материализованные представления

Вы можете просмотреть асинхронные материализованные представления в вашей базе данных, используя SHOW MATERIALIZED VIEWS или запрашивая системное представление метаданных в Information Schema.

- Проверить все асинхронные материализованные представления в вашей базе данных.

```
SHOW MATERIALIZED VIEWS;
```

- Проверить конкретное асинхронное материализованное представление.

```
SHOW MATERIALIZED VIEWS WHERE NAME = "order_mv";
```

- Проверить конкретные асинхронные материализованные представления путем сопоставления имени.

```
SHOW MATERIALIZED VIEWS WHERE NAME LIKE "order%";
```

- Проверить все асинхронные материализованные представления, запросив представление метаданных `materialized_views` в Information Schema. Для получения дополнительной информации см. information_schema.materialized_views.

```
SELECT * FROM information_schema.materialized_views;
```

Проверка определения асинхронного материализованного представления

Вы можете проверить запрос, используемый для создания асинхронного материализованного представления, с помощью SHOW CREATE MATERIALIZED VIEW.

```
SHOW CREATE MATERIALIZED VIEW order_mv;
```

Проверка статуса выполнения асинхронного материализованного представления

Вы можете проверить статус выполнения (создание или обновление) асинхронного материализованного представления, запросив `tasks` и `task_runs` в Information Schema.

Следующий пример проверяет статус выполнения материализованного представления, которое было создано последним:

1. Проверьте `TASK_NAME` самой последней задачи в таблице `tasks`.

```
mysql> select * from information_schema.tasks  order by CREATE_TIME
desc limit 1\G;
*************************** 1. row ***************************
  TASK_NAME: mv-59299
CREATE_TIME: 2022-12-12 17:33:51
  SCHEDULE: MANUAL
  DATABASE: ssb_1
DEFINITION: insert overwrite hive_mv_lineorder_flat_1 SELECT
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_orderkey`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_linenumber`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_custkey`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_partkey`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_orderpriority`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_ordtotalprice`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_revenue`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`p_mfgr`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`s_nation`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`c_city`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`c_nation`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_orderdate`
FROM `hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`
WHERE
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_orderdate` =
'1997-01-01'
EXPIRE_TIME: NULL
1 row in set (0.02 sec)
```

2. Проверьте статус выполнения в таблице `task_runs`, используя найденный `TASK_NAME`.

```
mysql> select * from information_schema.task_runs where task_name='mv-
59299' order by CREATE_TIME\G
*************************** 1. row ***************************
    QUERY_ID: d9cef11f-7a00-11ed-bd90-00163e14767f
    TASK_NAME: mv-59299
  CREATE_TIME: 2022-12-12 17:39:19
  FINISH_TIME: 2022-12-12 17:39:22
        STATE: SUCCESS
    DATABASE: ssb_1
  DEFINITION: insert overwrite hive_mv_lineorder_flat_1 SELECT
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_orderkey`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_linenumber`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_custkey`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_partkey`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_orderpriority`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_ordtotalprice`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_revenue`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`p_mfgr`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`s_nation`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`c_city`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`c_nation`,
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_orderdate`
FROM `hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`
WHERE
`hive_ci`.`dla_scan`.`lineorder_flat_1000_1000_orc`.`lo_orderdate` =
'1997-01-01'
  EXPIRE_TIME: 2022-12-15 17:39:19
  ERROR_CODE: 0
ERROR_MESSAGE: NULL
    PROGRESS: 100%
2 rows in set (0.02 sec)
```

Удаление асинхронного материализованного представления

Вы можете удалить асинхронное материализованное представление с помощью DROP MATERIALIZED VIEW.

```
DROP MATERIALIZED VIEW order_mv;
```

Связанные переменные сеанса

Следующие переменные управляют поведением асинхронного материализованного представления:

- `analyze_mv`: Выполнять ли и как анализировать материализованное представление после обновления. Допустимые значения — пустая строка (Не анализировать), `sample` (Выборочный сбор статистики) и `full` (Полный сбор статистики). По умолчанию `sample`.

6.6.2 Ссылки на асинхронные материализованные представления

- CREATE MATERIALIZED VIEW
- ALTER MATERIALIZED VIEW
- DROP MATERIALIZED VIEW
- SHOW MATERIALIZED VIEWS
- Information_schema.materialized_views

6.6.3 Use cases

Моделирование данных с помощью материализованных представлений

Этот раздел описывает, как выполнять моделирование данных с помощью асинхронных материализованных представлений Selena. Таким образом, вы можете значительно упростить ETL pipeline вашего хранилища данных и значительно улучшить качество данных и производительность запросов.

Обзор

Моделирование данных — это процесс очистки, наслоения, агрегации и связывания данных с рациональными методологиями. Оно может создать понятное представление сырых данных, которые слишком грубые, слишком сложные или слишком дорогие для прямого анализа, и предоставить действенные инсайты в данные.

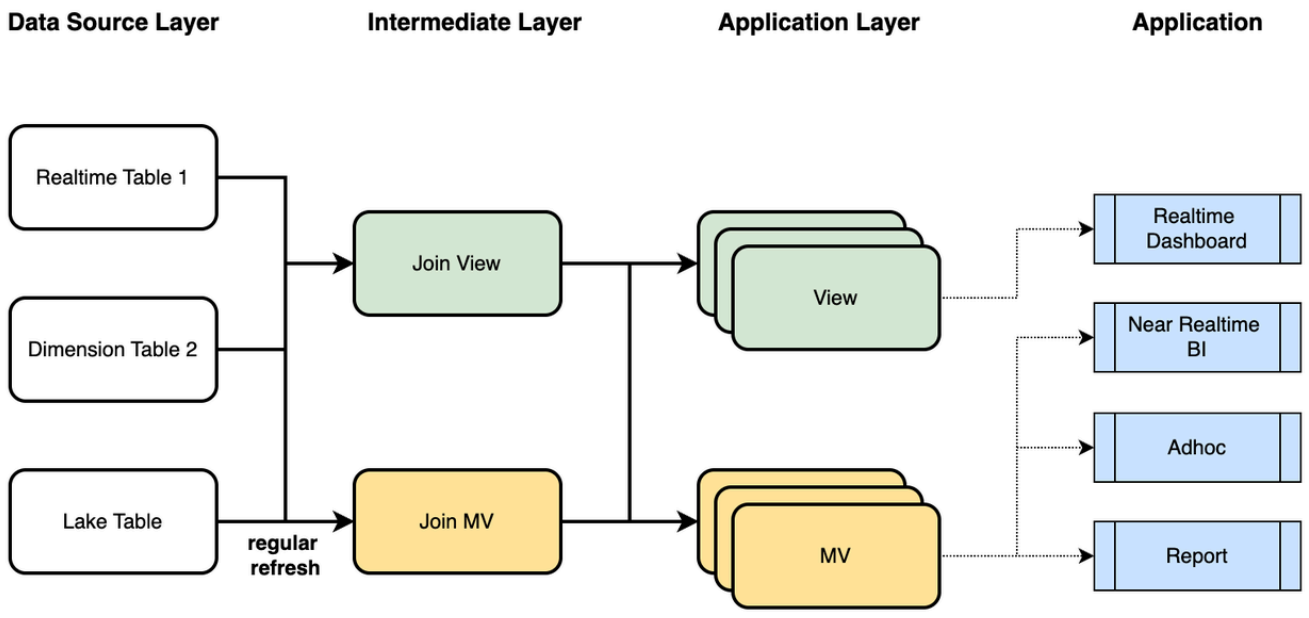
Однако, обычная проблема в реальном моделировании данных заключается в том, что процесс моделирования с трудом поспевает за темпами развития бизнеса, и трудно измерить возврат инвестиций для усилий по моделированию данных. Несмотря на то, что методологии моделирования просты, бизнес-эксперты должны иметь прочную основу в организации и управлении данными, что является сложным процессом. На ранних

стадиях бизнеса лица, принимающие решения, редко выделяют достаточные ресурсы для моделирования данных, и трудно увидеть ценность, которую может принести моделирование данных. Более того, бизнес-модели могут быстро меняться, и сами методологии моделирования нуждаются в итерации и эволюции. Поэтому многие аналитики данных склонны избегать моделирования и использовать сырые данные напрямую, что неизбежно приводит к проблемам качества данных и производительности запросов. Когда возникает необходимость в моделировании, становится трудно реструктурировать паттерны аналитики данных, которые уже были установлены, чтобы соответствовать моделям данных.

Использование материализованных представлений для моделирования данных может эффективно решить эти проблемы. Асинхронные материализованные представления Selena могут:

- **Упростить архитектуру хранилища данных**: Поскольку Selena может предоставить комплексный опыт управления данными, вам не нужно поддерживать другие системы обработки данных, экономя человеческие и системные ресурсы, потраченные на них.
- **Облегчить опыт моделирования данных**: Любой аналитик данных, обладающий только базовыми знаниями SQL, способен выполнять моделирование данных с помощью Selena. Моделирование данных больше не является исключительной прерогативой опытных инженеров данных.
- **Снизить сложность обслуживания**: Асинхронные материализованные представления Selena могут автоматически управлять связями происхождения и зависимостями между слоями данных, устраняя необходимость в целой платформе данных для выполнения этой задачи.

© ООО «Платформа Данных Селена», 2026



В реальных ситуациях вы можете выполнять моделирование данных, комбинируя использование представлений (логических представлений) Selena и асинхронных материализованных представлений следующим образом:

1. Используйте представления для связывания real-time данных с данными измерений и используйте материализованные представления для связывания исторических данных из озера данных с данными измерений. Выполните необходимую очистку данных и семантическое отображение, чтобы получить детальные данные для промежуточного слоя, которые отражают семантику, требуемую в ваших бизнес-сценариях.
2. На уровне приложения выполняйте вычисления Join, Aggregation, Union и Window, адаптированные для различных бизнес-сценариев. Это даст представления для real-time конвейеров и материализованные представления для near-real-time конвейеров.
3. На стороне приложения выберите подходящее аналитическое хранилище данных (ADS) для анализа запросов на основе ваших требований к своевременности и производительности. Эти ADS могут обслуживать real-time дашборды, near-real-time BI, ad-hoc запросы и запланированные отчеты.

В течение этого процесса вы будете использовать несколько встроенных возможностей Selena, которые будут подробно описаны в следующем разделе.

Возможности асинхронного материализованного представления

Асинхронное материализованное представление Selena обладает следующими атомарными функциями, которые могут помочь в моделировании данных:

- **Автоматическое обновление**: После загрузки данных в базовые таблицы материализованные представления могут автоматически обновляться. Вам не нужно поддерживать задачу планирования извне.
- **Partitioned обновление**: Near-real-time вычисления могут быть достигнуты через partitioned обновление материализованных представлений, построенных на таблицах с временными рядами.
- **Синергия с представлениями**: Вы можете достичь многоуровневого моделирования, используя материализованные представления и логические представления, тем самым обеспечивая повторное использование промежуточного слоя и упрощение моделей данных.
- **Изменение схемы**: Вы можете изменить результаты вычислений с помощью простых операторов SQL, без необходимости изменять сложные data pipelines.

С этими функциями вы можете разработать комплексные и адаптивные модели данных для удовлетворения различных бизнес-потребностей и сценариев.

Автоматическое обновление

При создании асинхронного материализованного представления вы можете указать стратегию обновления, используя оператор REFRESH. В настоящее время Selena поддерживает следующие стратегии обновления асинхронных материализованных представлений:

- **Автоматическое обновление** (`REFRESH ASYNC`): Задачи обновления запускаются каждый раз, когда данные в базовых таблицах изменяются. Зависимости данных автоматически управляются материализованным представлением.
- **Запланированное обновление** (`REFRESH ASYNC EVERY (INTERVAL <refresh_interval>)`): Задачи обновления запускаются через регулярные интервалы, например, каждую минуту, день или месяц. Если нет изменений данных в базовых таблицах, задача обновления не будет запущена.
- **Ручное обновление** (`REFRESH MANUAL`): Задачи обновления запускаются только путем выполнения REFRESH MATERIALIZED VIEW вручную. Эта стратегия обновления может использоваться, когда вы поддерживаете внешний framework планирования для запуска задач обновления.

Синтаксис:

```
CREATE MATERIALIZED VIEW <name>
REFRESH
    [ ASYNC |
      ASYNC [START <time>] EVERY(<interval>) |
      MANUAL
    ]
AS <query>
```

Partitioned обновление

При создании асинхронного материализованного представления вы можете указать оператор PARTITION BY для связывания partitions базовой таблицы с partitions материализованного представления, достигая таким образом обновления на уровне partition.

- `PARTITION BY <column>`: Вы можете ссылаться на тот же столбец partitioning для базовой таблицы и материализованного представления. В результате базовая таблица и материализованное представление разбиты на partition с одинаковой гранулярностью.
- `PARTITION BY date_trunc(<column>)`: Вы можете использовать функцию date_trunc для назначения другой стратегии partition (на уровне гранулярности) для материализованного представления путем усечения единицы времени.
- `PARTITION BY { time_slice | date_slice }(<column>)`: По сравнению с date_trunc, time_slice и date_slice предлагают более гибкие настройки гранулярности времени, позволяя более точный контроль над partitioning на основе времени.

Синтаксис:

```
CREATE MATERIALIZED VIEW <name>
REFRESH ASYNC
PARTITION BY
    [
        <base_table_column> |
        date_trunc(<granularity>, <base_table_column>) |
        time_slice(<base_table_column>, <granularity>) |
        date_slice(<base_table_column>, <granularity>)
    ]
AS <query>
```

Синергия с представлениями

- Материализованные представления могут быть созданы на основе представлений. В этом случае, когда базовые таблицы, на которые ссылается представление, претерпевают изменения данных, материализованное представление может автоматически обновляться.
- Вы также можете создавать материализованные представления на основе других материализованных представлений, обеспечивая многоуровневые каскадные механизмы обновления.
- Представления могут быть созданы на основе материализованных представлений, которые эквивалентны обычным таблицам.

Изменение схемы

- Вы можете выполнить атомарный обмен между двумя асинхронными материализованными представлениями, используя оператор ALTER MATERIALIZED VIEW SWAP. Это позволяет создать новое материализованное представление с добавленными столбцами или измененными типами столбцов, а затем заменить старое им.
- Определение представлений может быть напрямую изменено с помощью оператора ALTER VIEW.
- Обычные таблицы в Selena могут быть изменены либо с помощью операций SWAP, либо ALTER.
- Кроме того, когда есть изменения в базовых таблицах (которыми могут быть материализованные представления, представления или обычные таблицы), это запускает каскадные изменения в соответствующих материализованных представлениях.

Многоуровневое моделирование

Во многих реальных бизнес-сценариях существуют различные формы источников данных, включая real-time детальные данные, данные измерений и исторические данные из озер данных. С другой стороны, бизнес-требования требуют разнообразных аналитических методов, таких как real-time дашборды, near-real-time BI запросы, ad-hoc запросы и запланированные отчеты. Различные сценарии имеют разные требования — некоторые требуют гибкости, некоторые приоритизируют производительность, в то время как другие подчеркивают экономическую эффективность.

Очевидно, что одно решение не может адекватно удовлетворить такие универсальные требования. Selena может эффективно удовлетворить эти потребности, комбинируя использование представлений и материализованных представлений. Поскольку представления не поддерживают физических данных, каждый раз, когда запрашивается представление, запрос анализируется и выполняется в соответствии с определением представления. Для сравнения, материализованные представления, которые содержат предварительно вычисленные результаты, могут предотвратить накладные расходы повторного выполнения. Представления подходят для выражения бизнес-семантики и упрощения сложности SQL, но они не могут снизить затраты на выполнение запросов. Материализованные представления, с другой стороны, оптимизируют производительность запросов через предварительное вычисление и подходят для упрощения ETL pipelines.

Ниже приводится сводка различий между представлениями и материализованными представлениями:

	Представление	**Материализованное представление**
Случаи использования	Бизнес-моделирование, управление данными	Моделирование данных, прозрачное ускорение, интеграция озера данных
Стоимость хранения	Нет стоимости хранения	Стоимость хранения возникает при хранении предварительно вычисленных результатов
Стоимость обновления	Нет стоимости обновления	Стоимость обновления возникает при обновлении данных базовой таблицы
Преимущества производительности	Нет преимущества производительности	Ускорение запросов за счет повторного использования предварительно вычисленных результатов
Атрибут real-time данных	Возвращаются последние данные, потому что запросы к представлениям вычисляются в real-time.	Данные могут быть не актуальными, потому что результаты предварительно вычислены.
Зависимость	Представления становятся недействительными, если имена базовых таблиц изменены, потому что они ссылаются на базовые таблицы по имени.	Изменения имен базовых таблиц не влияют на доступность материализованных представлений, которые ссылаются на базовые таблицы по ID.
Синтаксис создания	CREATE VIEW	CREATE MATERIALIZED VIEW
Синтаксис изменения	ALTER VIEW	ALTER MATERIALIZED VIEW

Вы можете использовать следующие операторы для изменения ваших представлений, материализованных представлений и базовых таблиц:

```
-- Изменить таблицу.
ALTER TABLE <table_name> ADD COLUMN <column_desc>;

-- Обменять две таблицы.
ALTER TABLE <table1> SWAP WITH <table2>;

-- Изменить определение представления.
ALTER VIEW <view_name> AS <query>;

-- Обменять два материализованных представления
-- (путем обмена имени двух материализованных представлений без влияния на
данные внутри).
ALTER MATERIALIZED VIEW <mv1> SWAP WITH <mv2>;

-- Повторно активировать материализованное представление.
ALTER MATERIALIZED VIEW <mv_name> ACTIVE;
```

Изменения схемы следуют этим принципам:

- Операции Rename и Swap для таблицы устанавливают зависимые материализованные представления неактивными. Для операций Schema Change зависимые материализованные представления устанавливаются неактивными только когда операции Schema Change выполняются для столбцов базовой таблицы, на которые ссылаются материализованные представления.
- Если вы изменяете определение представления, зависимые материализованные представления устанавливаются неактивными.
- Если материализованное представление обменивается, любые вложенные материализованные представления, построенные на нем, устанавливаются неактивными.
- Статус неактивности каскадно распространяется вверх, пока не останется зависимостей материализованных представлений.
- Неактивные материализованные представления не могут обновляться или использоваться для автоматической перезаписи запросов.
- Неактивные материализованные представления все еще могут запрашиваться напрямую, но согласованность данных не гарантируется, пока они снова не станут активными.

В то время как согласованность данных неактивных материализованных представлений не может быть гарантирована, вы можете восстановить их функциональность, используя следующие методы:

- **Ручная активация**: Вы можете вручную восстановить неактивное материализованное представление, выполнив `ALTER MATERIALIZED VIEW <mv_name> ACTIVE`. Этот оператор пересоздаст материализованное представление на основе его исходного SQL определения. Обратите внимание, что SQL определение все еще должно быть действительным после изменений базовой схемы. В противном случае операция не удастся.
- **Активация перед обновлением**: Selena попытается активировать неактивное материализованное представление перед его обновлением.
- **Автоматическая активация**: Selena попытается автоматически активировать неактивные материализованные представления. Однако своевременность этого процесса не может быть гарантирована. Вы можете отключить эту функцию, выполнив `ADMIN SET FRONTEND CONFIG('enable_mv_automatic_active_check'='false')`. Эта функция доступна начиная с v1.5.2.

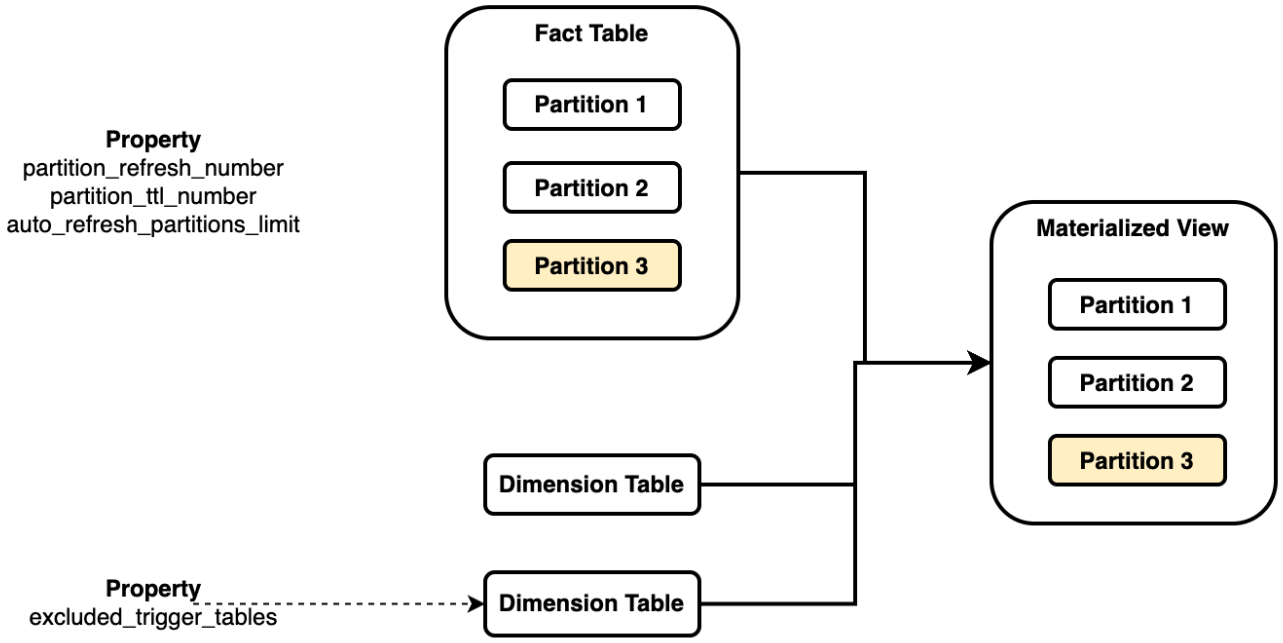
Partitioned моделирование

В дополнение к многоуровневому моделированию, partitioned моделирование также является важным аспектом моделирования данных. Моделирование данных часто включает связывание данных на основе бизнес-семантики и установку Time-To-Live (TTL) данных в соответствии с требованиями своевременности. Partitioned моделирование играет значительную роль в этом процессе.

Partitioned моделирование является важным аспектом моделирования данных, дополняя многоуровневое моделирование. Оно включает связывание данных на основе бизнес-семантики и установку Time-To-Live (TTL) для данных в соответствии с требованиями своевременности. Data partitioning играет значительную роль в этом процессе.

Различные способы связывания данных приводят к различным подходам к моделированию, таким как star schemas и snowflake schemas. Эти модели имеют что-то общее — все они используют таблицы фактов и таблицы измерений. Некоторые бизнес-сценарии требуют нескольких больших таблиц фактов, в то время как другие имеют дело со сложными таблицами измерений и отношениями между ними. Материализованные представления Selena поддерживают partition связывание для таблиц фактов, что

означает, что таблица фактов разбита на partition, и результаты join материализованного представления разбиты на partition таким же образом.



Как показано на рисунке выше, материализованное представление связывает таблицу фактов с несколькими таблицами измерений:

- Вам нужно ссылаться на ключ partition определенной базовой таблицы (обычно таблицы фактов) как на ключ partitioning материализованного представления (`PARTITION BY fact_tbl.col`), чтобы связать их стратегии partitioning. Каждое материализованное представление может быть связано только с одной базовой таблицей.
- Когда данные в partition ссылаемой таблицы изменяются, соответствующий partition в материализованном представлении обновляется, не влияя на другие partitions.
- Когда данные в не ссылаемых таблицах изменяются, все материализованное представление обновляется по умолчанию. Однако, вы можете выбрать игнорировать изменения данных в определенных не ссылаемых базовых таблицах, чтобы материализованное представление не обновлялось при изменении данных в этих таблицах.

Такое partition связывание поддерживает различные бизнес-сценарии:

- **Обновления таблицы фактов**: Вы можете разбить таблицу фактов на partition на детальном уровне, например, ежедневно или ежечасно. После обновления таблицы

фактов соответствующие partitions в материализованном представлении автоматически обновляются.

- **Обновления таблицы измерений**: Обычно обновления данных в таблицах измерений приводят к обновлению всех связанных результатов, что может быть дорогостоящим. Вы можете выбрать игнорировать обновления данных в некоторых таблицах измерений, чтобы избежать обновления всего материализованного представления, или вы можете указать временной диапазон, чтобы только partitions в пределах временного диапазона могли обновляться.
- **Автоматическое обновление внешней таблицы**: Во внешних источниках данных, таких как Apache Hive или Apache Iceberg, данные изменяются на уровне partition. Материализованные представления Selena могут подписываться на изменения во внешних каталогах на уровне partition и обновлять только соответствующий partition материализованного представления.
- **TTL**: При установке стратегии partitioning для материализованного представления вы можете установить количество последних partitions для сохранения, сохраняя таким образом только самые свежие данные. Это полезно в бизнес-сценариях, когда аналитики запрашивают только актуальные данные из определенного окна времени, и им не нужно сохранять все исторические данные.

Несколько параметров могут использоваться для контроля поведения обновления:

- `partition_refresh_number`: количество partitions для обновления в каждой операции обновления.
- `partition_refresh_strategy`: стратегия обновления, выбранная для каждой операции обновления.
- `excluded_trigger_tables`: таблица, изменения данных которой могут быть проигнорированы, чтобы избежать запуска автоматического обновления.
- `auto_refresh_partitions_limit`: количество partitions для обновления в каждой автоматической операции обновления.
- `excluded_refresh_tables`: Исключить таблицы, которые нужно обновить, обычно используется вместе с `excluded_trigger_tables`.

Для получения дополнительной информации см. CREATE MATERIALIZED VIEW.

В настоящее время partitioned материализованные представления имеют следующие ограничения:

- Вы можете построить partitioned материализованное представление только на основе partitioned таблицы.
- Вы можете использовать только столбцы типа DATE или DATETIME в качестве ключа partitioning. Тип данных STRING не поддерживается.
- Вы можете выполнять только partition roll-up, используя функции date_trunc, time_slice и date_slice.
- Вы можете указать только один столбец в качестве ключа partitioning. Несколько столбцов partitioning не поддерживаются.

Резюме

Использование асинхронных материализованных представлений Selena для моделирования данных предлагает преимущество упрощения управления pipeline и повышения эффективности и гибкости моделирования данных через декларативный язык моделирования.

Помимо моделирования данных, асинхронные материализованные представления Selena находят применение в различных сценариях, включающих прозрачное ускорение и интеграцию озера данных. Это способствует дальнейшему исследованию ценности данных и повышению эффективности данных.

Перезапись запросов с помощью материализованных представлений

Этот раздел описывает, как использовать асинхронные материализованные представления Selena для перезаписи и ускорения запросов.

Обзор

Асинхронное материализованное представление Selena использует широко принятый алгоритм прозрачной перезаписи запросов на основе формы SPJG (select-project-join-group-by). Без необходимости изменять оператор запроса, Selena может автоматически переписывать запросы к базовым таблицам в запросы к соответствующему материализованному представлению, которое содержит предварительно вычисленные результаты. В результате материализованные представления могут помочь вам значительно снизить вычислительные затраты и существенно ускорить выполнение запросов.

Функция перезаписи запросов на основе асинхронных материализованных представлений особенно полезна в следующих сценариях:

- **Предварительная агрегация метрик**

  Вы можете использовать материализованные представления для создания слоя предварительно агрегированных метрик, если вы имеете дело с данными высокой размерности.

- **Joins широких таблиц**

  Материализованные представления позволяют прозрачно ускорять запросы с joins нескольких больших широких таблиц в сложных сценариях.

- **Ускорение запросов в озере данных**

  Создание материализованного представления на основе внешнего каталога может легко ускорить запросы к данным в вашем озере данных.

> **NOTE**
>
> Асинхронные материализованные представления, созданные на базовых таблицах в JDBC catalog, не поддерживают перезапись запросов.

Функции

Автоматическая перезапись запросов на основе асинхронного материализованного представления Selena имеет следующие атрибуты:

- **Строгая согласованность данных**: Если базовые таблицы являются нативными таблицами, Selena гарантирует, что результаты, полученные через перезапись запросов на основе материализованного представления, согласованы с результатами, возвращаемыми из прямого запроса к базовым таблицам.
- **Staleness rewrite**: Selena поддерживает staleness rewrite, позволяя вам допустить определенный уровень устаревания данных для решения сценариев с частыми изменениями данных.
- **Joins нескольких таблиц**: Асинхронное материализованное представление Selena поддерживает различные типы joins, включая некоторые сложные сценарии join, такие как View Delta Joins и Derivable Joins, позволяя ускорять запросы в сценариях, включающих большие широкие таблицы.

- **Перезапись агрегации**: Selena может переписывать запросы с агрегациями для улучшения производительности отчетов.
- **Вложенное материализованное представление**: Selena поддерживает перезапись сложных запросов на основе вложенных материализованных представлений, расширяя область запросов, которые могут быть переписаны.
- **Union rewrite**: Вы можете объединить функцию Union rewrite с TTL (Time-to-Live) partitions материализованного представления для достижения разделения горячих и холодных данных, что позволяет запрашивать горячие данные из материализованных представлений, а исторические данные из базовой таблицы.
- **Материализованные представления на представлениях**: Вы можете ускорять запросы в сценариях с моделированием данных на основе представлений.
- **Материализованные представления на внешних каталогах**: Вы можете ускорять запросы в озерах данных.
- **Перезапись сложных выражений**: Может обрабатывать сложные выражения, включая вызовы функций и арифметические операции, удовлетворяя продвинутые аналитические и вычислительные требования.

Эти функции будут подробно рассмотрены в следующих разделах.

Перезапись Join

Selena поддерживает перезапись запросов с различными типами joins, включая Inner Join, Cross Join, Left Outer Join, Full Outer Join, Right Outer Join, Semi Join и Anti Join.

Следующий пример перезаписи запросов с joins. Создайте две базовые таблицы следующим образом:

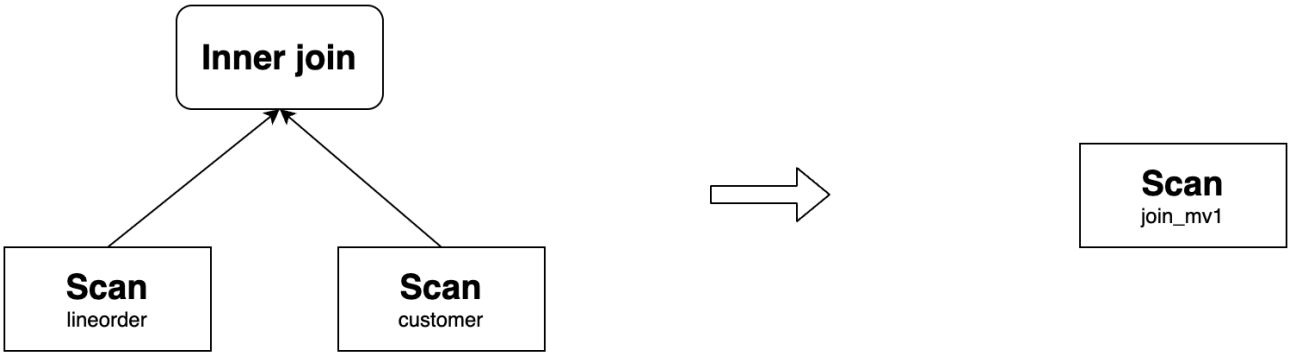
```
CREATE TABLE customer (
  c_custkey     INT(11)     NOT NULL,
  c_name        VARCHAR(26) NOT NULL,
  c_address     VARCHAR(41) NOT NULL,
  c_city        VARCHAR(11) NOT NULL,
  c_nation      VARCHAR(16) NOT NULL,
  c_region      VARCHAR(13) NOT NULL,
  c_phone       VARCHAR(16) NOT NULL,
  c_mktsegment  VARCHAR(11) NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(c_custkey)
DISTRIBUTED BY HASH(c_custkey) BUCKETS 12;

CREATE TABLE lineorder (
  lo_orderkey         INT(11) NOT NULL,
  lo_linenumber       INT(11) NOT NULL,
  lo_custkey          INT(11) NOT NULL,
  lo_partkey          INT(11) NOT NULL,
  lo_suppkey          INT(11) NOT NULL,
  lo_orderdate        INT(11) NOT NULL,
  lo_orderpriority    VARCHAR(16) NOT NULL,
  lo_shippriority     INT(11) NOT NULL,
  lo_quantity         INT(11) NOT NULL,
  lo_extendedprice    INT(11) NOT NULL,
  lo_ordtotalprice    INT(11) NOT NULL,
  lo_discount         INT(11) NOT NULL,
  lo_revenue          INT(11) NOT NULL,
  lo_supplycost       INT(11) NOT NULL,
  lo_tax              INT(11) NOT NULL,
  lo_commitdate       INT(11) NOT NULL,
  lo_shipmode         VARCHAR(11) NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(lo_orderkey)
DISTRIBUTED BY HASH(lo_orderkey) BUCKETS 48;
```

С указанными выше базовыми таблицами вы можете создать материализованное представление следующим образом:

```
CREATE MATERIALIZED VIEW join_mv1
DISTRIBUTED BY HASH(lo_orderkey)
AS
SELECT lo_orderkey, lo_linenumber, lo_revenue, lo_partkey, c_name,
c_address
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey;
```

Такое материализованное представление может переписать следующий запрос:

```
SELECT lo_orderkey, lo_linenumber, lo_revenue, c_name, c_address
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey;
```



Selena поддерживает перезапись запросов join со сложными выражениями, такими как арифметические операции, строковые функции, функции дат, выражения CASE WHEN и предикаты OR. Например, указанное выше материализованное представление может переписать следующий запрос:

```
SELECT
    lo_orderkey,
    lo_linenumber,
    (2 * lo_revenue + 1) * lo_linenumber,
    upper(c_name),
    substr(c_address, 3)
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey;
```

В дополнение к обычному сценарию, Selena дополнительно поддерживает перезапись запросов join в более сложных сценариях.

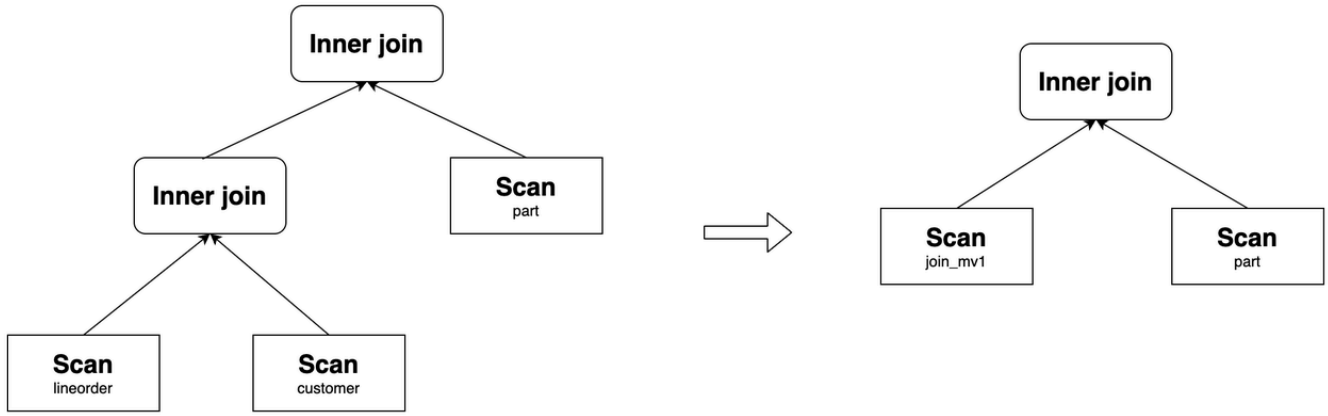
Перезапись Query Delta Join

Query Delta Join относится к сценарию, в котором таблицы, объединенные в запросе, являются надмножеством таблиц, объединенных в материализованном представлении. Например, рассмотрим следующий запрос, который включает joins трех таблиц: `lineorder`, `customer` и `part`. Если материализованное представление `join_mv1` содержит только join `lineorder` и `customer`, Selena может переписать запрос, используя `join_mv1`.

Пример:

```
SELECT lo_orderkey, lo_linenumber, lo_revenue, c_name, c_address, p_name
FROM
    lineorder INNER JOIN customer ON lo_custkey = c_custkey
    INNER JOIN part ON lo_partkey = p_partkey;
```

Его исходный план запроса и план после перезаписи следующие:



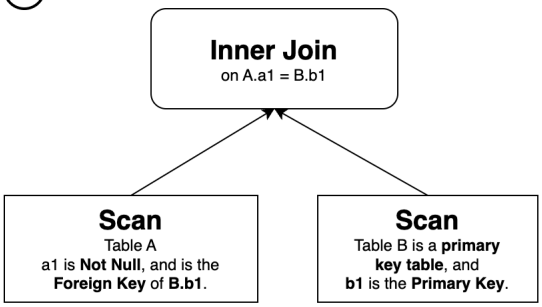
Перезапись View Delta Join

View Delta Join относится к сценарию, в котором таблицы, объединенные в запросе, являются подмножеством таблиц, объединенных в материализованном представлении. Эта функция обычно используется в сценариях, включающих большие широкие таблицы. Например, в контексте Star Schema Benchmark (SSB), вы можете создать материализованное представление, которое объединяет все таблицы для улучшения производительности запросов. Посредством тестирования было обнаружено, что

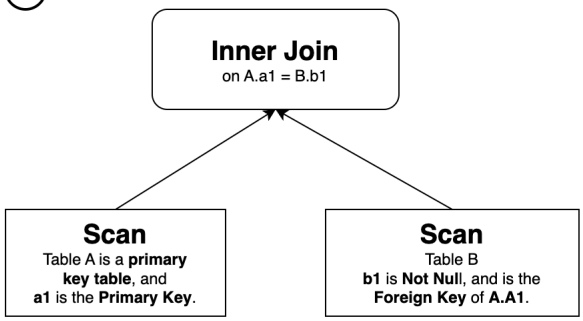
производительность запросов для joins нескольких таблиц может достичь того же уровня производительности, что и запрос к соответствующей большой широкой таблице после прозрачной перезаписи запросов через материализованное представление.

Чтобы выполнить перезапись View Delta Join, материализованное представление должно содержать join сохранения кардинальности 1:1, который не существует в запросе. Вот девять типов joins, которые считаются joins сохранения кардинальности, и удовлетворение любому из них позволяет перезапись View Delta Join:

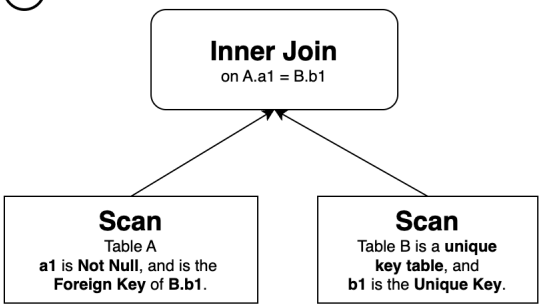
1


Inner Join
on A.a1 = B.b1
Scan
Table A
a1 is Not Null, and is the
Foreign Key of B.b1.
Scan
Table B is a primary
key table, and
b1 is the Primary Key.

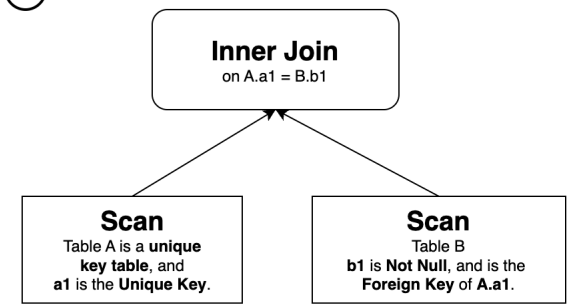
2


Inner Join
on A.a1 = B.b1
Scan
Table A is a primary
key table, and
a1 is the Primary Key.
Scan
Table B
b1 is Not Null, and is the
Foreign Key of A.A1.

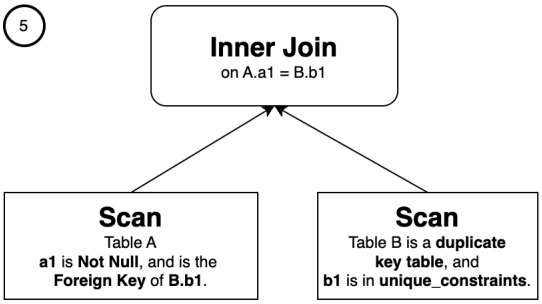
3


Inner Join
on A.a1 = B.b1
Scan
Table A
a1 is Not Null, and is the
Foreign Key of B.b1.
Scan
Table B is a unique
key table, and
b1 is the Unique Key.

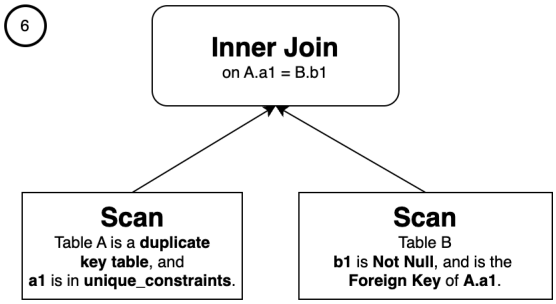
4


Inner Join
on A.a1 = B.b1
Scan
Table A is a unique
key table, and
a1 is the Unique Key.
Scan
Table B
b1 is Not Null, and is the
Foreign Key of A.a1.

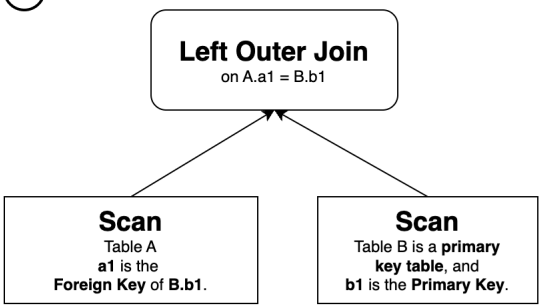
5


Inner Join
on A.a1 = B.b1
Scan
Table A
a1 is Not Null, and is the
Foreign Key of B.b1.
Scan
Table B is a duplicate
key table, and
b1 is in unique_constraints.

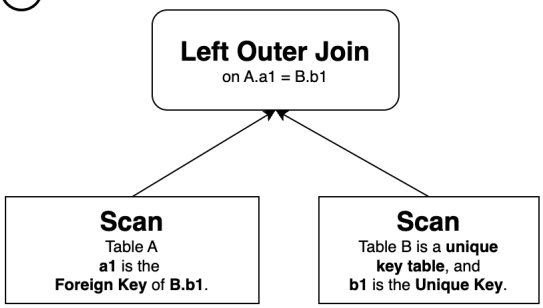
6


Inner Join
on A.a1 = B.b1
Scan
Table A is a duplicate
key table, and
a1 is in unique_constraints.
Scan
Table B
b1 is Not Null, and is the
Foreign Key of A.a1.

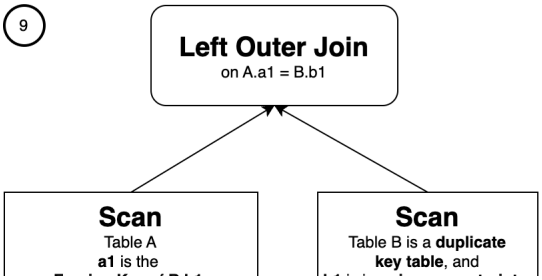
7


Left Outer Join
on A.a1 = B.b1
Scan
Table A
a1 is the
Foreign Key of B.b1.
Scan
Table B is a primary
key table, and
b1 is the Primary Key.

8


Left Outer Join
on A.a1 = B.b1
Scan
Table A
a1 is the
Foreign Key of B.b1.
Scan
Table B is a unique
key table, and
b1 is the Unique Key.

9


Left Outer Join
on A.a1 = B.b1
Scan
Table A
a1 is the
Scan
Table B is a duplicate
key table, and

Foreign Key of B.b1.

b1 is in unique_constraints.

Возьмем SSB тесты в качестве примера, создайте следующие базовые таблицы:

```
CREATE TABLE customer (
  c_custkey         INT(11)       NOT NULL,
  c_name            VARCHAR(26)   NOT NULL,
  c_address         VARCHAR(41)   NOT NULL,
  c_city            VARCHAR(11)   NOT NULL,
  c_nation          VARCHAR(16)   NOT NULL,
  c_region          VARCHAR(13)   NOT NULL,
  c_phone           VARCHAR(16)   NOT NULL,
  c_mktsegment      VARCHAR(11)   NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(c_custkey)
DISTRIBUTED BY HASH(c_custkey) BUCKETS 12
PROPERTIES (
"unique_constraints" = "c_custkey"   -- Указать уникальные ограничения.
);

CREATE TABLE dates (
  d_datekey          DATE          NOT NULL,
  d_date             VARCHAR(20)   NOT NULL,
  d_dayofweek        VARCHAR(10)   NOT NULL,
  d_month            VARCHAR(11)   NOT NULL,
  d_year             INT(11)       NOT NULL,
  d_yearmonthnum     INT(11)       NOT NULL,
  d_yearmonth        VARCHAR(9)    NOT NULL,
  d_daynuminweek     INT(11)       NOT NULL,
  d_daynuminmonth    INT(11)       NOT NULL,
  d_daynuminyear     INT(11)       NOT NULL,
  d_monthnuminyear   INT(11)       NOT NULL,
  d_weeknuminyear    INT(11)       NOT NULL,
  d_sellingseason    VARCHAR(14)   NOT NULL,
  d_lastdayinweekfl  INT(11)       NOT NULL,
  d_lastdayinmonthfl INT(11)       NOT NULL,
  d_holidayfl        INT(11)       NOT NULL,
  d_weekdayfl        INT(11)       NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(d_datekey)
DISTRIBUTED BY HASH(d_datekey) BUCKETS 1
PROPERTIES (
"unique_constraints" = "d_datekey"   -- Указать уникальные ограничения.
);

CREATE TABLE supplier (
  s_suppkey          INT(11)       NOT NULL,
  s_name             VARCHAR(26)   NOT NULL,
  s_address          VARCHAR(26)   NOT NULL,
```

```
  s_city              VARCHAR(11)   NOT NULL,
  s_nation            VARCHAR(16)   NOT NULL,
  s_region            VARCHAR(13)   NOT NULL,
  s_phone             VARCHAR(16)   NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(s_suppkey)
DISTRIBUTED BY HASH(s_suppkey) BUCKETS 12
PROPERTIES (
"unique_constraints" = "s_suppkey"   -- Указать уникальные ограничения.
);

CREATE TABLE part (
  p_partkey           INT(11)       NOT NULL,
  p_name              VARCHAR(23)   NOT NULL,
  p_mfgr              VARCHAR(7)    NOT NULL,
  p_category          VARCHAR(8)    NOT NULL,
  p_brand             VARCHAR(10)   NOT NULL,
  p_color             VARCHAR(12)   NOT NULL,
  p_type              VARCHAR(26)   NOT NULL,
  p_size              TINYINT(11)   NOT NULL,
  p_container         VARCHAR(11)   NOT NULL
) ENGINE=OLAP
DUPLICATE KEY(p_partkey)
DISTRIBUTED BY HASH(p_partkey) BUCKETS 12
PROPERTIES (
"unique_constraints" = "p_partkey"   -- Указать уникальные ограничения.
);

CREATE TABLE lineorder (
  lo_orderdate        DATE          NOT NULL, -- Указать как NOT NULL.
  lo_orderkey         INT(11)       NOT NULL,
  lo_linenumber       TINYINT       NOT NULL,
  lo_custkey          INT(11)       NOT NULL, -- Указать как NOT NULL.
  lo_partkey          INT(11)       NOT NULL, -- Указать как NOT NULL.
  lo_suppkey          INT(11)       NOT NULL, -- Указать как NOT NULL.
  lo_orderpriority    VARCHAR(100)  NOT NULL,
  lo_shippriority     TINYINT       NOT NULL,
  lo_quantity         TINYINT       NOT NULL,
  lo_extendedprice    INT(11)       NOT NULL,
  lo_ordtotalprice    INT(11)       NOT NULL,
  lo_discount         TINYINT       NOT NULL,
  lo_revenue          INT(11)       NOT NULL,
  lo_supplycost       INT(11)       NOT NULL,
  lo_tax              TINYINT       NOT NULL,
  lo_commitdate       DATE          NOT NULL,
  lo_shipmode         VARCHAR(100)  NOT NULL
```

```
) ENGINE=OLAP
DUPLICATE KEY(lo_orderdate,lo_orderkey)
PARTITION BY RANGE(lo_orderdate)
(PARTITION p1 VALUES [("0000-01-01"), ("1993-01-01")),
PARTITION p2 VALUES [("1993-01-01"), ("1994-01-01")),
PARTITION p3 VALUES [("1994-01-01"), ("1995-01-01")),
PARTITION p4 VALUES [("1995-01-01"), ("1996-01-01")),
PARTITION p5 VALUES [("1996-01-01"), ("1997-01-01")),
PARTITION p6 VALUES [("1997-01-01"), ("1998-01-01")),
PARTITION p7 VALUES [("1998-01-01"), ("1999-01-01")))
DISTRIBUTED BY HASH(lo_orderkey) BUCKETS 48
PROPERTIES (
"foreign_key_constraints" = "
    (lo_custkey) REFERENCES customer(c_custkey);
    (lo_partkey) REFERENCES part(p_partkey);
    (lo_suppkey) REFERENCES supplier(s_suppkey)" -- Указать Foreign Keys.
);
```

Создайте материализованное представление `lineorder_flat_mv`, которое объединяет `lineorder`, `customer`, `supplier`, `part` и `dates`:

```
CREATE MATERIALIZED VIEW lineorder_flat_mv
DISTRIBUTED BY HASH(LO_ORDERDATE, LO_ORDERKEY) BUCKETS 48
PARTITION BY LO_ORDERDATE
REFRESH MANUAL
PROPERTIES (
    "partition_refresh_number"="1"
)
AS SELECT /*+ SET_VAR(query_timeout = 7200) */     -- Установить timeout
для операции обновления.
       l.LO_ORDERDATE        AS LO_ORDERDATE,
       l.LO_ORDERKEY         AS LO_ORDERKEY,
       l.LO_LINENUMBER       AS LO_LINENUMBER,
       l.LO_CUSTKEY          AS LO_CUSTKEY,
       l.LO_PARTKEY          AS LO_PARTKEY,
       l.LO_SUPPKEY          AS LO_SUPPKEY,
       l.LO_ORDERPRIORITY    AS LO_ORDERPRIORITY,
       l.LO_SHIPPRIORITY     AS LO_SHIPPRIORITY,
       l.LO_QUANTITY         AS LO_QUANTITY,
       l.LO_EXTENDEDPRICE    AS LO_EXTENDEDPRICE,
       l.LO_ORDTOTALPRICE    AS LO_ORDTOTALPRICE,
       l.LO_DISCOUNT         AS LO_DISCOUNT,
       l.LO_REVENUE          AS LO_REVENUE,
       l.LO_SUPPLYCOST       AS LO_SUPPLYCOST,
       l.LO_TAX              AS LO_TAX,
       l.LO_COMMITDATE       AS LO_COMMITDATE,
       l.LO_SHIPMODE         AS LO_SHIPMODE,
       c.C_NAME              AS C_NAME,
       c.C_ADDRESS           AS C_ADDRESS,
       c.C_CITY              AS C_CITY,
       c.C_NATION            AS C_NATION,
       c.C_REGION            AS C_REGION,
       c.C_PHONE             AS C_PHONE,
       c.C_MKTSEGMENT        AS C_MKTSEGMENT,
       s.S_NAME              AS S_NAME,
       s.S_ADDRESS           AS S_ADDRESS,
       s.S_CITY              AS S_CITY,
       s.S_NATION            AS S_NATION,
       s.S_REGION            AS S_REGION,
       s.S_PHONE             AS S_PHONE,
       p.P_NAME              AS P_NAME,
       p.P_MFGR              AS P_MFGR,
       p.P_CATEGORY          AS P_CATEGORY,
       p.P_BRAND             AS P_BRAND,
       p.P_COLOR             AS P_COLOR,
       p.P_TYPE              AS P_TYPE,
```

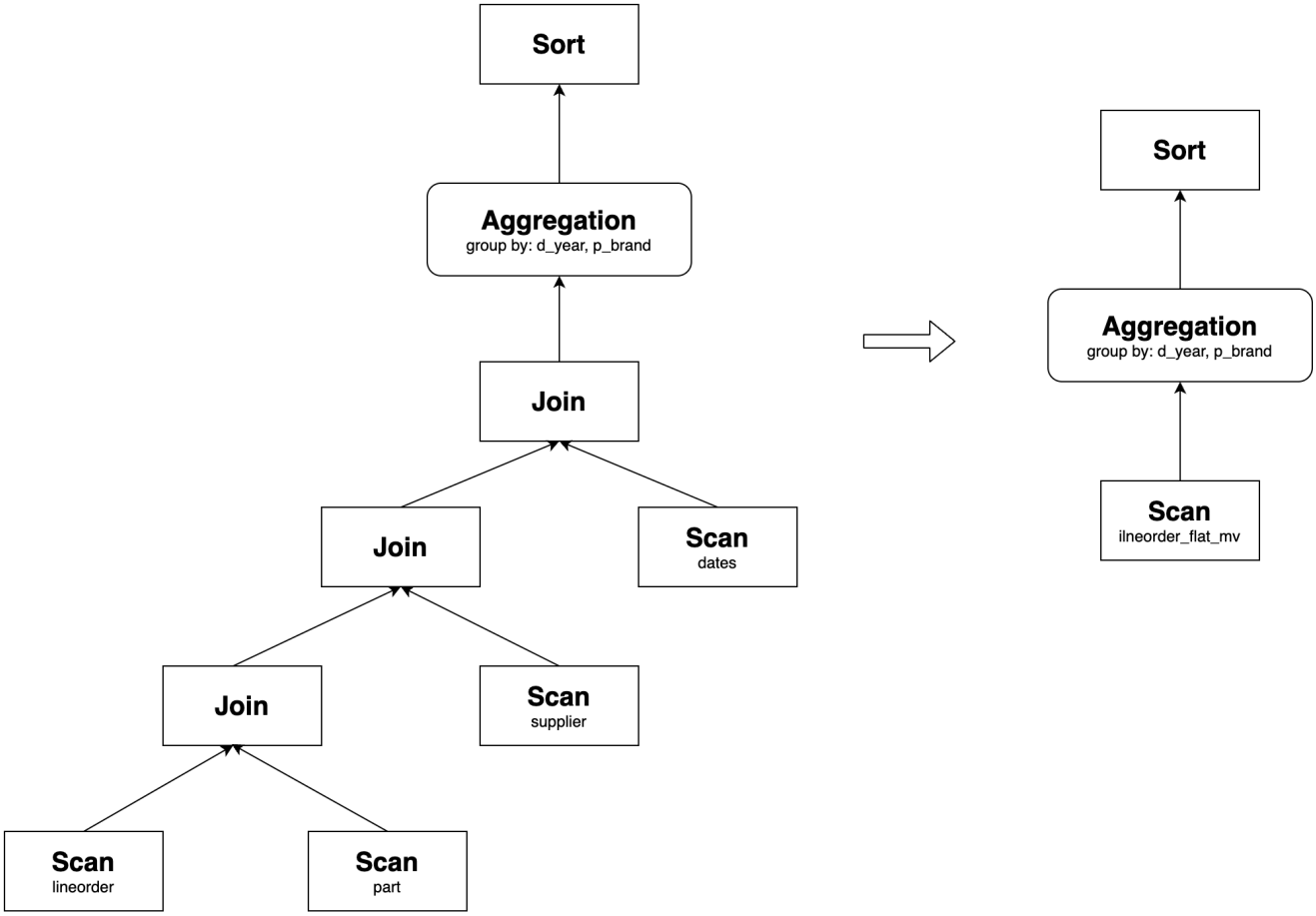
```
        p.P_SIZE                AS P_SIZE,
        p.P_CONTAINER           AS P_CONTAINER,
        d.D_DATE                AS D_DATE,
        d.D_DAYOFWEEK           AS D_DAYOFWEEK,
        d.D_MONTH               AS D_MONTH,
        d.D_YEAR                AS D_YEAR,
        d.D_YEARMONTHNUM        AS D_YEARMONTHNUM,
        d.D_YEARMONTH           AS D_YEARMONTH,
        d.D_DAYNUMINWEEK        AS D_DAYNUMINWEEK,
        d.D_DAYNUMINMONTH       AS D_DAYNUMINMONTH,
        d.D_DAYNUMINYEAR        AS D_DAYNUMINYEAR,
        d.D_MONTHNUMINYEAR      AS D_MONTHNUMINYEAR,
        d.D_WEEKNUMINYEAR       AS D_WEEKNUMINYEAR,
        d.D_SELLINGSEASON       AS D_SELLINGSEASON,
        d.D_LASTDAYINWEEKFL     AS D_LASTDAYINWEEKFL,
        d.D_LASTDAYINMONTHFL    AS D_LASTDAYINMONTHFL,
        d.D_HOLIDAYFL           AS D_HOLIDAYFL,
        d.D_WEEKDAYFL           AS D_WEEKDAYFL
    FROM lineorder              AS l
        INNER JOIN customer     AS c ON c.C_CUSTKEY = l.LO_CUSTKEY
        INNER JOIN supplier     AS s ON s.S_SUPPKEY = l.LO_SUPPKEY
        INNER JOIN part         AS p ON p.P_PARTKEY = l.LO_PARTKEY
        INNER JOIN dates        AS d ON l.LO_ORDERDATE = d.D_DATEKEY;
```

SSB Q2.1 включает join четырех таблиц, но в нем отсутствует таблица `customer` по сравнению с материализованным представлением `lineorder_flat_mv`. В `lineorder_flat_mv`, `lineorder INNER JOIN customer` по сути является join сохранения кардинальности. Следовательно, логически этот join может быть исключен без влияния на результаты запроса. В результате Q2.1 может быть переписан, используя `lineorder_flat_mv`.

SSB Q2.1:

```
SELECT sum(lo_revenue) AS lo_revenue, d_year, p_brand
FROM lineorder
JOIN dates ON lo_orderdate = d_datekey
JOIN part ON lo_partkey = p_partkey
JOIN supplier ON lo_suppkey = s_suppkey
WHERE p_category = 'MFGR#12' AND s_region = 'AMERICA'
GROUP BY d_year, p_brand
ORDER BY d_year, p_brand;
```

Его исходный план запроса и план после перезаписи следующие:



Аналогично, другие запросы в SSB также могут быть прозрачно переписаны, используя `lineorder_flat_mv`, таким образом оптимизируя производительность запросов.

Перезапись Join Derivability

Join Derivability относится к сценарию, в котором типы join в материализованном представлении и запросе не согласованы, но результаты join материализованного представления содержат результаты join запроса. В настоящее время поддерживаются два сценария — join трех или более таблиц и join двух таблиц.

- **Сценарий первый: Join трех или более таблиц**

  Предположим, что материализованное представление содержит Left Outer Join между таблицами `t1` и `t2` и Inner Join между таблицами `t2` и `t3`. В обоих joins условие join включает столбцы из `t2`.

Запрос, с другой стороны, содержит Inner Join между t1 и t2, и Inner Join между t2 и t3. В обоих joins условие join включает столбцы из t2.

В этом случае запрос может быть переписан, используя материализованное представление. Это происходит потому, что в материализованном представлении сначала выполняется Left Outer Join, а затем Inner Join. Правая таблица, сгенерированная Left Outer Join, не имеет результатов для совпадения (то есть столбцы в правой таблице являются NULL). Эти результаты впоследствии отфильтровываются во время Inner Join. Следовательно, логика материализованного представления и запроса эквивалентна, и запрос может быть переписан.

Пример:

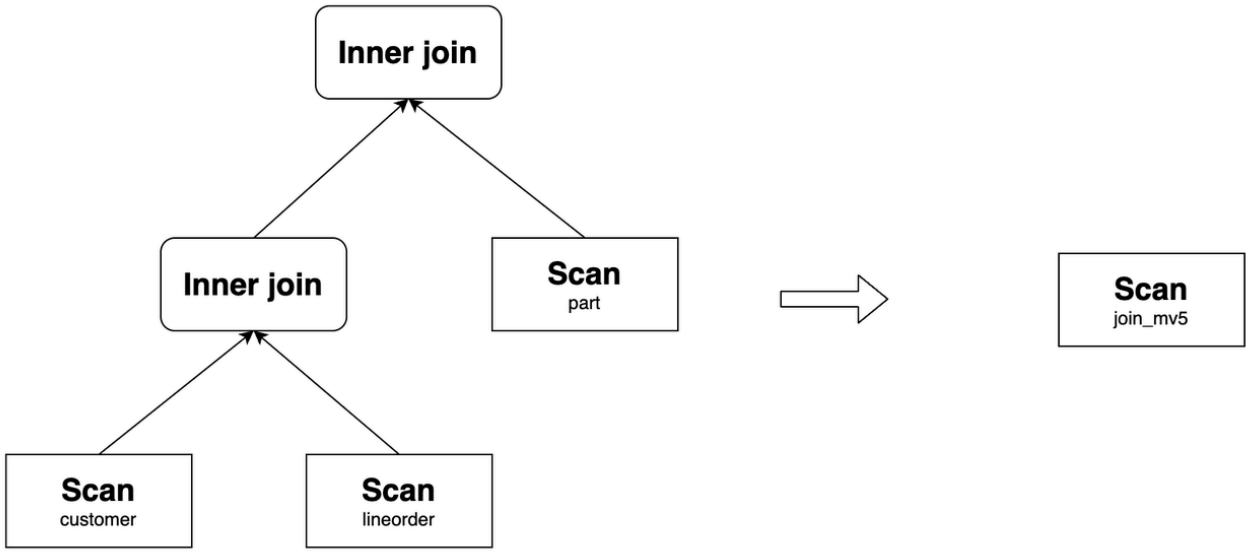
Создайте материализованное представление `join_mv5`:

```
CREATE MATERIALIZED VIEW join_mv5
PARTITION BY lo_orderdate
DISTRIBUTED BY hash(lo_orderkey)
PROPERTIES (
  "partition_refresh_number" = "1"
)
AS
SELECT lo_orderkey, lo_orderdate, lo_linenumber, lo_revenue, c_custkey,
c_address, p_name
FROM customer LEFT OUTER JOIN lineorder
ON c_custkey = lo_custkey
INNER JOIN part
ON p_partkey = lo_partkey;
```

`join_mv5` может переписать следующий запрос:

```
SELECT lo_orderkey, lo_orderdate, lo_linenumber, lo_revenue, c_custkey,
c_address, p_name
FROM customer INNER JOIN lineorder
ON c_custkey = lo_custkey
INNER JOIN part
ON p_partkey = lo_partkey;
```

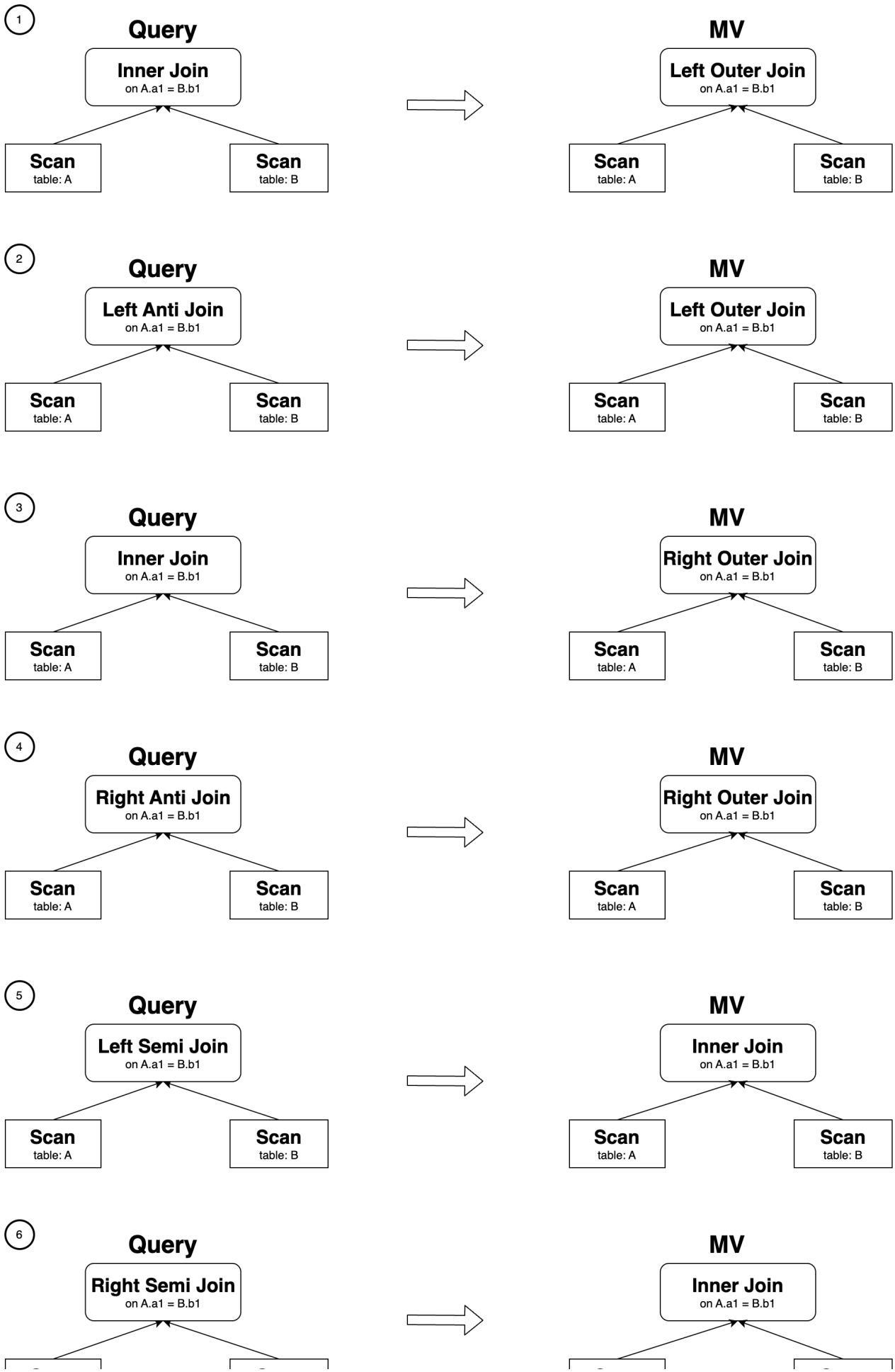
Его исходный план запроса и план после перезаписи следующие:



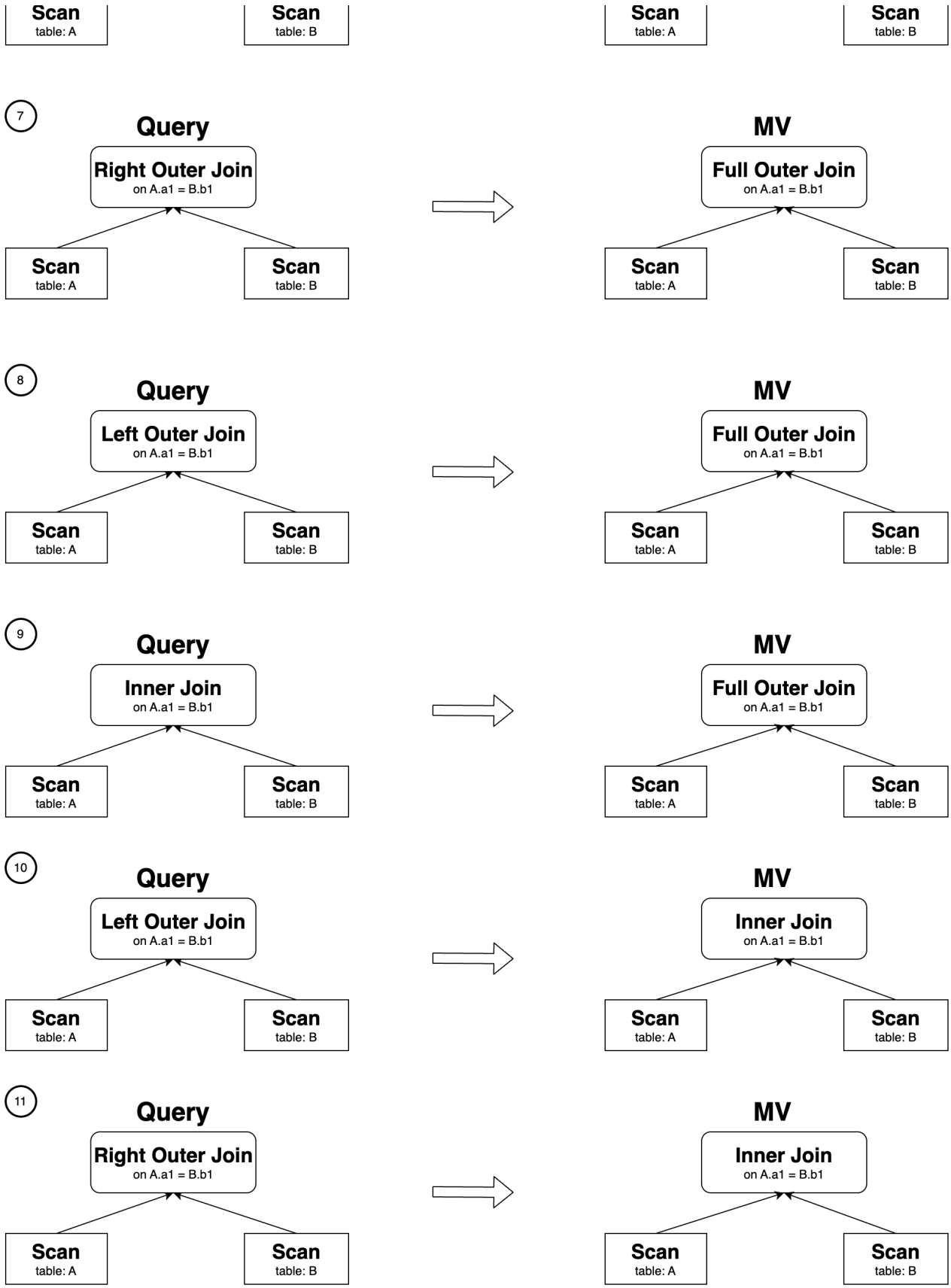
Аналогично, если материализованное представление определено как `t1 INNER JOIN t2 INNER JOIN t3`, а запрос — `LEFT OUTER JOIN t2 INNER JOIN t3`, запрос также может быть переписан. Кроме того, эта возможность перезаписи распространяется на сценарии, включающие более трех таблиц.

- **Сценарий второй: Join двух таблиц**

  Функция Join Derivability Rewrite, включающая две таблицы, поддерживает следующие конкретные случаи:



© ООО «Платформа Данных Селена», 2026



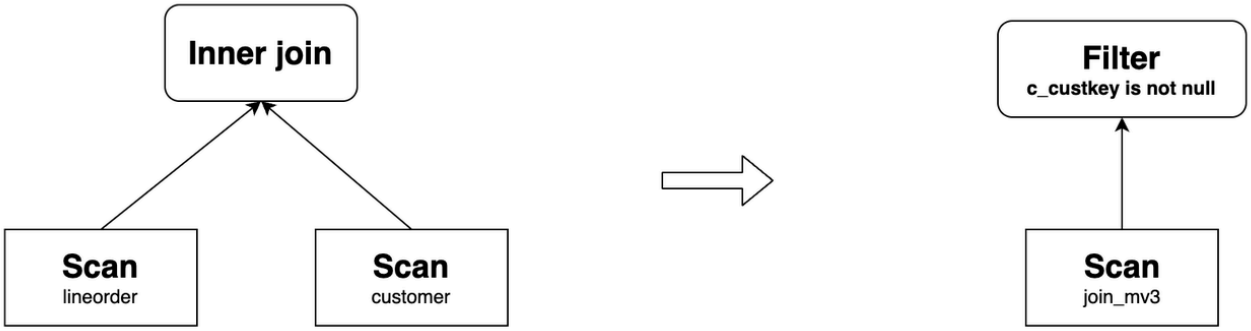
В случаях с 1 по 9 фильтрующие предикаты должны быть добавлены к переписанному результату для обеспечения семантической эквивалентности. Например, создайте материализованное представление следующим образом:

```
CREATE MATERIALIZED VIEW join_mv3
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT lo_orderkey, lo_linenumber, lo_revenue, c_custkey, c_address
FROM lineorder LEFT OUTER JOIN customer
ON lo_custkey = c_custkey;
```

Следующий запрос может быть переписан, используя `join_mv3`, и предикат `c_custkey IS NOT NULL` добавляется к переписанному результату:

```
SELECT lo_orderkey, lo_linenumber, lo_revenue, c_custkey, c_address
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey;
```

Его исходный план запроса и план после перезаписи следующие:



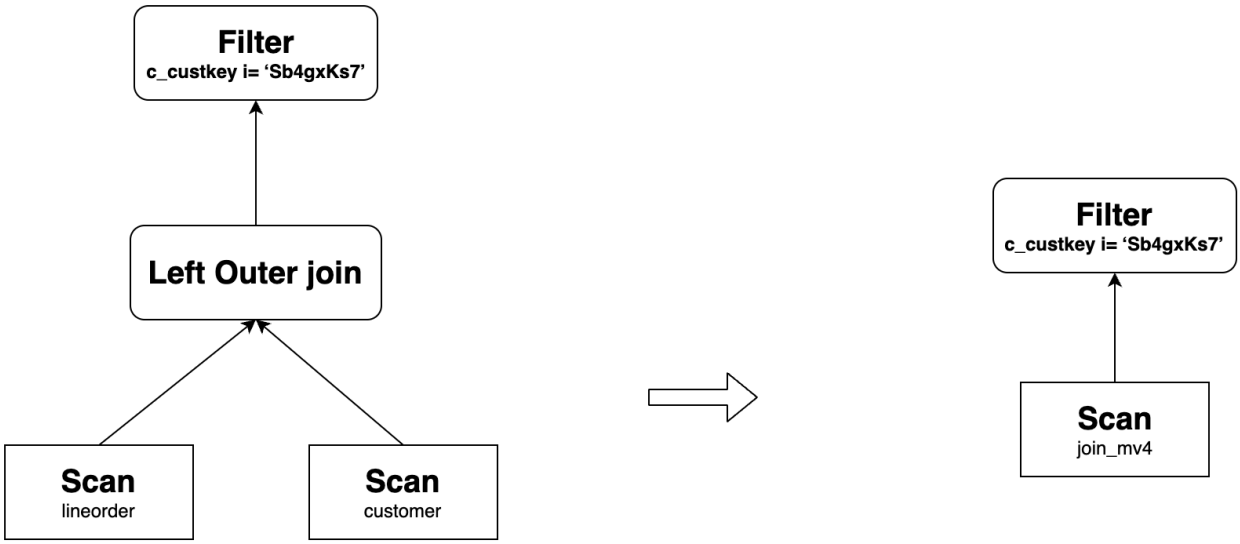
В случае 10, Left Outer Join запрос должен включать фильтрующий предикат `IS NOT NULL` в правой таблице, например, `=`, `<>`, `>`, `<`, `<=`, `>=`, `LIKE`, `IN`, `NOT LIKE` или `NOT IN`. Например, создайте материализованное представление следующим образом:

```
CREATE MATERIALIZED VIEW join_mv4
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT lo_orderkey, lo_linenumber, lo_revenue, c_custkey, c_address
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey;
```

`join_mv4` может переписать следующий запрос, где `customer.c_address = "Sb4gxKs7"` является фильтрующим предикатом `IS NOT NULL`:

```
SELECT lo_orderkey, lo_linenumber, lo_revenue, c_custkey, c_address
FROM lineorder LEFT OUTER JOIN customer
ON lo_custkey = c_custkey
WHERE customer.c_address = "Sb4gxKs7";
```

Его исходный план запроса и план после перезаписи следующие:



Перезапись агрегации

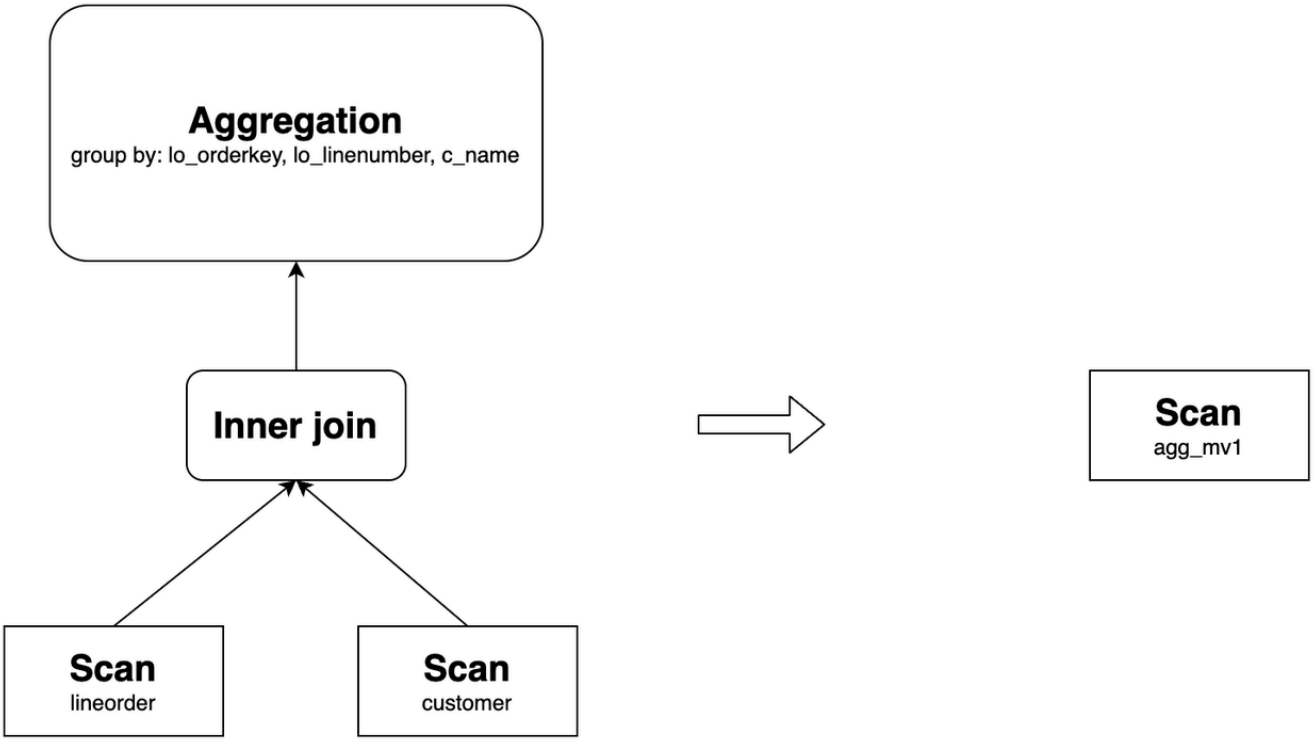
Асинхронное материализованное представление Selena поддерживает перезапись многотабличных агрегатных запросов со всеми доступными агрегатными функциями, включая bitmap_union, hll_union и percentile_union. Например, создайте материализованное представление следующим образом:

```
CREATE MATERIALIZED VIEW agg_mv1
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT
  lo_orderkey,
  lo_linenumber,
  c_name,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey
GROUP BY lo_orderkey, lo_linenumber, c_name;
```

Оно может переписать следующий запрос:

```
SELECT
  lo_orderkey,
  lo_linenumber,
  c_name,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey
GROUP BY lo_orderkey, lo_linenumber, c_name;
```

Его исходный план запроса и план после перезаписи следующие:



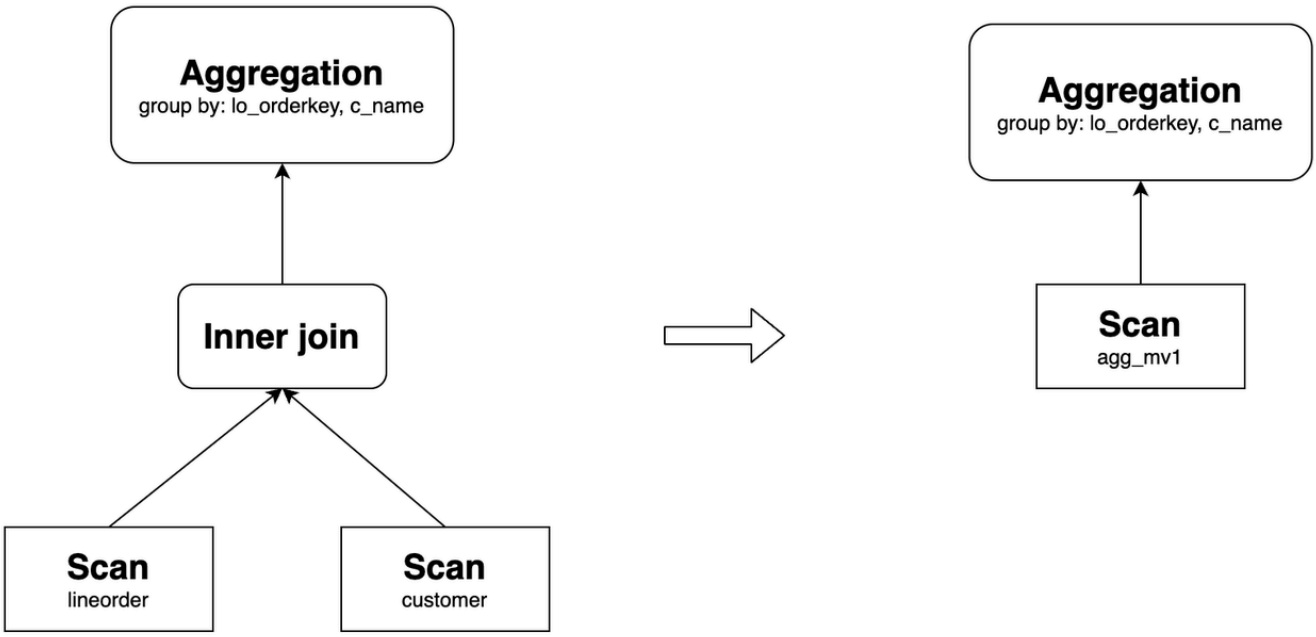
Следующие разделы излагают сценарии, где функция Aggregation Rewrite может быть полезной.

Перезапись Aggregation Rollup

Selena поддерживает перезапись запросов с Aggregation Rollup, то есть Selena может переписывать агрегатные запросы с оператором `GROUP BY a`, используя асинхронное материализованное представление, созданное с оператором `GROUP BY a,b`. Например, следующий запрос может быть переписан, используя `agg_mv1`:

```
SELECT
  lo_orderkey,
  c_name,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey
GROUP BY lo_orderkey, c_name;
```

Его исходный план запроса и план после перезаписи следующие:



> **NOTE**
>
> В настоящее время перезапись grouping set, grouping set with rollup или grouping set with cube не поддерживается.

Только определенные агрегатные функции поддерживают перезапись запросов с Aggregate Rollup. В предыдущем примере, если материализованное представление `order_agg_mv` использует `count(distinct client_id)` вместо `bitmap_union(to_bitmap(client_id))`, Selena не может переписать запросы с Aggregate Rollup.

Следующая таблица показывает соответствие между агрегатными функциями в исходном запросе и агрегатной функцией, используемой для построения материализованного

представления. Вы можете выбрать соответствующие агрегатные функции для построения материализованного представления в соответствии с вашим бизнес-сценарием.

Агрегатная функция, поддерживаемая в исходных запросах	Функция, поддерживающая Aggregate Rollup в материализованном представлении
sum	sum
count	count
min	min
max	max
avg	sum / count
bitmap_union, bitmap_union_count, count(distinct)	bitmap_union
hll_raw_agg, hll_union_agg, ndv, approx_count_distinct	hll_union
percentile_approx, percentile_union	percentile_union

DISTINCT агрегаты без соответствующего столбца GROUP BY не могут быть переписаны с Aggregate Rollup. Однако, начиная с Selena v1.5.2, если запрос с Aggregate Rollup DISTINCT агрегатной функцией не имеет столбца GROUP BY, но имеет равенственный предикат, он также может быть переписан соответствующим материализованным представлением, потому что Selena может преобразовать равенственные предикаты в выражение константы GROUP BY.

В следующем примере Selena может переписать запрос с помощью материализованного представления `order_agg_mv1`.

```
CREATE MATERIALIZED VIEW order_agg_mv1
DISTRIBUTED BY HASH(`order_id`) BUCKETS 12
REFRESH ASYNC START('2022-09-01 10:00:00') EVERY (interval 1 day)
AS
SELECT
    order_date,
    count(distinct client_id)
FROM order_list
GROUP BY order_date;


-- Запрос
SELECT
    order_date,
    count(distinct client_id)
FROM order_list WHERE order_date='2023-07-03';
```

В дополнение к указанным выше функциям, начиная с Selena v1.5.2, асинхронные материализованные представления также поддерживают универсальные агрегатные функции, которые также могут использоваться для перезаписи запросов. Для получения дополнительной информации об универсальных агрегатных функциях см. Generic aggregate function states.

```
-- Создать асинхронное материализованное представление test_mv2 для
хранения aggregate states.
CREATE MATERIALIZED VIEW test_mv2
PARTITION BY (dt)
DISTRIBUTED BY RANDOM
AS
SELECT
    dt,
    -- Исходные агрегатные функции.
    min(id) AS min_id,
    max(id) AS max_id,
    sum(id) AS sum_id,
    bitmap_union(to_bitmap(id)) AS bitmap_union_id,
    hll_union(hll_hash(id)) AS hll_union_id,
    percentile_union(percentile_hash(id)) AS percentile_union_id,
    -- Generic aggregate state функции.
    ds_hll_count_distinct_union(ds_hll_count_distinct_state(id)) AS hll_id,
    avg_union(avg_state(id)) AS avg_id,
    array_agg_union(array_agg_state(id)) AS array_agg_id,
    min_by_union(min_by_state(province, id)) AS min_by_province_id
FROM t1
GROUP BY dt;

-- Обновить материализованное представление.
REFRESH MATERIALIZED VIEW test_mv2 WITH SYNC MODE;

-- Прямые запросы к агрегатной функции будут прозрачно ускорены с помощью
test_mv2.
SELECT
    dt,
    min(id),
    max(id),
    sum(id),
    bitmap_union_count(to_bitmap(id)), -- count(distinct id)
    hll_union_agg(hll_hash(id)), -- approx_count_distinct(id)
    percentile_approx(id, 0.5),
    ds_hll_count_distinct(id),
    avg(id),
    array_agg(id),
    min_by(province, id)
FROM t1
WHERE dt >= '2024-01-01'
GROUP BY dt;

SELECT
```

```
    min(id),
    max(id),
    sum(id),
    bitmap_union_count(to_bitmap(id)), -- count(distinct id)
    hll_union_agg(hll_hash(id)), -- approx_count_distinct(id)
    percentile_approx(id, 0.5),
    ds_hll_count_distinct(id),
    avg(id),
    array_agg(id),
    min_by(province, id)
FROM t1
WHERE dt >= '2024-01-01';
```

Pushdown агрегации

Начиная с v1.5.2, Selena поддерживает pushdown агрегации для перезаписи запросов материализованного представления. Когда эта функция включена, агрегатные функции будут проталкиваться вниз к Scan Operator во время выполнения запроса и переписываться материализованным представлением до выполнения Join Operator. Это снизит расширение данных, вызванное Join, и тем самым улучшит производительность запросов.

Эта функция отключена по умолчанию. Чтобы включить эту функцию, вы должны установить системную переменную `enable_materialized_view_agg_pushdown_rewrite` в `true`.

Предположим, вы хотите ускорить следующий запрос на основе SSB `SQL1`:

```
-- SQL1
SELECT
    LO_ORDERDATE, sum(LO_REVENUE), max(LO_REVENUE), count(distinct
LO_REVENUE)
FROM lineorder l JOIN dates d
ON l.LO_ORDERDATE = d.d_date
GROUP BY LO_ORDERDATE
ORDER BY LO_ORDERDATE;
```

`SQL1` состоит из агрегаций для таблицы `lineorder` и Join `lineorder` и `dates`. Поскольку агрегации происходят внутри `lineorder`, а Join с `dates` используется только для фильтрации данных, `SQL1` логически эквивалентен следующему `SQL2`:

```
-- SQL2
SELECT
    LO_ORDERDATE, sum(sum1), max(max1), bitmap_union_count(bitmap1)
FROM
 (SELECT
  LO_ORDERDATE,  sum(LO_REVENUE) AS sum1, max(LO_REVENUE) AS max1,
bitmap_union(to_bitmap(LO_REVENUE)) AS bitmap1
  FROM lineorder
  GROUP BY LO_ORDERDATE) l JOIN dates d
ON l.LO_ORDERDATE = d.d_date
GROUP BY LO_ORDERDATE
ORDER BY LO_ORDERDATE;
```

`SQL2` выносит агрегации вперед, тем самым уменьшая размер данных Join. Вы можете создать материализованное представление на основе подзапроса `SQL2` и включить pushdown агрегации для перезаписи и ускорения агрегаций:

```
-- Создать материализованное представление mv0
CREATE MATERIALIZED VIEW mv0 REFRESH MANUAL AS
SELECT
  LO_ORDERDATE,
  sum(LO_REVENUE) AS sum1,
  max(LO_REVENUE) AS max1,
  bitmap_union(to_bitmap(LO_REVENUE)) AS bitmap1
FROM lineorder
GROUP BY LO_ORDERDATE;

-- Включить pushdown агрегации для перезаписи запросов материализованного
представления
SET enable_materialized_view_agg_pushdown_rewrite=true;
```

Затем `SQL1` будет переписан и ускорен материализованным представлением. Он переписывается в следующий запрос:

```
SELECT
    LO_ORDERDATE, sum(sum1), max(max1), bitmap_union_count(bitmap1)
FROM
 (SELECT LO_ORDERDATE, sum1, max1, bitmap1 FROM mv0) l JOIN dates d
ON l.LO_ORDERDATE = d.d_date
GROUP BY LO_ORDERDATE
ORDER BY LO_ORDERDATE;
```

Обратите внимание, что только определенные агрегатные функции, которые поддерживают перезапись Aggregate Rollup, подходят для pushdown. Это:

- MIN
- MAX
- COUNT
- COUNT DISTINCT
- SUM
- BITMAP_UNION
- HLL_UNION
- PERCENTILE_UNION
- BITMAP_AGG
- ARRAY_AGG_DISTINCT

> **ПРИМЕЧАНИЕ**
>
> - После pushdown агрегатные функции должны быть свернуты для соответствия исходной семантике. Для получения дополнительных инструкций по Aggregation Rollup, пожалуйста, обратитесь к Aggregation Rollup Rewrite.
> - Pushdown агрегации поддерживает перезапись Rollup Count Distinct на основе функций Bitmap или HLL.
> - Pushdown агрегации поддерживает только проталкивание агрегатных функций вниз к Scan Operator перед операторами Join, Filter или Where.
> - Pushdown агрегации поддерживает только перезапись запросов и ускорение на основе материализованного представления, построенного на одной таблице.

Перезапись COUNT DISTINCT

Selena поддерживает перезапись вычислений COUNT DISTINCT в вычисления на основе bitmap, обеспечивая высокопроизводительную точную дедупликацию с использованием

материализованных представлений. Например, создайте материализованное представление следующим образом:

```
CREATE MATERIALIZED VIEW distinct_mv
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT lo_orderkey, bitmap_union(to_bitmap(lo_custkey)) AS
distinct_customer
FROM lineorder
GROUP BY lo_orderkey;
```

Оно может переписать следующий запрос:

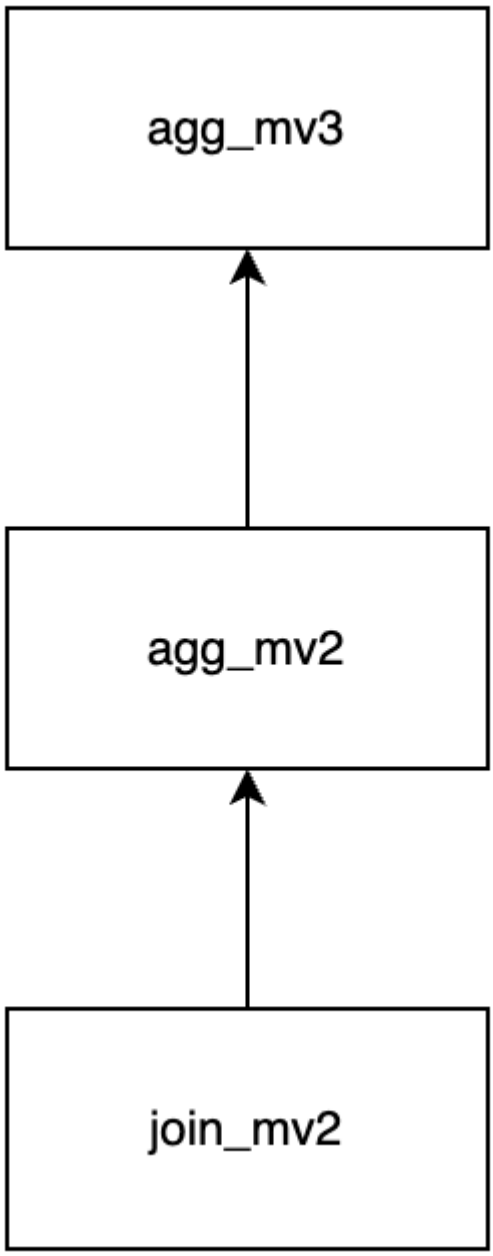
```
SELECT lo_orderkey, count(distinct lo_custkey)
FROM lineorder
GROUP BY lo_orderkey;
```

Перезапись вложенного материализованного представления

Selena поддерживает перезапись запросов с использованием вложенного материализованного представления. Например, создайте материализованные представления `join_mv2`, `agg_mv2` и `agg_mv3` следующим образом:

```
CREATE MATERIALIZED VIEW join_mv2
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT lo_orderkey, lo_linenumber, lo_revenue, c_name, c_address
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey;


CREATE MATERIALIZED VIEW agg_mv2
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT
  lo_orderkey,
  lo_linenumber,
  c_name,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM join_mv2
GROUP BY lo_orderkey, lo_linenumber, c_name;

CREATE MATERIALIZED VIEW agg_mv3
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT
  lo_orderkey,
  sum(total_revenue) AS total_revenue,
  max(max_discount) AS max_discount
FROM agg_mv2
GROUP BY lo_orderkey;
```

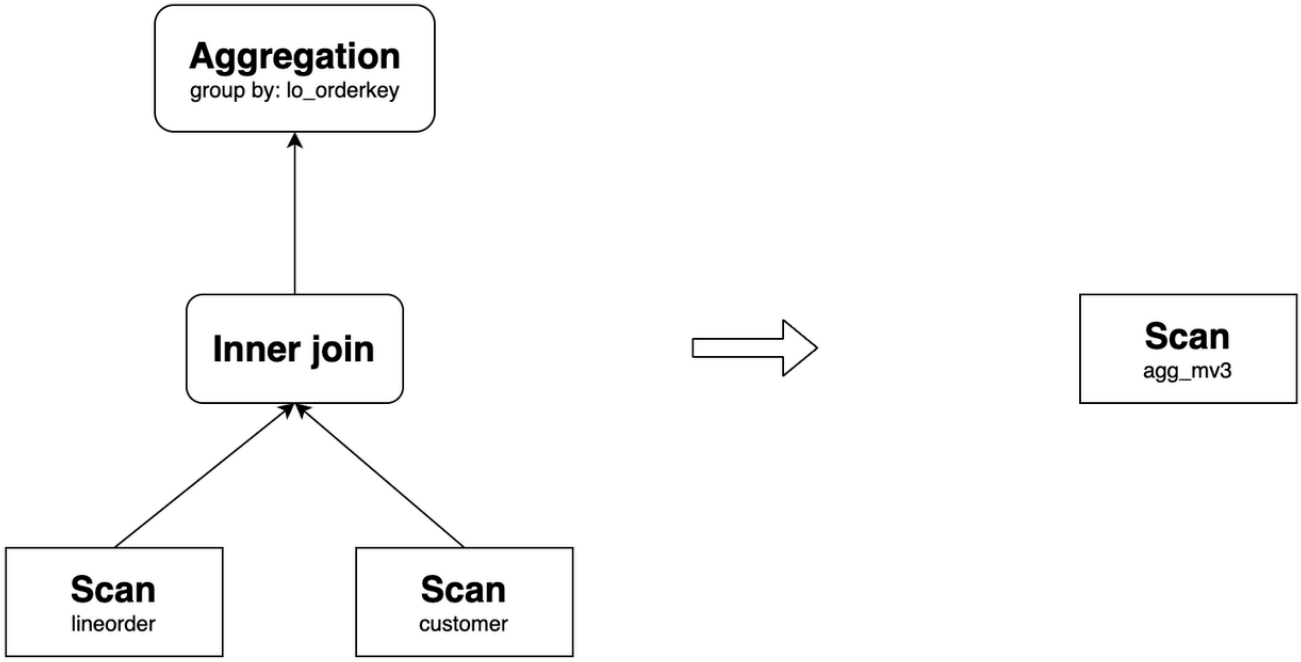
Их взаимоотношения следующие:



`agg_mv3` может переписать следующий запрос:

```
SELECT
  lo_orderkey,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM lineorder INNER JOIN customer
ON lo_custkey = c_custkey
GROUP BY lo_orderkey;
```

Его исходный план запроса и план после перезаписи следующие:



Перезапись Union

Перезапись Predicate Union

Когда область предиката материализованного представления является подмножеством области предиката запроса, запрос может быть переписан с использованием операции UNION.

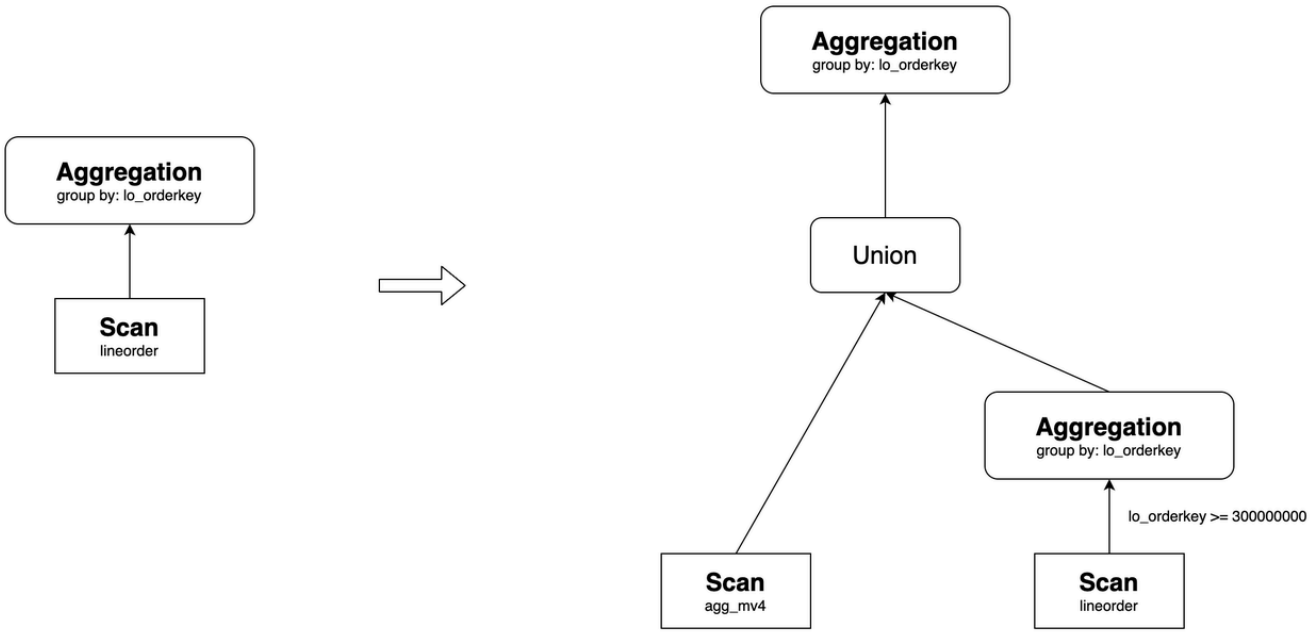
Например, создайте материализованное представление следующим образом:

```
CREATE MATERIALIZED VIEW agg_mv4
DISTRIBUTED BY hash(lo_orderkey)
AS
SELECT
  lo_orderkey,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM lineorder
WHERE lo_orderkey < 300000000
GROUP BY lo_orderkey;
```

Оно может переписать следующий запрос:

```
select
  lo_orderkey,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM lineorder
GROUP BY lo_orderkey;
```

Его исходный план запроса и план после перезаписи следующие:



В этом контексте `agg_mv5` содержит данные, где `lo_orderkey < 300000000`. Данные, где `lo_orderkey >= 300000000`, напрямую получаются из базовой таблицы `lineorder`. Наконец, эти два набора данных объединяются с использованием операции UNION, а затем агрегируются для получения окончательного результата.

Перезапись Partition Union

Предположим, вы создали partitioned материализованное представление на основе partitioned таблицы. Когда диапазон partition, который сканируется переписываемым запросом, является надмножеством самого последнего диапазона partition материализованного представления, запрос будет переписан с использованием операции UNION.

Например, рассмотрим следующее материализованное представление `agg_mv4`. Его базовая таблица `lineorder` в настоящее время содержит partitions от `p1` до `p7`, и

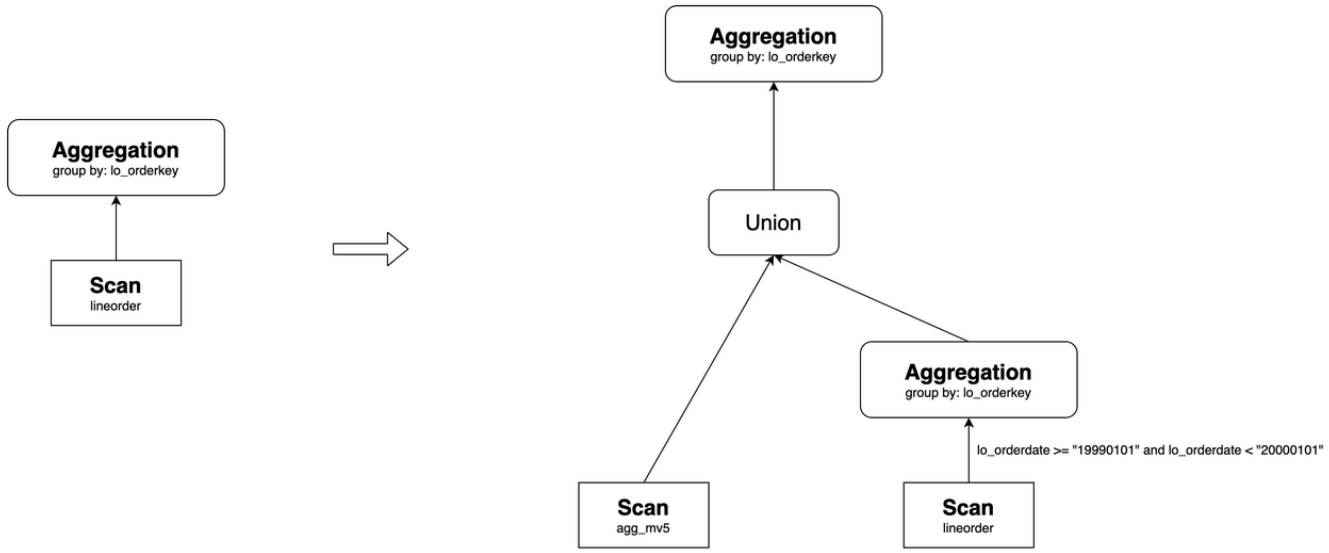
материализованное представление также содержит partitions от `p1` до `p7`.

```
CREATE MATERIALIZED VIEW agg_mv5
DISTRIBUTED BY hash(lo_orderkey)
PARTITION BY RANGE(lo_orderdate)
REFRESH MANUAL
AS
SELECT
  lo_orderdate,
  lo_orderkey,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM lineorder
GROUP BY lo_orderkey;
```

Если новый partition `p8` с диапазоном partition `[("19990101"), ("20000101"))` добавляется к `lineorder`, следующий запрос может быть переписан с использованием операции UNION:

```
SELECT
  lo_orderdate,
  lo_orderkey,
  sum(lo_revenue) AS total_revenue,
  max(lo_discount) AS max_discount
FROM lineorder
GROUP BY lo_orderkey;
```

Его исходный план запроса и план после перезаписи следующие:



Как показано выше, `agg_mv5` содержит данные из partitions от `p1` до `p7`, а данные из partition `p8` напрямую запрашиваются из `lineorder`. Наконец, эти два набора данных объединяются с использованием операции UNION.

Перезапись материализованного представления на основе представления

Начиная с v1.5.2, Selena поддерживает создание материализованных представлений на основе представлений. Последующие запросы к представлениям могут быть переписаны, если они имеют паттерн SPJG. По умолчанию запросы к представлениям автоматически транскрибируются в запросы к базовым таблицам представлений, а затем прозрачно сопоставляются и переписываются.

Однако в реальных сценариях аналитики данных могут выполнять моделирование данных на основе сложных вложенных представлений, которые не могут быть напрямую транскрибированы. В результате материализованные представления, созданные на основе таких представлений, не могут переписывать запросы. Чтобы улучшить свои возможности в предыдущем сценарии, Selena оптимизирует логику перезаписи запросов материализованного представления на основе представления начиная с v1.5.2.

Основы

В предыдущей логике перезаписи запросов Selena транскрибирует запросы к представлению в запросы к базовым таблицам представления. Перезапись запросов встретит сбои, если план выполнения транскрибированного запроса не соответствует паттерну SPJG.

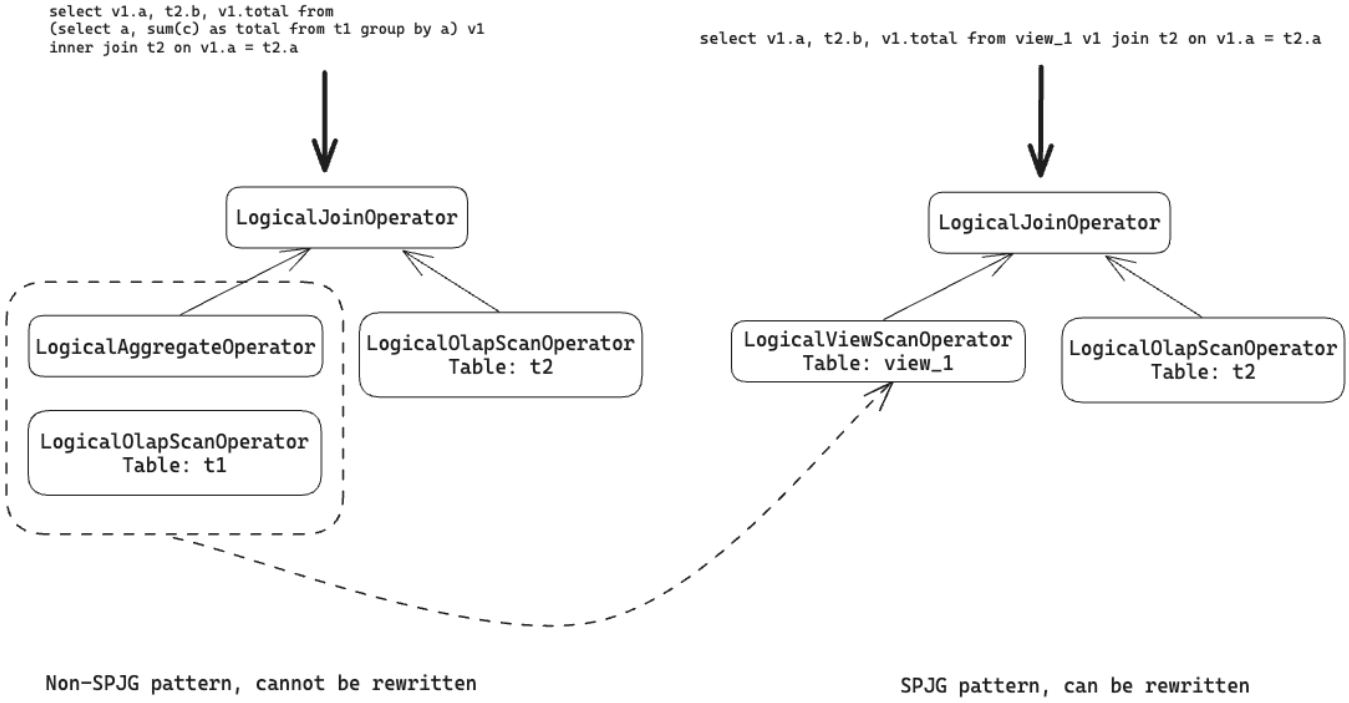
Чтобы решить эту проблему, Selena вводит новый оператор - LogicalViewScanOperator, чтобы упростить структуру дерева плана выполнения без транскрибирования запроса. Этот оператор стремится сопоставить дерево плана выполнения с паттерном SPJG, тем самым облегчая перезапись запросов.

Следующий пример перечисляет запрос с подзапросом AGGREGATE, представление, построенное на подзапросе, транскрибированный запрос на основе представления и материализованное представление, построенное на представлении:

```sql
-- Исходный запрос:
SELECT
  v1.a,
  t2.b,
  v1.total
FROM(
  SELECT
    a,
    sum(c) AS total
  FROM t1
  GROUP BY a
) v1
INNER JOIN t2 ON v1.a = t2.a;

-- Представление:
CREATE VIEW view_1 AS
SELECT
  t1.a,
  sum(t1.c) AS total
FROM t1
GROUP BY t1.a;

-- Транскрибированный запрос:
SELECT
  v1.a,
  t2.b,
  v1.total
FROM view_1 v1
JOIN t2 ON v1.a = t2.a;

-- Материализованное представление:
CREATE MATERIALIZED VIEW mv1
DISTRIBUTED BY hash(a)
REFRESH MANUAL
AS
SELECT
  v1.a,
  t2.b,
  v1.total
FROM view_1 v1
JOIN t2 ON v1.a = t2.a;
```

План выполнения исходного запроса, как показано слева на следующей диаграмме, не соответствует паттерну SPJG из-за LogicalAggregateOperator внутри JOIN. Selena не

поддерживает перезапись запросов для таких случаев. Однако, определив представление на основе подзапроса, исходный запрос может быть транскрибирован в запрос к представлению. С помощью LogicalViewScanOperator Selena может перенести несоответствующую часть в паттерн SPJG, тем самым позволяя перезаписать запрос в этих обстоятельствах.



Использование

Перезапись запросов материализованного представления на основе представления отключена по умолчанию.

Чтобы включить эту функцию, вы должны установить следующую переменную:

```
SET enable_view_based_mv_rewrite = true;
```

Случаи использования

Перезапись запросов с использованием материализованных представлений на основе одного представления

Selena поддерживает перезапись запросов с материализованным представлением, построенным на одном представлении, включая запросы с агрегациями.

Например, вы можете построить следующее представление и материализованное представление для TPC-H Query 18:

```
CREATE VIEW q18_view
AS
SELECT
  c_name,
  c_custkey,
  o_orderkey,
  o_orderdate,
  o_totalprice,
  sum(l_quantity)
FROM
  customer,
  orders,
  lineitem
WHERE
  o_orderkey IN (
    SELECT
      l_orderkey
    FROM
      lineitem
    GROUP BY
      l_orderkey having
        sum(l_quantity) > 315
  )
  AND c_custkey = o_custkey
  AND o_orderkey = l_orderkey
GROUP BY
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice;

CREATE MATERIALIZED VIEW q18_mv
DISTRIBUTED BY hash(c_custkey, o_orderkey)
REFRESH MANUAL
AS
SELECT * FROM q18_view;
```

Материализованное представление может переписать оба следующих запроса:

```
mysql> EXPLAIN LOGICAL SELECT * FROM q18_view;
+-----------------------------------------------------------------------------
-----------------------------+
| Explain String
|
+-----------------------------------------------------------------------------
-----------------------------+
| - Output => [2:c_name, 1:c_custkey, 9:o_orderkey, 10:o_orderdate,
13:o_totalprice, 52:sum]            |
|     - SCAN [q18_mv] => [1:c_custkey, 2:c_name, 52:sum, 9:o_orderkey,
10:o_orderdate, 13:o_totalprice] |
|             MaterializedView: true
|
|             Estimates: {row: 9, cpu: 486.00, memory: 0.00, network: 0.00,
cost: 243.00}               |
|             partitionRatio: 1/1, tabletRatio: 96/96
|
|             1:c_custkey := 60:c_custkey
|
|             2:c_name := 59:c_name
|
|             52:sum := 64:sum(l_quantity)
|
|             9:o_orderkey := 61:o_orderkey
|
|             10:o_orderdate := 62:o_orderdate
|
|             13:o_totalprice := 63:o_totalprice
|
+-----------------------------------------------------------------------------
-----------------------------+
```

```
mysql> EXPLAIN LOGICAL SELECT c_name, sum(`sum(l_quantity)`) FROM q18_view
GROUP BY c_name;
+-----------------------------------------------------------------------------
---------------------------+
| Explain String
|
+-----------------------------------------------------------------------------
---------------------------+
| - Output => [2:c_name, 59:sum]
|
|     - AGGREGATE(GLOBAL) [2:c_name]
|
|             Estimates: {row: 9, cpu: 306.00, memory: 306.00, network:
0.00, cost: 1071.00}          |
|             59:sum := sum(59:sum)
|
|         - EXCHANGE(SHUFFLE) [2]
|
|                 Estimates: {row: 9, cpu: 30.60, memory: 0.00, network:
30.60, cost: 306.00}         |
|             - AGGREGATE(LOCAL) [2:c_name]
|
|                     Estimates: {row: 9, cpu: 61.20, memory: 30.60,
network: 0.00, cost: 244.80}     |
|                     59:sum := sum(52:sum)
|
|                 - SCAN [q18_mv] => [2:c_name, 52:sum]
|
|                         MaterializedView: true
|
|                         Estimates: {row: 9, cpu: 306.00, memory: 0.00,
network: 0.00, cost: 153.00} |
|                         partitionRatio: 1/1, tabletRatio: 96/96
|
|                         2:c_name := 60:c_name
|
|                         52:sum := 65:sum(l_quantity)
|
+-----------------------------------------------------------------------------
---------------------------+
```

Перезапись запросов с JOIN с использованием материализованных представлений на основе представления

Selena поддерживает перезапись запросов с JOINs между представлениями или между представлениями и таблицами, включая агрегации на JOINs.

Например, вы можете создать следующие представления и материализованное представление:

```
CREATE VIEW view_1 AS
SELECT
  l_partkey,
  l_suppkey,
  sum(l_quantity) AS total_quantity
FROM lineitem
GROUP BY
  l_partkey,
  l_suppkey;


CREATE VIEW view_2 AS
SELECT
  l_partkey,
  l_suppkey,
   sum(l_tax) AS total_tax
FROM lineitem
GROUP BY
  l_partkey,
  l_suppkey;


CREATE MATERIALIZED VIEW mv_1
DISTRIBUTED BY hash(l_partkey, l_suppkey)
REFRESH MANUAL AS
SELECT
  v1.l_partkey,
  v2.l_suppkey,
  total_quantity,
  total_tax
FROM view_1 v1
JOIN view_2 v2 ON v1.l_partkey = v2.l_partkey
AND v1.l_suppkey = v2.l_suppkey;
```

Материализованное представление может переписать оба следующих запроса:

```
mysql>  EXPLAIN LOGICAL
    -> SELECT v1.l_partkey,
    ->        v2.l_suppkey,
    ->        total_quantity,
    ->        total_tax
    -> FROM view_1 v1
    -> JOIN view_2 v2 ON v1.l_partkey = v2.l_partkey
    -> AND v1.l_suppkey = v2.l_suppkey;
+-------------------------------------------------------------------------
------------------------------+
| Explain String
|
+-------------------------------------------------------------------------
------------------------------+
| - Output => [4:l_partkey, 25:l_suppkey, 17:sum, 37:sum]
|
|     - SCAN [mv_1] => [17:sum, 4:l_partkey, 37:sum, 25:l_suppkey]
|
|             MaterializedView: true
|
|             Estimates: {row: 799541, cpu: 31981640.00, memory: 0.00,
network: 0.00, cost: 15990820.00} |
|             partitionRatio: 1/1, tabletRatio: 96/96
|
|             17:sum := 43:total_quantity
|
|             4:l_partkey := 41:l_partkey
|
|             37:sum := 44:total_tax
|
|             25:l_suppkey := 42:l_suppkey
|
+-------------------------------------------------------------------------
------------------------------+
```

```
mysql> EXPLAIN LOGICAL
    -> SELECT v1.l_partkey,
    ->        sum(total_quantity),
    ->        sum(total_tax)
    -> FROM view_1 v1
    -> JOIN view_2 v2 ON v1.l_partkey = v2.l_partkey
    -> AND v1.l_suppkey = v2.l_suppkey
    -> group by v1.l_partkey;
+-----------------------------------------------------------------------------
-----------------------------------------+
| Explain String
|
+-----------------------------------------------------------------------------
-----------------------------------------+
| - Output => [4:l_partkey, 41:sum, 42:sum]
|
|     - AGGREGATE(GLOBAL) [4:l_partkey]
|
|             Estimates: {row: 196099, cpu: 4896864.00, memory: 3921980.00,
network: 0.00, cost: 29521223.20}        |
|             41:sum := sum(41:sum)
|
|             42:sum := sum(42:sum)
|
|         - EXCHANGE(SHUFFLE) [4]
|
|                 Estimates: {row: 136024, cpu: 489686.40, memory: 0.00,
network: 489686.40, cost: 19228831.20}      |
|             - AGGREGATE(LOCAL) [4:l_partkey]
|
|                     Estimates: {row: 136024, cpu: 5756695.20, memory:
489686.40, network: 0.00, cost: 18249458.40} |
|                     41:sum := sum(17:sum)
|
|                     42:sum := sum(37:sum)
|
|                 - SCAN [mv_1] => [17:sum, 4:l_partkey, 37:sum]
|
|                         MaterializedView: true
|
|                         Estimates: {row: 799541, cpu: 28783476.00,
memory: 0.00, network: 0.00, cost: 14391738.00} |
|                         partitionRatio: 1/1, tabletRatio: 96/96
|
|                         17:sum := 45:total_quantity
```

```
|
|                          4:l_partkey := 43:l_partkey
|
|                          37:sum := 46:total_tax
|
+-------------------------------------------------------------------------
-----------------------------------------+
```

Перезапись запросов с использованием материализованных представлений, построенных на представлениях на основе внешних таблиц

Вы можете построить представления на таблицах во внешних каталогах, а затем материализованные представления на представлениях для перезаписи запросов. Использование аналогично таковому для внутренних таблиц.

Перезапись материализованного представления на основе внешнего каталога

Selena поддерживает построение асинхронных материализованных представлений на Hive catalogs, Hudi catalogs, Iceberg catalogs и Paimon catalogs и прозрачную перезапись запросов с их помощью.

Текстовая перезапись материализованного представления

Начиная с v1.5.2, Selena поддерживает текстовую перезапись материализованного представления, которая значительно расширяет возможности перезаписи запросов.

Основы

Для достижения текстовой перезаписи материализованного представления Selena сравнивает абстрактное синтаксическое дерево запроса (или его подзапросов) с таковым определения материализованного представления. Когда они соответствуют друг другу, Selena переписывает запрос на основе материализованного представления. Текстовая перезапись материализованного представления проста, эффективна и имеет меньше ограничений, чем обычная перезапись запросов материализованного представления типа SPJG. При правильном использовании эта функция может значительно ускорить производительность запросов.

Текстовая перезапись материализованного представления не ограничивается операторами типа SPJG. Она также поддерживает операторы, такие как Union, Window, Order, Limit и CTE.

Использование

Текстовая перезапись материализованного представления включена по умолчанию. Вы можете вручную отключить эту функцию, установив переменную `enable_materialized_view_text_match_rewrite` в `false`.

Элемент конфигурации FE `enable_materialized_view_text_based_rewrite` контролирует, строить ли абстрактное синтаксическое дерево при создании асинхронного материализованного представления. Эта функция также включена по умолчанию. Установка этого элемента в `false` отключит текстовую перезапись материализованного представления на системном уровне.

Переменная `materialized_view_subuqery_text_match_max_count` контролирует максимальное количество раз для сравнения абстрактных синтаксических деревьев материализованного представления и подзапросов. Значение по умолчанию — `4`. Увеличение этого значения также увеличит потребление времени оптимизатора.

Обратите внимание, что только когда материализованное представление соответствует требованию своевременности (согласованность данных), оно может использоваться для текстовой перезаписи запросов. Вы можете вручную установить правило проверки согласованности, используя свойство `query_rewrite_consistency` при создании материализованного представления. Для получения дополнительной информации см. CREATE MATERIALIZED VIEW.

Случаи использования

Запросы подходят для текстовой перезаписи материализованного представления в следующих сценариях:

- Исходный запрос соответствует определению материализованного представления.
- Подзапрос исходного запроса соответствует определению материализованного представления.

По сравнению с обычной перезаписью запросов материализованного представления типа SPJG, текстовая перезапись материализованного представления поддерживает более сложные запросы, например, многоуровневые агрегации.

К СВЕДЕНИЮ

- Рекомендуется инкапсулировать запрос для соответствия в подзапросе исходного запроса.
- Пожалуйста, не инкапсулируйте операторы ORDER BY в определении материализованного представления или подзапросе исходного запроса. В противном случае запрос не может быть переписан, потому что операторы ORDER BY в подзапросе исключаются по умолчанию.

Например, вы можете создать следующее материализованное представление:

```
CREATE MATERIALIZED VIEW mv1 REFRESH MANUAL AS
SELECT
  user_id,
  count(1)
FROM (
  SELECT
    user_id,
    time,
    bitmap_union(to_bitmap(tag_id)) AS a
  FROM user_tags
  GROUP BY
    user_id,
    time) t
GROUP BY user_id;
```

Материализованное представление может переписать оба следующих запроса:

```
SELECT
  user_id,
  count(1)
FROM (
  SELECT
    user_id,
    time,
    bitmap_union(to_bitmap(tag_id)) AS a
  FROM user_tags
  GROUP BY
    user_id,
    time) t
GROUP BY user_id;
SELECT count(1)
FROM
(
    SELECT
      user_id,
      count(1)
    FROM (
      SELECT
        user_id,
        time,
        bitmap_union(to_bitmap(tag_id)) AS a
      FROM user_tags
      GROUP BY
        user_id,
        time) t
    GROUP BY user_id
)m;
```

Однако материализованное представление не может переписать следующий запрос, потому что исходный запрос содержит оператор ORDER BY:

```
SELECT
  user_id,
  count(1)
FROM (
  SELECT
    user_id,
    time,
    bitmap_union(to_bitmap(tag_id)) AS a
  FROM user_tags
  GROUP BY
    user_id,
    time) t
GROUP BY user_id
ORDER BY user_id;
```

Настройка перезаписи запросов

Вы можете настроить перезапись запросов асинхронного материализованного представления через следующие переменные сеанса:

Переменная	По умолчанию	Описание
enable_materialized_view_union_rewrite	true	Логическое значение для контроля, включать ли перезапись запросов Union материализованного представления.
enable_rule_based_materialized_view_rewrite	true	Логическое значение для контроля, включать ли перезапись запросов материализованного представления на основе правил. Эта переменная в основном используется в перезаписи запросов одной таблицы.
nested_mv_rewrite_max_level	3	Максимальные уровни вложенных материализованных представлений, которые могут использоваться для перезаписи запросов. Тип: INT. Диапазон: [1, +∞). Значение `1` указывает, что материализованные представления, созданные на других материализованных представлениях, не будут использоваться для перезаписи запросов.

Проверка, переписан ли запрос

Вы можете проверить, переписан ли ваш запрос, просмотрев его план запроса с помощью оператора EXPLAIN. Если поле `TABLE` в разделе `OlapScanNode` показывает имя

соответствующего материализованного представления, это означает, что запрос был переписан на основе материализованного представления.

```
mysql> EXPLAIN SELECT
    order_id, sum(goods.price) AS total
    FROM order_list INNER JOIN goods
    ON goods.item_id1 = order_list.item_id2
    GROUP BY order_id;
+-------------------------------------+
| Explain String                      |
+-------------------------------------+
| PLAN FRAGMENT 0                     |
|  OUTPUT EXPRS:1: order_id | 8: sum |
|   PARTITION: RANDOM                 |
|                                     |
|   RESULT SINK                       |
|                                     |
|   1:Project                         |
|   |  <slot 1> : 9: order_id         |
|   |  <slot 8> : 10: total           |
|   |                                 |
|   0:OlapScanNode                    |
|      TABLE: order_mv                |
|      PREAGGREGATION: ON             |
|      partitions=1/1                 |
|      rollup: order_mv               |
|      tabletRatio=0/12               |
|      tabletList=                    |
|      cardinality=3                  |
|      avgRowSize=4.0                 |
|      numNodes=0                     |
+-------------------------------------+
20 rows in set (0.01 sec)
```

Отключение перезаписи запросов

По умолчанию Selena включает перезапись запросов для асинхронных материализованных представлений, созданных на основе default catalog. Вы можете отключить эту функцию, установив переменную сеанса `enable_materialized_view_rewrite` в `false`.

Для асинхронных материализованных представлений, созданных на основе внешнего каталога, вы можете отключить эту функцию, установив свойство материализованного

представления `force_external_table_query_rewrite` в `false`, используя ALTER MATERIALIZED VIEW.

Ограничения

В отношении перезаписи запросов материализованного представления Selena в настоящее время имеет следующие ограничения:

- Selena не поддерживает перезапись запросов с недетерминированными функциями, включая rand, random, uuid и sleep.
- Selena не поддерживает перезапись запросов с оконными функциями.
- Материализованные представления, определенные с операторами, содержащими LIMIT, ORDER BY, UNION, EXCEPT, INTERSECT, MINUS, GROUPING SETS, WITH CUBE или WITH ROLLUP, не могут использоваться для перезаписи запросов.
- Строгая согласованность результатов запросов не гарантируется между базовыми таблицами и материализованными представлениями, построенными на внешних каталогах.
- Асинхронные материализованные представления, созданные на базовых таблицах в JDBC catalog, не поддерживают перезапись запросов.

В отношении перезаписи запросов материализованного представления на основе представления Selena в настоящее время имеет следующие ограничения:

- В настоящее время Selena не поддерживает перезапись Partition Union.

- Перезапись запросов не поддерживается, если представление содержит случайные функции, включая rand(), random(), uuid() и sleep().

- Перезапись запросов не поддерживается, если представление содержит столбцы с одинаковыми именами. Вы должны назначить разные псевдонимы для столбцов с одинаковыми именами.

- Представления, которые используются для создания материализованного представления, должны содержать по крайней мере один столбец следующих типов данных: целочисленные типы, типы дат и строковые типы. Например, вы не можете создать материализованное, которое запрашивает представление, потому что `total_cost` является столбцом типа DOUBLE.

```
CREATE VIEW v1
AS
SELECT sum(cost) AS total_cost
FROM t1;
```

Ускорение запросов к озеру данных с помощью материализованных представлений

Этот раздел описывает, как оптимизировать производительность запросов к вашему озеру данных с помощью асинхронных материализованных представлений Selena.

Selena предлагает готовые возможности запросов к озеру данных, которые весьма эффективны для разведочных запросов и анализа данных в озере. В большинстве сценариев Data Cache может обеспечить кэширование файлов на уровне блоков, избегая снижения производительности, вызванного колебаниями удаленного хранилища и большим количеством операций I/O.

Однако, когда дело доходит до построения сложных и эффективных отчетов с использованием данных из озера или дальнейшего ускорения этих запросов, вы все еще можете столкнуться с проблемами производительности. С помощью асинхронных материализованных представлений вы можете достичь более высокой одновременности и лучшей производительности запросов для отчетов и приложений данных в озере.

Обзор

Selena поддерживает построение асинхронных материализованных представлений на основе внешних каталогов, таких как Hive catalog, Iceberg catalog, Hudi catalog, JDBC catalog и Paimon catalog. Материализованные представления на основе внешних каталогов особенно полезны в следующих сценариях:

- **Прозрачное ускорение отчетов озера данных**

  Чтобы обеспечить производительность запросов отчетов озера данных, инженеры данных обычно должны тесно работать с аналитиками данных, чтобы изучить логику построения слоя ускорения для отчетов. Если слой ускорения требует дальнейших обновлений, они должны соответствующим образом обновить логику построения, расписания обработки и операторы запросов.

Благодаря возможности перезаписи запросов материализованных представлений, ускорение отчетов может быть сделано прозрачным и незаметным для пользователей. Когда обнаруживаются медленные запросы, инженеры данных могут проанализировать паттерн медленных запросов и создать материализованные представления по требованию. Запросы на стороне приложения затем интеллектуально переписываются и прозрачно ускоряются материализованным представлением, что позволяет быстро улучшить производительность запросов без изменения логики бизнес-приложения или оператора запроса.

- **Инкрементальное вычисление real-time данных, связанных с историческими данными**

  Предположим, что ваше бизнес-приложение требует связывания real-time данных в нативных таблицах Selena и исторических данных в озере данных для инкрементальных вычислений. В этой ситуации материализованные представления могут предоставить простое решение. Например, если таблица фактов real-time является нативной таблицей в Selena, а таблица измерений хранится в озере данных, вы можете легко выполнять инкрементальные вычисления, создав материализованные представления, которые связывают нативную таблицу с таблицей во внешних источниках данных.

- **Быстрое построение слоев метрик**

  Вычисление и обработка метрик могут столкнуться с трудностями при работе с данными высокой размерности. Вы можете использовать материализованные представления, которые позволяют выполнять предварительную агрегацию данных и rollup, чтобы создать относительно легкий слой метрик. Более того, материализованные представления могут обновляться автоматически, что дополнительно снижает сложность вычисления метрик.

Материализованные представления, Data Cache и нативные таблицы в Selena — все это эффективные методы достижения значительного повышения производительности запросов. В следующей таблице сравниваются их основные различия:

	Data Cache	Материализованное представление	Нативная таблица
Загрузка и обновление данных	Запросы автоматически	Задачи обновления запускаются автоматически.	Поддерживает различные методы импорта, но требует

	Data Cache	Материализованное представление	Нативная таблица
	запускают кэширование данных.		ручного обслуживания задач импорта
Гранулярность кэширования данных	• Поддерживает кэширование данных на уровне блоков • Следует механизму вытеснения кэша LRU • Результаты вычислений не кэшируются	Хранит предварительно вычисленные результаты запросов	Хранит данные на основе схемы таблицы
Производительность запросов	Data Cache ≤ Материализованное представление = Нативная таблица		
Оператор запроса	• Не нужно изменять операторы запросов к озеру данных • Когда запросы попадают в кэш, происходит вычисление.	• Не нужно изменять операторы запросов к озеру данных • Использует перезапись запросов для повторного использования предварительно вычисленных результатов	Требует изменения операторов запросов для запроса к нативной таблице

По сравнению с прямым запросом данных из озера или загрузкой данных в нативные таблицы, материализованные представления предлагают несколько уникальных преимуществ:

- **Ускорение с локальным хранилищем**: Материализованные представления могут использовать преимущества ускорения Selena с локальным хранилищем, такие как индексы, partitioning, bucketing и collocate groups, что приводит к лучшей производительности запросов по сравнению с прямым запросом данных из озера данных.
- **Нулевое обслуживание задач загрузки**: Материализованные представления обновляют данные прозрачно через автоматические задачи обновления. Нет необходимости обслуживать задачи загрузки для выполнения запланированных

обновлений данных. Кроме того, материализованные представления на основе Hive, Iceberg и Paimon catalog могут обнаруживать изменения данных и выполнять инкрементальные обновления на уровне partition.
- **Интеллектуальная перезапись запросов**: Запросы могут быть прозрачно переписаны для использования материализованных представлений. Вы можете получить выгоду от ускорения мгновенно, без необходимости изменять операторы запросов, используемые вашим приложением.

Поэтому мы рекомендуем использовать материализованные представления в следующих сценариях:

- Даже когда включен Data Cache, производительность запросов не соответствует вашим требованиям к задержке запроса и одновременности.
- Запросы включают многократно используемые компоненты, такие как фиксированные агрегатные функции или паттерны join.
- Данные организованы в partitions, в то время как запросы включают агрегацию на относительно высоком уровне (например, агрегация по дням).

В следующих сценариях мы рекомендуем приоритизировать ускорение через Data Cache:

- Запросы не имеют много многократно используемых компонентов и могут сканировать любые данные из озера данных.
- Удаленное хранилище имеет значительные колебания или нестабильность, что потенциально может повлиять на доступ.

Создание материализованных представлений на основе внешних каталогов

Создание материализованного представления для таблиц во внешних каталогах аналогично созданию материализованного представления для нативных таблиц Selena. Вам нужно только установить подходящую стратегию обновления в соответствии с используемым источником данных и вручную включить перезапись запросов для материализованных представлений на основе внешних каталогов.

> **ПРИМЕЧАНИЕ**
>
> Материализованные представления внешних таблиц не поддерживают автоматическое обновление **запускаемое изменениями данных базовой таблицы**. Они поддерживают только асинхронное обновление **с фиксированным интервалом** и ручное обновление.

Выбор подходящей стратегии обновления

В настоящее время Selena не может обнаруживать изменения данных на уровне partition в Hudi catalogs. Поэтому выполняется полное обновление после запуска задачи.

Для Hive Catalog, Iceberg Catalog (начиная с v1.5.2), JDBC catalog (начиная с v1.5.2, только для MySQL таблиц с range-partitioned) и Paimon Catalog (начиная с v1.5.2), Selena поддерживает обнаружение изменений данных на уровне partition. В результате Selena может:

- Обновлять только partitions с изменениями данных, чтобы избежать полного обновления, снижая потребление ресурсов, вызванное обновлением.
- Обеспечивать согласованность данных в некоторой степени во время перезаписи запросов. Если есть изменения данных в базовой таблице в озере данных, запрос не будет переписан для использования материализованного представления.

> **ПОДСКАЗКА**
>
> Вы все еще можете выбрать допустить определенный уровень несогласованности данных, установив свойство `mv_rewrite_staleness_second` при создании материализованного представления. Для получения дополнительной информации см. CREATE MATERIALIZED VIEW.

Обратите внимание, что если вам нужно обновлять по partition, ключи partitioning материализованного представления должны быть включены в ключи базовой таблицы.

Начиная с v1.5.2, Selena поддерживает создание partitioned материализованных представлений для Iceberg таблиц с Partition Transforms, и материализованные представления partitioned по столбцу после трансформации. В настоящее время

поддерживаются только Iceberg таблицы с трансформациями `identity`, `year`, `month`, `day` или `hour`.

Следующий пример показывает определение Iceberg таблицы с трансформацией partition `day` и создает материализованное представление с выровненными partitions на ее основе:

```
-- Определение Iceberg таблицы.
CREATE TABLE spark_catalog.test.iceberg_sample_datetime_day (
  id         BIGINT,
  data       STRING,
  category   STRING,
  ts         TIMESTAMP)
USING iceberg
PARTITIONED BY (days(ts))

-- Создание материализованного представления для Iceberg таблицы.
CREATE MATERIALIZED VIEW `test_iceberg_datetime_day_mv` (`id`, `data`,
`category`, `ts`)
PARTITION BY (`ts`)
DISTRIBUTED BY HASH(`id`)
REFRESH MANUAL
AS
SELECT
  `iceberg_sample_datetime_day`.`id`,
  `iceberg_sample_datetime_day`.`data`,
  `iceberg_sample_datetime_day`.`category`,
  `iceberg_sample_datetime_day`.`ts`
FROM `iceberg`.`test`.`iceberg_sample_datetime_day`;
```

Для Hive catalogs вы можете включить функцию обновления кэша метаданных Hive, чтобы позволить Selena обнаруживать изменения данных на уровне partition. Когда эта функция включена, Selena периодически обращается к Hive Metastore Service (HMS) или AWS Glue для проверки информации метаданных о недавно запрошенных горячих данных.

Чтобы включить функцию обновления кэша метаданных Hive, вы можете установить следующий элемент динамической конфигурации FE, используя ADMIN SET FRONTEND CONFIG:

Элементы конфигурации

enable_background_refresh_connector_metadata

По умолчанию: true в v1.5.2, false в v1.5.2
Описание: Включать ли периодическое обновление кэша метаданных Hive. После включения Selena опрашивает metastore (Hive Metastore или AWS Glue) вашего Hive cluster и обновляет кэшированные метаданные часто запрашиваемых Hive catalogs для восприятия изменений данных. True указывает на включение обновления кэша метаданных Hive, а false указывает на отключение.

background_refresh_metadata_interval_millis

По умолчанию: 600000 (10 минут)
Описание: Интервал между двумя последовательными обновлениями кэша метаданных Hive. Единица измерения: миллисекунды.

background_refresh_metadata_time_secs_since_last_access_secs

По умолчанию: 86400 (24 часа)
Описание: Время истечения задачи обновления кэша метаданных Hive. Для Hive catalog, к которому был осуществлен доступ, если к нему не было доступа более указанного времени, Selena прекращает обновление его кэшированных метаданных. Для Hive catalog, к которому не было доступа, Selena не будет обновлять его кэшированные метаданные. Единица измерения: секунды.

Начиная с v1.5.2, Selena поддерживает обнаружение изменений данных для Iceberg Catalog на уровне partition. В настоящее время поддерживаются только Iceberg V1 таблицы.

Включение перезаписи запросов для материализованных представлений на основе внешних каталогов

По умолчанию Selena не поддерживает перезапись запросов для материализованных представлений, построенных на Hudi и JDBC catalogs, потому что перезапись запросов в этом сценарии не может обеспечить строгую согласованность результатов. Вы можете включить эту функцию, установив свойство `force_external_table_query_rewrite` в `true` при создании материализованного представления. Для материализованных представлений, построенных на таблицах в Hive catalogs, перезапись запросов включена по умолчанию.

Пример:

```
CREATE MATERIALIZED VIEW ex_mv_par_tbl
PARTITION BY emp_date
DISTRIBUTED BY hash(empid)
PROPERTIES (
"force_external_table_query_rewrite" = "true"
)
AS
select empid, deptno, emp_date
from `hive_catalog`.`emp_db`.`emps_par_tbl`
where empid < 5;
```

В сценариях, включающих перезапись запросов, если вы используете очень сложный оператор запроса для построения материализованного представления, мы рекомендуем разделить оператор запроса и построить несколько простых материализованных представлений вложенным образом. Вложенные материализованные представления более универсальны и могут охватывать более широкий спектр паттернов запросов.

Лучшие практики

В реальных бизнес-сценариях вы можете идентифицировать запросы с высокой задержкой выполнения и потреблением ресурсов, анализируя журналы аудита или журналы больших запросов. Вы можете дополнительно использовать query profiles для точного определения конкретных этапов, где запрос медленный. Следующие разделы предоставляют инструкции и примеры о том, как повысить производительность запросов озера данных с помощью материализованных представлений.

Случай первый: Ускорение вычислений join в озере данных

Вы можете использовать материализованные представления для ускорения запросов join в озере данных.

Предположим, что следующие запросы к вашему Hive catalog особенно медленные:

```sql
--Q1
SELECT SUM(lo_extendedprice * lo_discount) AS REVENUE
FROM hive.ssb_1g_csv.lineorder, hive.ssb_1g_csv.dates
WHERE
    lo_orderdate = d_datekey
    AND d_year = 1993
    AND lo_discount BETWEEN 1 AND 3
    AND lo_quantity < 25;

--Q2
SELECT SUM(lo_extendedprice * lo_discount) AS REVENUE
FROM hive.ssb_1g_csv.lineorder, hive.ssb_1g_csv.dates
WHERE
    lo_orderdate = d_datekey
    AND d_yearmonth = 'Jan1994'
    AND lo_discount BETWEEN 4 AND 6
    AND lo_quantity BETWEEN 26 AND 35;

--Q3
SELECT SUM(lo_revenue), d_year, p_brand
FROM hive.ssb_1g_csv.lineorder, hive.ssb_1g_csv.dates,
hive.ssb_1g_csv.part, hive.ssb_1g_csv.supplier
WHERE
    lo_orderdate = d_datekey
    AND lo_partkey = p_partkey
    AND lo_suppkey = s_suppkey
    AND p_brand BETWEEN 'MFGR#2221' AND 'MFGR#2228'
    AND s_region = 'ASIA'
GROUP BY d_year, p_brand
ORDER BY d_year, p_brand;
```

Анализируя их query profiles, вы можете заметить, что время выполнения запроса в основном тратится на hash join между таблицей `lineorder` и другими таблицами измерений по столбцу `lo_orderdate`.

Здесь Q1 и Q2 выполняют агрегацию после join `lineorder` и `dates`, в то время как Q3 выполняет агрегацию после join `lineorder`, `dates`, `part` и `supplier`.

Следовательно, вы можете использовать возможность View Delta Join rewrite Selena для построения материализованного представления, которое объединяет `lineorder`, `dates`, `part` и `supplier`.

```
CREATE MATERIALIZED VIEW lineorder_flat_mv
DISTRIBUTED BY HASH(LO_ORDERDATE, LO_ORDERKEY) BUCKETS 48
PARTITION BY LO_ORDERDATE
REFRESH ASYNC EVERY(INTERVAL 1 DAY)
PROPERTIES (
    -- Указание уникальных ограничений.
    "unique_constraints" = "
    hive.ssb_1g_csv.supplier.s_suppkey;
    hive.ssb_1g_csv.part.p_partkey;
    hive.ssb_1g_csv.dates.d_datekey",
    -- Указание Foreign Keys.
    "foreign_key_constraints" = "
    hive.ssb_1g_csv.lineorder(lo_partkey) REFERENCES
hive.ssb_1g_csv.part(p_partkey);
    hive.ssb_1g_csv.lineorder(lo_suppkey) REFERENCES
hive.ssb_1g_csv.supplier(s_suppkey);
    hive.ssb_1g_csv.lineorder(lo_orderdate) REFERENCES
hive.ssb_1g_csv.dates(d_datekey)",
    -- Включение перезаписи запросов для материализованного представления
на основе внешнего каталога.
    "force_external_table_query_rewrite" = "TRUE"
)
AS SELECT
       l.LO_ORDERDATE AS LO_ORDERDATE,
       l.LO_ORDERKEY AS LO_ORDERKEY,
       l.LO_PARTKEY AS LO_PARTKEY,
       l.LO_SUPPKEY AS LO_SUPPKEY,
       l.LO_QUANTITY AS LO_QUANTITY,
       l.LO_EXTENDEDPRICE AS LO_EXTENDEDPRICE,
       l.LO_DISCOUNT AS LO_DISCOUNT,
       l.LO_REVENUE AS LO_REVENUE,
       s.S_REGION AS S_REGION,
       p.P_BRAND AS P_BRAND,
       d.D_YEAR AS D_YEAR,
       d.D_YEARMONTH AS D_YEARMONTH
   FROM hive.ssb_1g_csv.lineorder AS l
            INNER JOIN hive.ssb_1g_csv.supplier AS s ON s.S_SUPPKEY =
l.LO_SUPPKEY
            INNER JOIN hive.ssb_1g_csv.part AS p ON p.P_PARTKEY =
l.LO_PARTKEY
            INNER JOIN hive.ssb_1g_csv.dates AS d ON l.LO_ORDERDATE =
d.D_DATEKEY;
```

Случай второй: Ускорение агрегаций и агрегаций на joins в озере данных

Материализованные представления могут использоваться для ускорения агрегатных запросов, независимо от того, выполняются ли они для одной таблицы или включают несколько таблиц.

- Агрегатный запрос для одной таблицы

  Для типичных запросов к одной таблице их query profile покажет, что узел AGGREGATE потребляет значительное количество времени. Вы можете использовать общие агрегатные операторы для построения материализованных представлений.

  Предположим, что следующий запрос медленный:

  ```
  --Q4
  SELECT
  lo_orderdate, count(distinct lo_orderkey)
  FROM hive.ssb_1g_csv.lineorder
  GROUP BY lo_orderdate
  ORDER BY lo_orderdate limit 100;
  ```

  Q4 вычисляет ежедневное количество уникальных заказов. Поскольку вычисление count distinct может быть вычислительно затратным, вы можете создать следующие два типа материализованных представлений для ускорения:

```
CREATE MATERIALIZED VIEW mv_2_1
DISTRIBUTED BY HASH(lo_orderdate)
PARTITION BY LO_ORDERDATE
REFRESH ASYNC EVERY(INTERVAL 1 DAY)
AS
SELECT
lo_orderdate, count(distinct lo_orderkey)
FROM hive.ssb_1g_csv.lineorder
GROUP BY lo_orderdate;

CREATE MATERIALIZED VIEW mv_2_2
DISTRIBUTED BY HASH(lo_orderdate)
PARTITION BY LO_ORDERDATE
REFRESH ASYNC EVERY(INTERVAL 1 DAY)
AS
SELECT
-- lo_orderkey должен быть типа BIGINT, чтобы его можно было
использовать для перезаписи запросов.
lo_orderdate, bitmap_union(to_bitmap(lo_orderkey))
FROM hive.ssb_1g_csv.lineorder
GROUP BY lo_orderdate;
```

Обратите внимание, что в этом контексте не создавайте материализованные представления с операторами LIMIT и ORDER BY, чтобы избежать неудач перезаписи. Для получения дополнительной информации об ограничениях перезаписи запросов см. Query rewrite with materialized views - Limitations.

- Многотабличный агрегатный запрос

  В сценариях, включающих агрегации результатов join, вы можете создать вложенные материализованные представления на основе существующих материализованных представлений, которые объединяют таблицы для дальнейшей агрегации результатов join. Например, на основе примера из Случая первого, вы можете создать следующее материализованное представление для ускорения Q1 и Q2, потому что их паттерны агрегации похожи:

```
CREATE MATERIALIZED VIEW mv_2_3
DISTRIBUTED BY HASH(lo_orderdate)
PARTITION BY LO_ORDERDATE
REFRESH ASYNC EVERY(INTERVAL 1 DAY)
AS
SELECT
lo_orderdate, lo_discount, lo_quantity, d_year, d_yearmonth,
SUM(lo_extendedprice * lo_discount) AS REVENUE
FROM lineorder_flat_mv
GROUP BY lo_orderdate, lo_discount, lo_quantity, d_year, d_yearmonth;
```

Конечно, возможно выполнять как вычисления join, так и агрегации в рамках одного материализованного представления. Хотя эти типы материализованных представлений могут иметь меньше возможностей для перезаписи запросов (из-за их специфических вычислений), они обычно занимают меньше места для хранения после агрегации. Ваш выбор может быть основан на вашем конкретном случае использования.

```
CREATE MATERIALIZED VIEW mv_2_4
DISTRIBUTED BY HASH(lo_orderdate)
PARTITION BY LO_ORDERDATE
REFRESH ASYNC EVERY(INTERVAL 1 DAY)
PROPERTIES (
    "force_external_table_query_rewrite" = "TRUE"
)
AS
SELECT lo_orderdate, lo_discount, lo_quantity, d_year, d_yearmonth,
SUM(lo_extendedprice * lo_discount) AS REVENUE
FROM hive.ssb_1g_csv.lineorder, hive.ssb_1g_csv.dates
WHERE lo_orderdate = d_datekey
GROUP BY lo_orderdate, lo_discount, lo_quantity, d_year, d_yearmonth;
```

Случай третий: Ускорение joins на агрегациях в озере данных

В некоторых сценариях может быть необходимо сначала выполнить вычисления агрегации для одной таблицы, а затем выполнить запросы join с другими таблицами. Чтобы полностью использовать возможности перезаписи запросов Selena, мы рекомендуем построить вложенные материализованные представления. Например:

```
--Q5
SELECT * FROM  (
    SELECT
      l.lo_orderkey, l.lo_orderdate, c.c_custkey, c_region,
sum(l.lo_revenue)
    FROM
      hive.ssb_1g_csv.lineorder l
      INNER JOIN (
        SELECT distinct c_custkey, c_region
        from
          hive.ssb_1g_csv.customer
        WHERE
          c_region IN ('ASIA', 'AMERICA')
      ) c ON l.lo_custkey = c.c_custkey
      GROUP BY  l.lo_orderkey, l.lo_orderdate, c.c_custkey, c_region
  ) c1
WHERE
  lo_orderdate = '19970503';
```

Q5 сначала выполняет агрегатный запрос для таблицы `customer`, а затем выполняет join и агрегацию с таблицей `lineorder`. Похожие запросы могут включать различные фильтры по `c_region` и `lo_orderdate`. Чтобы использовать возможности перезаписи запросов, вы можете создать два материализованных представления, одно для агрегации, а другое для join.

```
--mv_3_1
CREATE MATERIALIZED VIEW mv_3_1
DISTRIBUTED BY HASH(c_custkey)
REFRESH ASYNC EVERY(INTERVAL 1 DAY)
PROPERTIES (
    "force_external_table_query_rewrite" = "TRUE"
)
AS
SELECT distinct c_custkey, c_region from hive.ssb_1g_csv.customer;

--mv_3_2
CREATE MATERIALIZED VIEW mv_3_2
DISTRIBUTED BY HASH(lo_orderdate)
PARTITION BY LO_ORDERDATE
REFRESH ASYNC EVERY(INTERVAL 1 DAY)
PROPERTIES (
    "force_external_table_query_rewrite" = "TRUE"
)
AS
SELECT l.lo_orderdate, l.lo_orderkey, mv.c_custkey, mv.c_region,
sum(l.lo_revenue)
FROM hive.ssb_1g_csv.lineorder l
INNER JOIN mv_3_1 mv
ON l.lo_custkey = mv.c_custkey
GROUP BY l.lo_orderkey, l.lo_orderdate, mv.c_custkey, mv.c_region;
```

Случай четвертый: Разделение горячих и холодных данных для real-time данных и исторических данных в озере данных

Рассмотрим следующий сценарий: данные, обновленные за последние три дня, записываются напрямую в Selena, в то время как менее свежие данные проверяются и пакетно записываются в Hive. Однако все еще есть запросы, которые могут включать данные за последние семь дней. В этом случае вы можете создать простую модель с материализованными представлениями для автоматического истечения данных.

```
CREATE MATERIALIZED VIEW mv_4_1
DISTRIBUTED BY HASH(lo_orderdate)
PARTITION BY LO_ORDERDATE
REFRESH ASYNC EVERY(INTERVAL 1 DAY)
AS
SELECT lo_orderkey, lo_orderdate, lo_revenue
FROM hive.ssb_1g_csv.lineorder
WHERE lo_orderdate<=current_date()
AND lo_orderdate>=date_add(current_date(), INTERVAL -4 DAY);
```

Вы можете дополнительно построить представления или материализованные представления на его основе в соответствии с логикой приложений верхнего уровня.

Создание разбитого материализованного представления

В этом разделе описано, как создать разбитое материализованное представление для различных вариантов использования.

Обзор

Асинхронные материализованные представления Selena поддерживают различные стратегии partitioning и функции, которые позволяют достичь следующих эффектов:

- **Инкрементное построение**

  При создании разбитого материализованного представления вы можете настроить задачу создания на обновление partition партиями, чтобы избежать чрезмерного потребления ресурсов.

- **Инкрементное обновление**

  Вы можете настроить задачу обновления на обновление только соответствующих partition материализованного представления, когда она обнаруживает изменения данных в определенных partition базовой таблицы. Обновление на уровне partition может значительно предотвратить потерю ресурсов, используемых для обновления всего материализованного представления.

- **Частичная материализация**

Вы можете установить TTL для partition материализованного представления, что позволяет частичную материализацию данных.

- **Прозрачная перезапись запросов**

  Запросы могут быть прозрачно перезаписаны на основе только тех обновленных partition материализованного представления. Partition, которые считаются устаревшими, не будут участвовать в плане запроса, и запрос будет выполнен на базовых таблицах для гарантии согласованности данных.

Ограничения

Разбитое материализованное представление может быть создано только на основе разбитой базовой таблицы (обычно таблицы фактов). Только путем отображения отношения partition между базовой таблицей и материализованным представлением вы можете построить синергию между ними.

В настоящее время Selena поддерживает создание разбитых материализованных представлений на таблицах из следующих источников данных:

- **Таблицы Selena OLAP в catalog по умолчанию**
  - Поддерживаемая стратегия partitioning: Range partitioning, List Partitioning и Expression Partitioning
  - Поддерживаемые типы данных для Partitioning Key: INT, DATE, DATETIME и STRING
  - Поддерживаемые типы таблиц: Primary Key, Duplicate Key, Aggregate Key и Unique Key
  - Поддерживается как в shared-nothing cluster, так и в shared-data cluster
- **Таблицы в Hive Catalog, Hudi Catalog, Iceberg Catalog и Paimon Catalog**
  - Поддерживаемый уровень partitioning: Основной уровень
  - Поддерживаемые типы данных для Partitioning Key: INT, DATE, DATETIME и STRING

ПРИМЕЧАНИЕ

- Вы не можете создать разбитое материализованное представление на основе неразбитой базовой (фактовой) таблицы.
- Для таблиц Selena OLAP две соседние partition базовой таблицы должны иметь последовательные диапазоны.
- Для многоуровневых разбитых базовых таблиц во внешних catalog только путь partitioning основного уровня может быть использован для создания разбитого материализованного представления. Например, для таблицы, разбитой в формате `yyyyMMdd/hour`, вы можете построить материализованное представление, разбитое только по `yyyyMMdd`.
- Начиная с v1.5.2, Selena поддерживает создание разбитых материализованных представлений на таблицах Iceberg с Partition Transforms, и материализованные представления разбиваются по столбцу после преобразования. Для получения дополнительной информации см. Data lake query acceleration with materialized views - Choose a suitable refresh strategy.

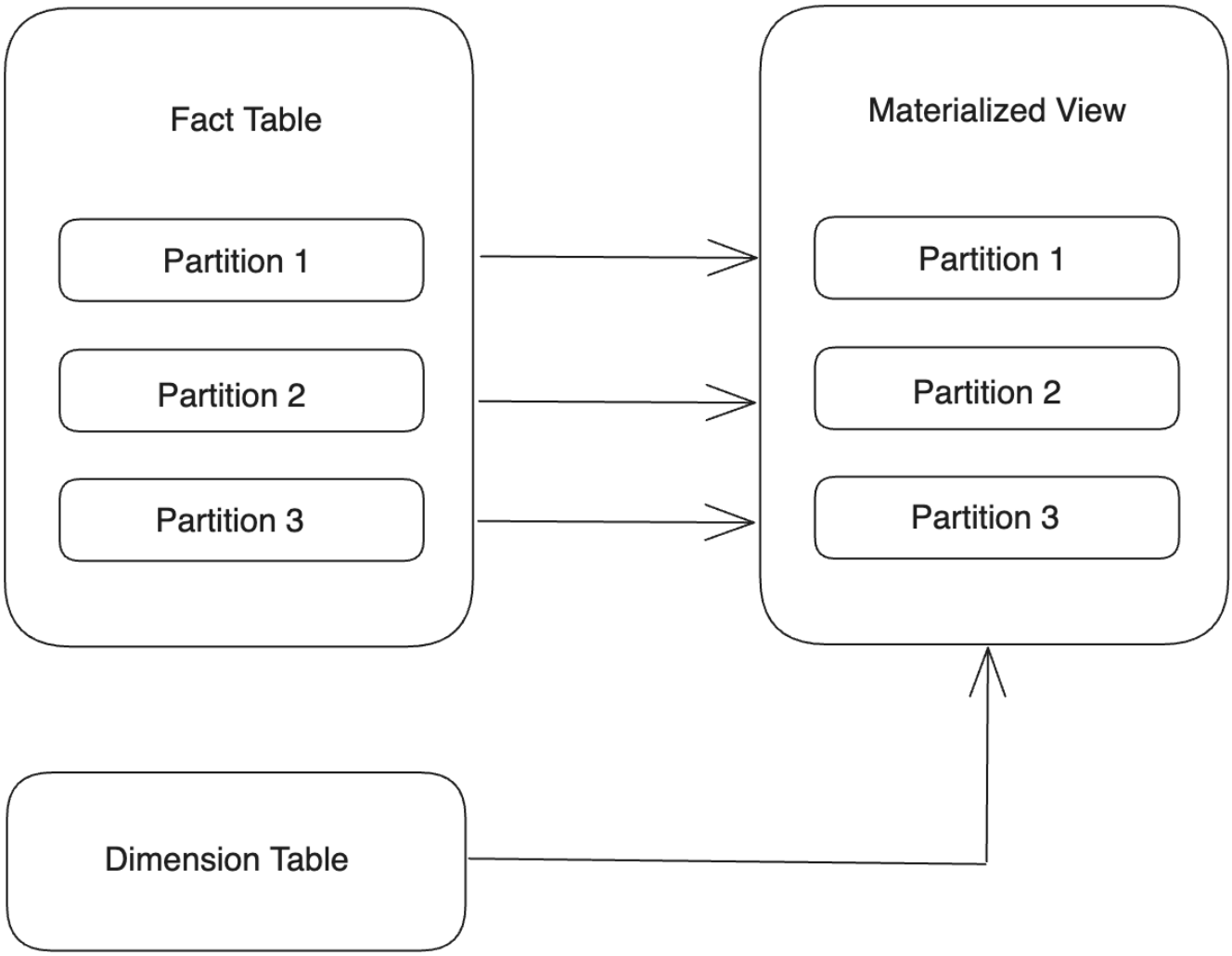
Варианты использования

Предположим, что есть базовые таблицы следующим образом:

```
CREATE TABLE IF NOT EXISTS par_tbl1 (
  datekey      DATE,       -- Столбец даты типа DATE, используемый как
Partitioning Key.
  k1           STRING,
  v1           INT,
  v2           INT
)
ENGINE=olap
PARTITION BY RANGE (datekey) (
  START ("2021-01-01") END ("2021-01-04") EVERY (INTERVAL 1 DAY)
)
DISTRIBUTED BY HASH(k1);

CREATE TABLE IF NOT EXISTS par_tbl2 (
  datekey      STRING,     -- Столбец даты типа STRING, используемый как
Partitioning Key.
  k1           STRING,
  v1           INT,
  v2           INT
)
ENGINE=olap
PARTITION BY RANGE (str2date(datekey, '%Y-%m-%d')) (
  START ("2021-01-01") END ("2021-01-04") EVERY (INTERVAL 1 DAY)
)
DISTRIBUTED BY HASH(k1);

CREATE TABLE IF NOT EXISTS par_tbl3 (
  datekey_new  DATE,       -- Эквивалентный столбец с par_tbl1.datekey.
  k1           STRING,
  v1           INT,
  v2           INT
)
ENGINE=olap
PARTITION BY RANGE (datekey_new) (
  START ("2021-01-01") END ("2021-01-04") EVERY (INTERVAL 1 DAY)
)
DISTRIBUTED BY HASH(k1);
```

Выравнивание partition один к одному

Вы можете создать материализованное представление, partition которого соответствуют partition базовой таблицы один к одному, используя тот же Partitioning Key.



- Если Partitioning Key базовой таблицы имеет тип DATE или DATETIME, вы можете напрямую указать тот же Partitioning Key для материализованного представления.

```
PARTITION BY <base_table_partitioning_column>
```

Пример:

```
CREATE MATERIALIZED VIEW par_mv1
REFRESH ASYNC
PARTITION BY datekey
AS
SELECT
  k1,
  sum(v1) AS SUM,
  datekey
FROM par_tbl1
GROUP BY datekey, k1;
```

- Если Partitioning Key базовой таблицы имеет тип STRING, вы можете использовать функцию str2date для преобразования строки даты в тип DATE или DATETIME.

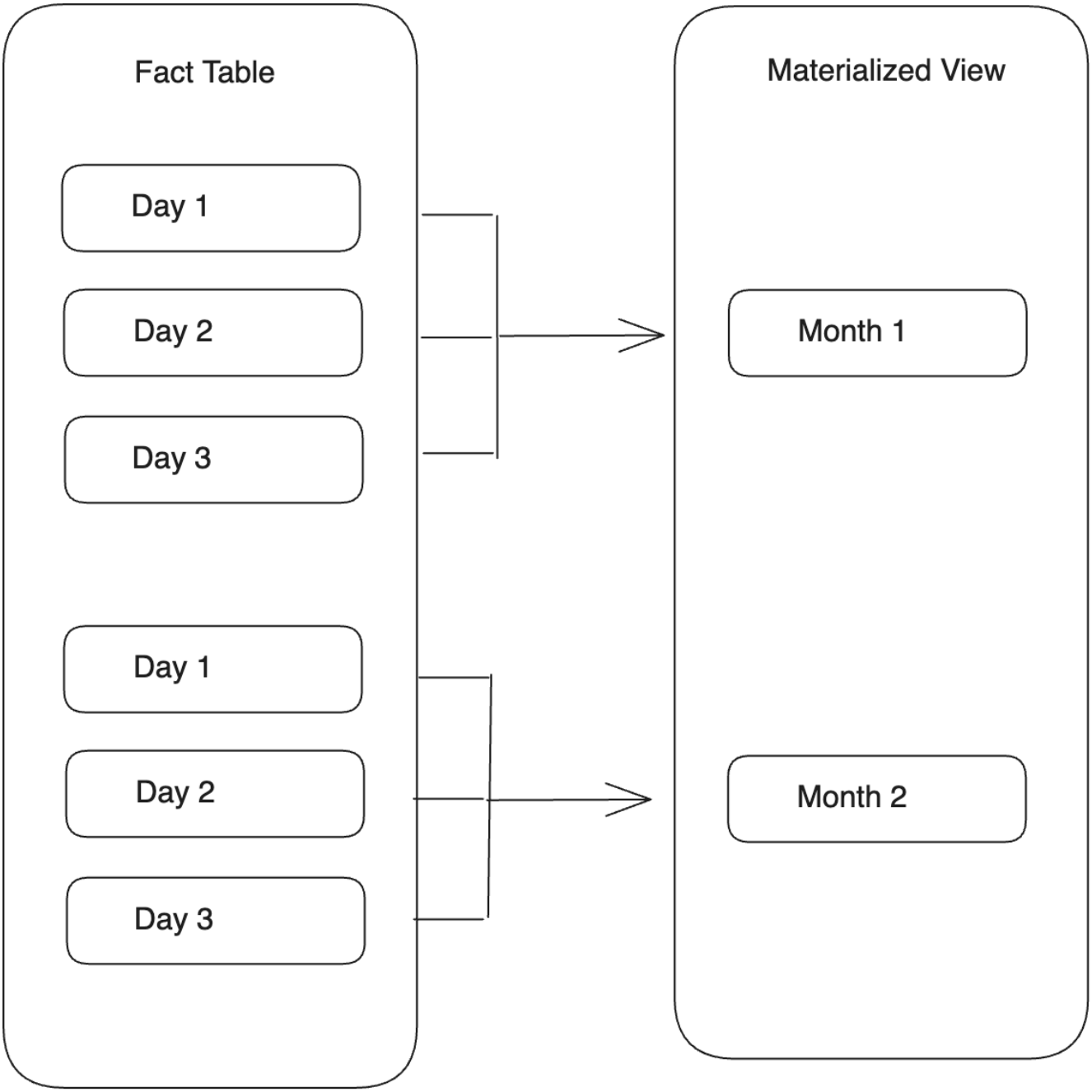
```
PARTITION BY str2date(<base_table_partitioning_column>, <format>)
```

Пример:

```
CREATE MATERIALIZED VIEW par_mv2
REFRESH ASYNC
PARTITION BY str2date(datekey, '%Y-%m-%d')
AS
SELECT
  k1,
  sum(v1) AS SUM,
  datekey
FROM par_tbl2
GROUP BY datekey, k1;
```

Выравнивание partition с сворачиванием временной детализации

Вы можете создать материализованное представление, детализация partitioning которого больше, чем у базовой таблицы, используя функцию date_trunc на Partitioning Key. Когда обнаруживаются изменения данных в partition базовой таблицы, Selena обновляет соответствующие свёрнутые partition в материализованном представлении.



- Если Partitioning Key базовой таблицы имеет тип DATE или DATETIME, вы можете напрямую использовать функцию date_trunc на Partitioning Key базовой таблицы.

```
PARTITION BY date_trunc(<format>, <base_table_partitioning_column>)
```

Пример:

```
CREATE MATERIALIZED VIEW par_mv3
REFRESH ASYNC
PARTITION BY date_trunc('month', datekey)
AS
SELECT
  k1,
  sum(v1) AS SUM,
  datekey
FROM par_tbl1
GROUP BY datekey, k1;
```

- Если Partitioning Key базовой таблицы имеет тип STRING, вы должны преобразовать Partitioning Key базовой таблицы в тип DATE или DATETIME в списке SELECT, установить для него псевдоним и использовать его в функции date_trunc для указания Partitioning Key материализованного представления.

```
PARTITION BY
date_trunc(<format>, <mv_partitioning_column>)
AS
SELECT
  str2date(<base_table_partitioning_column>, <format>) AS
<mv_partitioning_column>
```

Пример:

```
CREATE MATERIALIZED VIEW par_mv4
REFRESH ASYNC
PARTITION BY date_trunc('month', mv_datekey)
AS
SELECT
  datekey,
  k1,
  sum(v1) AS SUM,
  str2date(datekey, '%Y-%m-%d') AS mv_datekey
FROM par_tbl2
GROUP BY datekey, k1;
```

Выравнивание partition с настраиваемой временной детализацией

Метод сворачивания partition, упомянутый выше, позволяет разбивать материализованное представление только на основе конкретных временных детализаций и не позволяет настраивать диапазон времени partition. Если вашему бизнес-сценарию требуется partitioning с использованием настраиваемой временной детализации, вы можете создать материализованное представление и определить временную детализацию для его partition, используя функцию date_trunc с функцией time_slice, которая может преобразовать заданное время в начало или конец временного интервала на основе указанной временной детализации.

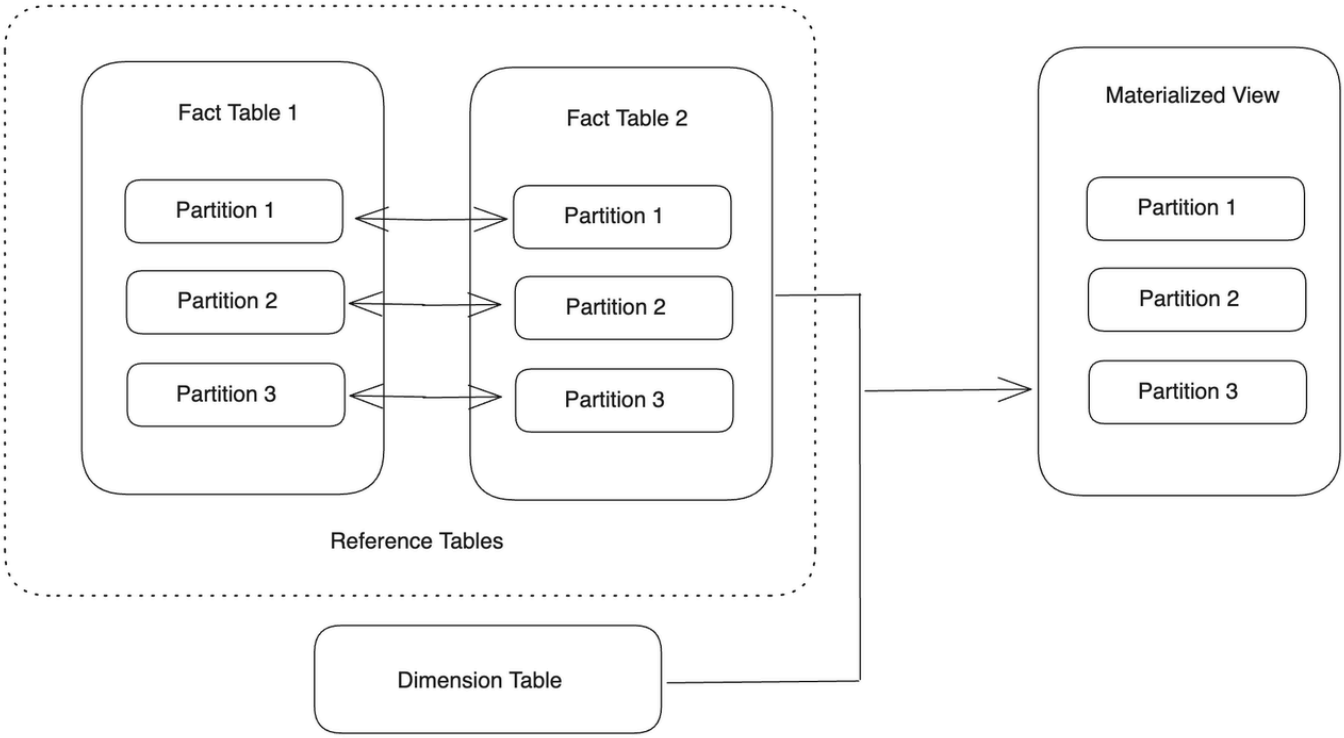
Вам необходимо определить новую временную детализацию (интервал), используя функцию time_slice на Partitioning Key базовой таблицы в списке SELECT, установить для него псевдоним и использовать его в функции date_trunc для указания Partitioning Key материализованного представления.

```
PARTITION BY
date_trunc(<format>, <mv_partitioning_column>)
AS
SELECT
  -- Вы можете использовать time_slice.
  time_slice(<base_table_partitioning_column>, <interval>) AS
<mv_partitioning_column>
```

Пример:

```
CREATE MATERIALIZED VIEW par_mv5
REFRESH ASYNC
PARTITION BY date_trunc('day', mv_datekey)
AS
SELECT
  k1,
  sum(v1) AS SUM,
  time_slice(datekey, INTERVAL 5 MINUTE) AS mv_datekey
FROM par_tbl1
GROUP BY datekey, k1;
```

Выравнивание partition с несколькими базовыми таблицами



Вы можете создать материализованное представление, partition которого выровнены с partition нескольких базовых таблиц, если partition базовых таблиц могут выравниваться друг с другом, то есть базовые таблицы используют один и тот же тип Partitioning Key. Вы можете использовать JOIN для соединения базовых таблиц и установить Partition Key в качестве общего столбца. Альтернативно, вы можете соединить их с помощью UNION. Базовые таблицы с выровненными partition называются справочными таблицами. Изменения данных в любой из справочных таблиц запустят задачу обновления соответствующих partition материализованного представления.

Эта функция поддерживается начиная с v1.5.2.

```sql
-- Соединение таблиц с помощью JOIN.
CREATE MATERIALIZED VIEW par_mv6
REFRESH ASYNC
PARTITION BY datekey
AS SELECT
  par_tbl1.datekey,
  par_tbl1.k1 AS t1k1,
  par_tbl3.k1 AS t2k1,
  sum(par_tbl1.v1) AS SUM1,
  sum(par_tbl3.v1) AS SUM2
FROM par_tbl1 JOIN par_tbl3 ON par_tbl1.datekey = par_tbl3.datekey_new
GROUP BY par_tbl1.datekey, t1k1, t2k1;

-- Соединение таблиц с помощью UNION.
CREATE MATERIALIZED VIEW par_mv7
REFRESH ASYNC
PARTITION BY datekey
AS SELECT
  par_tbl1.datekey,
  par_tbl1.k1 AS t1k1,
  sum(par_tbl1.v1) AS SUM1
FROM par_tbl1
GROUP BY
  par_tbl1.datekey,
  par_tbl1.k1
UNION ALL
SELECT
  par_tbl3.datekey_new,
  par_tbl3.k1 AS t2k1,
  sum(par_tbl3.v1) AS SUM2
FROM par_tbl3
GROUP BY
  par_tbl3.datekey_new,
  par_tbl3.k1;
```

Выравнивание нескольких столбцов partition

Начиная с v1.5.2, асинхронные материализованные представления поддерживают многостолбцовые выражения partition. Вы можете указать несколько столбцов partition для материализованного представления и отобразить их один к одному на столбцы partition базовых таблиц. Это обеспечивает интеграцию Lakehouse и улучшает повторное использование существующих возможностей внутренних таблиц Selena, предоставляя лучшее решение Lakehouse:

- **Упрощенный пользовательский опыт**: Упрощает процесс переноса данных из Data Lake в Selena.
- **Ускоренная производительность запросов**: Повторно использует возможности нативных таблиц, такие как Colocate Group, Bitmap Index, Bloom Filter Index, Sort Key и глобальные словари при создании материализованных представлений. Запросы, напрямую ссылающиеся на базовую таблицу, будут получать преимущества от прозрачного ускорения запросов через автоматическую перезапись.

Примечания к многостолбцовым выражениям partition

- В настоящее время многостолбцовые partition в материализованных представлениях могут быть отображены только один к одному или в отношении N:1 с partition базовой таблицы, а не в отношении M:N. Например, если базовая таблица имеет столбцы partition `(col1, col2, ..., coln)`, выражение partition материализованного представления может быть только одностолбцовым partition, таким как `col1`, `col2` или `coln`, или отображением один к одному со столбцами partition базовой таблицы, то есть `(col1, col2, ..., coln)`. Это ограничение разработано для упрощения логики отображения partition между базовой таблицей и материализованным представлением, избегая сложности, вносимой отношениями M:N.

- Поскольку выражения partition Iceberg поддерживают функцию `transform`, требуется дополнительная обработка при отображении выражений partition Iceberg на выражения partition материализованного представления Selena. Отношение отображения следующее:

Iceberg Transform	Выражение partition Iceberg	Выражение partition материализованного представления
Identity	`<col>`	`<col>`
hour	`hour(<col>)`	`date_trunc('hour', <col>)`
day	`day(<col>)`	`date_trunc('day', <col>)`
month	`month(<col>)`	`date_trunc('month', <col>)`
year	`year(<col>)`	`date_trunc('year', <col>)`
bucket	`bucket(<col>, <n>)`	Не поддерживается
truncate	`truncate(<col>)`	Не поддерживается

- Для столбцов partition не-Iceberg, где вычисление выражения partition не требуется, дополнительная обработка выражения partition не требуется. Вы можете отобразить их напрямую.

Следующий пример создает разбитое материализованное представление с многостолбцовым выражением partition на основе базовой таблицы из Iceberg Catalog (Spark).

Определение базовой таблицы в Spark:

```
-- Выражение partition базовой таблицы содержит несколько столбцов и
преобразование `days`.
CREATE TABLE lineitem_days (
      l_orderkey    BIGINT,
      l_partkey     INT,
      l_suppkey     INT,
      l_linenumber  INT,
      l_quantity    DECIMAL(15, 2),
      l_extendedprice  DECIMAL(15, 2),
      l_discount    DECIMAL(15, 2),
      l_tax         DECIMAL(15, 2),
      l_returnflag  VARCHAR(1),
      l_linestatus  VARCHAR(1),
      l_shipdate    TIMESTAMP,
      l_commitdate  TIMESTAMP,
      l_receiptdate TIMESTAMP,
      l_shipinstruct VARCHAR(25),
      l_shipmode     VARCHAR(10),
      l_comment      VARCHAR(44)
) USING ICEBERG
PARTITIONED BY (l_returnflag, l_linestatus, days(l_shipdate));
```

Создайте материализованное представление со столбцами partition, отображенными один к одному со столбцами базовой таблицы:

```
CREATE MATERIALIZED VIEW test_days
PARTITION BY (l_returnflag, l_linestatus, date_trunc('day', l_shipdate))
REFRESH DEFERRED MANUAL
AS
SELECT * FROM iceberg_catalog.test_db.lineitem_days;
```

Достижение инкрементного обновления и прозрачной перезаписи

Вы можете создать разбитое материализованное представление, которое обновляется по partition для достижения инкрементных обновлений материализованного представления и

прозрачной перезаписи запросов с частичной материализацией данных.

Для достижения этих целей вы должны учитывать следующие аспекты при создании материализованного представления:

- **Детализация обновления**

  Вы можете использовать свойство `partition_refresh_number` для указания детализации каждой операции обновления. `partition_refresh_number` контролирует максимальное количество partition, которые необходимо обновить в задаче обновления, когда запускается обновление. Если количество partition, которые необходимо обновить, превышает это значение, Selena разделит задачу обновления и выполнит её партиями. Partition обновляются в хронологическом порядке от наименее недавней partition к наиболее недавней partition (исключая partition, созданные динамически для будущего). Когда значение равно `-1`, задача обновления не будет разделена. Значение по умолчанию изменено с `-1` на `1` начиная с v1.5.2, что означает, что Selena обновляет partition по одной.

  Вы также можете использовать свойство `partition_refresh_strategy` для контроля адаптивности стратегии обновления. Этот параметр поддерживает следующие два значения:

  - "strict" (по умолчанию): Обновление partition строго следует настройке partition_refresh_number для контроля детализации обновления.
  - "adaptive": Адаптивно регулирует количество partition, обновляемых за одну партию, на основе объема данных в partition базовой таблицы, повышая эффективность обновления.
- **Область материализации**

  Область материализованных данных контролируется свойствами `partition_ttl_number` (для версий ранее v1.5.2) или `partition_ttl` (рекомендуется для v1.5.2 и более поздних версий). `partition_ttl_number` указывает количество самых последних partition для сохранения, а `partition_ttl` указывает временной диапазон данных материализованного представления для сохранения. Во время каждого обновления Selena упорядочивает partition в хронологическом порядке и сохраняет только те, которые удовлетворяют требованиям TTL.

- **Стратегия обновления**

- Материализованные представления с автоматическими стратегиями обновления (`REFRESH ASYNC`) автоматически обновляются каждый раз при изменении данных базовой таблицы.
- Материализованные представления с регулярными стратегиями обновления (`REFRESH ASYNC [START (<start_time>)] EVERY (INTERVAL <interval>)`) обновляются регулярно с определенным интервалом.

> **ПРИМЕЧАНИЕ**
>
> Материализованные представления с автоматическими стратегиями обновления и регулярными стратегиями обновления обновляются автоматически после запуска задач обновления. Selena записывает и сравнивает версии данных каждой partition базовой таблицы. Изменение версии данных указывает на изменение данных в partition. Как только Selena обнаруживает изменение данных в partition базовой таблицы, она обновляет соответствующую partition материализованного представления. Когда изменения данных в partition базовой таблицы не обнаружены, обновление соответствующей partition материализованного представления пропускается.

- Материализованные представления с ручными стратегиями обновления (`REFRESH MANUAL`) могут быть обновлены только путем ручного выполнения оператора REFRESH MATERIALIZED VIEW. Вы можете указать временной диапазон partition, которые необходимо обновить, чтобы избежать обновления всего материализованного представления. Если вы укажете `FORCE` в операторе, Selena принудительно обновит соответствующее материализованное представление или partition независимо от того, изменились ли данные в базовой таблице. Добавив `WITH SYNC MODE` к оператору, вы можете сделать синхронный вызов задачи обновления, и Selena вернет результат задачи только тогда, когда задача завершится успешно или неудачно.

Следующий пример создает разбитое материализованное представление `par_mv8`. Если Selena обнаруживает изменения данных в partition базовой таблицы, она обновляет соответствующую partition в материализованном представлении. Задача обновления разделяется на партии, каждая из которых обновляет только одну partition (`"partition_refresh_number" = "1"`). Сохраняются только две самые последние partition (`"partition_ttl_number" = "2"`), остальные удаляются во время обновления.

```
CREATE MATERIALIZED VIEW par_mv8
REFRESH ASYNC
PARTITION BY datekey
PROPERTIES(
  "partition_ttl_number" = "2",
  "partition_refresh_number" = "1"
)
AS
SELECT
  k1,
  sum(v1) AS SUM,
  datekey
FROM par_tbl1
GROUP BY datekey, k1;
```

Вы можете использовать оператор REFRESH MATERIALIZED VIEW для обновления этого материализованного представления. Следующий пример выполняет синхронный вызов для принудительного обновления некоторых partition `par_mv8` в определенном временном диапазоне.

```
REFRESH MATERIALIZED VIEW par_mv8
PARTITION START ("2021-01-03") END ("2021-01-04")
FORCE WITH SYNC MODE;
```

Вывод:

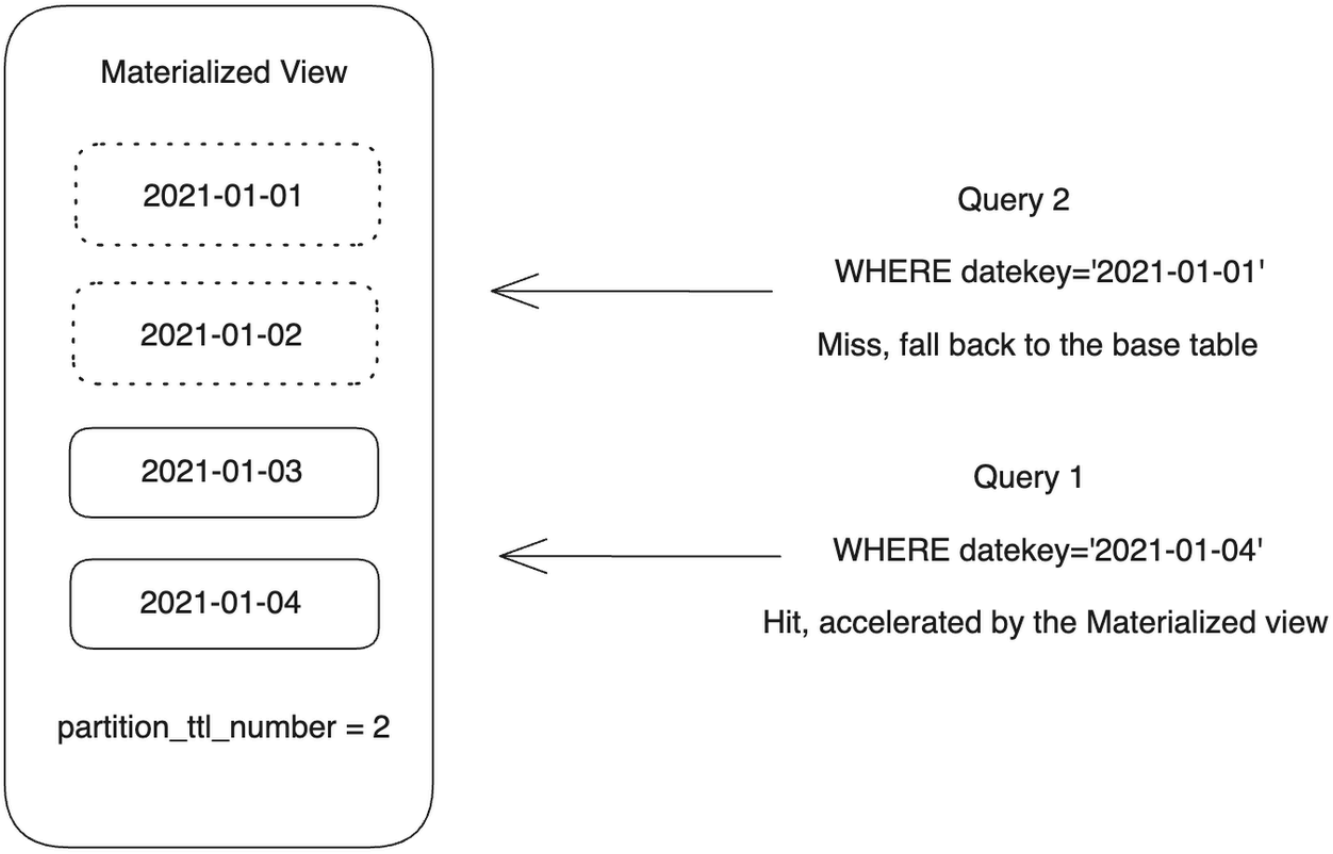
```
+--------------------------------------+
| QUERY_ID                             |
+--------------------------------------+
| 1d1c24b8-bf4b-11ee-a3cf-00163e0e23c9 |
+--------------------------------------+
1 row in set (1.12 sec)
```

С функцией TTL в `par_mv8` сохраняются только некоторые partition. Таким образом, вы достигли материализации частичных данных, что важно в сценариях, где большинство запросов направлены на недавние данные. Функция TTL позволяет прозрачно ускорять запросы на новые данные (например, в течение недели или месяца) с помощью материализованного представления, при этом значительно экономя затраты на хранение.

Запросы, которые не попадают в этот временной диапазон, направляются к базовой таблице.

В следующем примере запрос 1 будет ускорен материализованным представлением, потому что он попадает в partition, которая сохранена в `par_mv8`, в то время как запрос 2 будет направлен к базовой таблице, потому что он не попадает во временной диапазон, где сохранены partition.

```
-- Запрос 1
SELECT
  k1,
  sum(v1) AS SUM,
  datekey
FROM par_tbl1
WHERE datekey='2021-01-04'
GROUP BY datekey, k1;

-- Запрос 2
SELECT
  k1,
  sum(v1) AS SUM,
  datekey
FROM par_tbl1
WHERE datekey='2021-01-01'
GROUP BY datekey, k1;
```



6.6.4 Устранение неполадок асинхронных материализованных представлений

В этом разделе описано, как проверять асинхронные материализованные представления и решать проблемы, с которыми вы можете столкнуться при работе с ними.

> **ВНИМАНИЕ**
>
> Некоторые функции, показанные ниже, поддерживаются только начиная с Selena v1.5.2.

Проверка асинхронного материализованного представления

Чтобы получить полную картину асинхронных материализованных представлений, с которыми вы работаете, можно сначала проверить их рабочее состояние, историю обновлений и потребление ресурсов.

Проверка рабочего состояния асинхронного материализованного представления

Вы можете проверить рабочее состояние асинхронного материализованного представления с помощью SHOW MATERIALIZED VIEWS. Среди всей возвращаемой информации обратите внимание на следующие поля:

- `is_active`: Активно ли состояние материализованного представления. Только активное материализованное представление может использоваться для ускорения запросов и их перезаписи.
- `last_refresh_state`: Состояние последнего обновления, включая PENDING, RUNNING, FAILED и SUCCESS.
- `last_refresh_error_message`: Причина сбоя последнего обновления (если состояние материализованного представления неактивно).
- `rows`: Количество строк данных в материализованном представлении. Обратите внимание, что это значение может отличаться от фактического количества строк материализованного представления, поскольку обновления могут быть отложены.

Подробную информацию о других возвращаемых полях см. в SHOW MATERIALIZED VIEWS - Returns.

Пример:

```
MySQL > SHOW MATERIALIZED VIEWS LIKE 'mv_pred_2'\G
***************************[ 1. row ]***************************
id                                    | 112517
database_name                         | ssb_1g
name                                  | mv_pred_2
refresh_type                          | ASYNC
is_active                             | true
inactive_reason                       | <null>
partition_type                        | UNPARTITIONED
task_id                               | 457930
task_name                             | mv-112517
last_refresh_start_time               | 2023-08-04 16:46:50
last_refresh_finished_time            | 2023-08-04 16:46:54
last_refresh_duration                 | 3.996
last_refresh_state                    | SUCCESS
last_refresh_force_refresh            | false
last_refresh_start_partition          |
last_refresh_end_partition            |
last_refresh_base_refresh_partitions | {}
last_refresh_mv_refresh_partitions   |
last_refresh_error_code               | 0
last_refresh_error_message            |
rows                                  | 0
text                                  | CREATE MATERIALIZED VIEW `mv_pred_2`
(`lo_quantity`, `lo_revenue`, `sum`)
DISTRIBUTED BY HASH(`lo_quantity`, `lo_revenue`) BUCKETS 2
REFRESH ASYNC
PROPERTIES (
"replication_num" = "3",
"storage_medium" = "HDD"
)
AS SELECT `lineorder`.`lo_quantity`, `lineorder`.`lo_revenue`,
sum(`lineorder`.`lo_tax`) AS `sum`
FROM `ssb_1g`.`lineorder`
WHERE `lineorder`.`lo_linenumber` = 1
GROUP BY 1, 2;

1 row in set
Time: 0.003s
```

Просмотр истории обновлений асинхронного материализованного представления

Вы можете просмотреть историю обновлений асинхронного материализованного представления, запросив таблицу `task_runs` в базе данных `information_schema`. Среди всей возвращаемой информации обратите внимание на следующие поля:

- `CREATE_TIME` и `FINISH_TIME`: Время начала и окончания задачи обновления.
- `STATE`: Состояние задачи обновления, включая PENDING, RUNNING, FAILED и SUCCESS.
- `ERROR_MESSAGE`: Причина сбоя задачи обновления.

Пример:

```
MySQL > SELECT * FROM information_schema.task_runs WHERE task_name ='mv-
112517' \G
***************************[ 1. row ]***************************
QUERY_ID      | 7434cee5-32a3-11ee-b73a-8e20563011de
TASK_NAME     | mv-112517
CREATE_TIME   | 2023-08-04 16:46:50
FINISH_TIME   | 2023-08-04 16:46:54
STATE         | SUCCESS
DATABASE      | ssb_1g
EXPIRE_TIME   | 2023-08-05 16:46:50
ERROR_CODE    | 0
ERROR_MESSAGE | <null>
PROGRESS      | 100%
EXTRA_MESSAGE | {"forceRefresh":false,"mvPartitionsToRefresh":
[],"refBasePartitionsToRefreshMap":{},"basePartitionsToRefreshMap":{}}
PROPERTIES    | {"FORCE":"false"}
***************************[ 2. row ]***************************
QUERY_ID      | 72dd2f16-32a3-11ee-b73a-8e20563011de
TASK_NAME     | mv-112517
CREATE_TIME   | 2023-08-04 16:46:48
FINISH_TIME   | 2023-08-04 16:46:53
STATE         | SUCCESS
DATABASE      | ssb_1g
EXPIRE_TIME   | 2023-08-05 16:46:48
ERROR_CODE    | 0
ERROR_MESSAGE | <null>
PROGRESS      | 100%
EXTRA_MESSAGE | {"forceRefresh":true,"mvPartitionsToRefresh":
["mv_pred_2"],"refBasePartitionsToRefreshMap":
{},"basePartitionsToRefreshMap":{"lineorder":["lineorder"]}}
PROPERTIES    | {"FORCE":"true"}
```

Мониторинг потребления ресурсов асинхронным материализованным представлением

Вы можете отслеживать и анализировать ресурсы, потребляемые асинхронным материализованным представлением во время и после обновления.

Мониторинг потребления ресурсов во время обновления

Во время выполнения задачи обновления вы можете отслеживать его потребление ресурсов в реальном времени с помощью SHOW PROC '/current_queries'.

Среди всей возвращаемой информации обратите внимание на следующие поля:

- `ScanBytes`: Размер отсканированных данных.
- `ScanRows`: Количество отсканированных строк данных.
- `MemoryUsage`: Объем используемой памяти.
- `CPUTime`: Затраченное время CPU.
- `ExecTime`: Время выполнения запроса.

Пример:

```
MySQL > SHOW PROC '/current_queries'\G
***************************[ 1. row ]***************************
StartTime     | 2023-08-04 17:01:30
QueryId       | 806eed7d-32a5-11ee-b73a-8e20563011de
ConnectionId  | 0
Database      | ssb_1g
User          | root
ScanBytes     | 70.981 MB
ScanRows      | 6001215 rows
MemoryUsage   | 73.748 MB
DiskSpillSize | 0.000
CPUTime       | 2.515 s
ExecTime      | 2.583 s
```

Анализ потребления ресурсов после обновления

После выполнения задачи обновления вы можете проанализировать потребление ресурсов через профили запросов. Вы можете просмотреть профиль задачи обновления материализованного представления через веб-интерфейс узла Leader FE кластера.

Когда асинхронное материализованное представление обновляется, выполняется оператор INSERT OVERWRITE. Вы можете проверить соответствующий профиль запроса, чтобы проанализировать время и ресурсы, затраченные задачей обновления.

Среди всей возвращаемой информации обратите внимание на следующие метрики:

- `Total`: Общее время, затраченное запросом.
- `QueryCpuCost`: Общее время CPU, затраченное запросом. Время CPU агрегируется для параллельных процессов. В результате значение этой метрики может быть больше фактического времени выполнения запроса.
- `QueryMemCost`: Общая стоимость памяти запроса.
- Другие метрики для отдельных операторов, таких как операторы join и агрегирования.

Подробную информацию о том, как проверить профиль запроса и понять другие метрики, см. в Analyze query profile.

Проверка, перезаписываются ли запросы асинхронным материализованным представлением

Вы можете проверить, может ли запрос быть перезаписан с помощью асинхронного материализованного представления из его плана запроса, используя EXPLAIN.

Если метрика `SCAN` в плане запроса показывает имя соответствующего материализованного представления, запрос был перезаписан материализованным представлением.

Пример 1:

```
MySQL > SHOW CREATE TABLE mv_agg\G
***************************[ 1. row ]***************************
Materialized View        | mv_agg
Create Materialized View | CREATE MATERIALIZED VIEW `mv_agg` (`c_custkey`)
DISTRIBUTED BY RANDOM
REFRESH ASYNC
PROPERTIES (
"replication_num" = "3",
"replicated_storage" = "true",
"storage_medium" = "HDD"
)
AS SELECT `customer`.`c_custkey`
FROM `ssb_1g`.`customer`
GROUP BY `customer`.`c_custkey`;

MySQL > EXPLAIN LOGICAL SELECT `customer`.`c_custkey`
                      -> FROM `ssb_1g`.`customer`
                      -> GROUP BY `customer`.`c_custkey`;
+-----------------------------------------------------------------------------
---------+
| Explain String
|
+-----------------------------------------------------------------------------
---------+
| - Output => [1:c_custkey]
|
|     - SCAN [mv_agg] => [1:c_custkey]
|
|             Estimates: {row: 30000, cpu: ?, memory: ?, network: ?, cost:
15000.0} |
|             partitionRatio: 1/1, tabletRatio: 12/12
|
|             1:c_custkey := 10:c_custkey
|
+-----------------------------------------------------------------------------
---------+
```

Если вы отключите функцию перезаписи запросов, Selena примет обычный план запроса.

Пример 2:

```
MySQL > SET enable_materialized_view_rewrite = false;
MySQL > EXPLAIN LOGICAL SELECT `customer`.`c_custkey`
                      -> FROM `ssb_1g`.`customer`
                      -> GROUP BY `customer`.`c_custkey`;
+-----------------------------------------------------------------------
-------------+
| Explain String
|
+-----------------------------------------------------------------------
-------------+
| - Output => [1:c_custkey]
|
|     - AGGREGATE(GLOBAL) [1:c_custkey]
|
|             Estimates: {row: 15000, cpu: ?, memory: ?, network: ?, cost:
120000.0}    |
|         - SCAN [mv_bitmap] => [1:c_custkey]
|
|                 Estimates: {row: 60000, cpu: ?, memory: ?, network: ?,
cost: 30000.0} |
|                 partitionRatio: 1/1, tabletRatio: 12/12
|
+-----------------------------------------------------------------------
-------------+
```

Диагностика и решение проблем

Здесь мы перечислим некоторые распространенные проблемы, с которыми вы можете столкнуться при работе с асинхронным материализованным представлением, и соответствующие решения.

Сбой создания материализованного представления

Если вы не смогли создать асинхронное материализованное представление, то есть оператор CREATE MATERIALIZED VIEW не может быть выполнен, рассмотрите следующие аспекты:

- **Проверьте, не использовали ли вы по ошибке оператор SQL для синхронных материализованных представлений.**

  Selena предоставляет два различных материализованных представления: синхронное материализованное представление и асинхронное материализованное

представление.

Базовый оператор SQL, используемый для создания синхронного материализованного представления, выглядит следующим образом:

```
CREATE MATERIALIZED VIEW <mv_name>
AS <query>
```

Однако оператор SQL, используемый для создания асинхронного материализованного представления, содержит больше параметров:

```
CREATE MATERIALIZED VIEW <mv_name>
REFRESH ASYNC -- Стратегия обновления асинхронного материализованного
представления.
DISTRIBUTED BY HASH(<column>) -- Стратегия распределения данных
асинхронного материализованного представления.
AS <query>
```

Помимо оператора SQL, основное различие между двумя материализованными представлениями заключается в том, что асинхронные материализованные представления поддерживают весь синтаксис запросов, предоставляемый Selena, но синхронные материализованные представления поддерживают только ограниченный выбор агрегатных функций.

- **Проверьте, указали ли вы правильный столбец partitioning.**

  При создании асинхронного материализованного представления вы можете указать стратегию partitioning, которая позволяет обновлять материализованное представление на более детальном уровне.

  В настоящее время Selena поддерживает только range partitioning и поддерживает только ссылку на один столбец из выражения SELECT в операторе запроса, который используется для создания материализованного представления. Вы можете использовать функцию date_trunc() для усечения столбца, чтобы изменить уровень детализации стратегии partitioning. Обратите внимание, что любые другие выражения не поддерживаются.

- **Проверьте, есть ли у вас необходимые привилегии для создания материализованного представления.**

  При создании асинхронного материализованного представления вам нужны привилегии SELECT для всех объектов (таблиц, представлений, материализованных представлений), которые запрашиваются. Когда в запросе используются UDF, вам также необходимы привилегии USAGE для функций.

Сбой обновления материализованного представления

Если материализованное представление не удается обновить, то есть состояние задачи обновления не SUCCESS, рассмотрите следующие аспекты:

- **Проверьте, не приняли ли вы неподходящую стратегию обновления.**

  По умолчанию материализованное представление обновляется сразу после создания. Однако в версиях v1.5.2 и более ранних материализованные представления, использующие стратегию обновления MANUAL, не обновляются после создания. Вы должны обновить его вручную с помощью REFRESH MATERIALIZED VIEW.

- **Проверьте, не превышает ли задача обновления лимит памяти.**

  Обычно это происходит, когда асинхронное материализованное представление включает крупномасштабную агрегацию или join-вычисления, которые исчерпывают ресурсы памяти. Чтобы решить эту проблему, вы можете:

  - Указать стратегию partitioning для материализованного представления для обновления одной partition за раз.
  - Включить функцию Spill to Disk для задачи обновления. Начиная с v1.5.2, Selena поддерживает сброс промежуточных результатов на диск при обновлении материализованного представления. Выполните следующий оператор, чтобы включить Spill to Disk:

```
-- Определите свойства при создании материализованного представления.
CREATE MATERIALIZED VIEW mv1
REFRESH ASYNC
PROPERTIES ( 'session.enable_spill'='true' )
AS <query>;

-- Добавьте свойства к существующему материализованному представлению.
ALTER MATERIALIZED VIEW mv2 SET ('session.enable_spill' = 'true');
```

Тайм-аут обновления материализованного представления

Большие материализованные представления могут не обновиться из-за превышения времени ожидания задачей обновления. Рассмотрите следующие решения для решения этой проблемы:

- **Укажите стратегию partitioning для материализованного представления для достижения детализированного обновления**

  Как описано в Create partitioned materialized views, разбивая материализованное представление на partition, вы можете достичь инкрементного создания и обновления, тем самым избегая проблемы чрезмерного потребления ресурсов во время первоначального обновления.

- **Установите более длительное время ожидания**

  Время ожидания по умолчанию для задач обновления материализованных представлений составляет 5 минут в версиях ранее v1.5.2 и 1 час в v1.5.2 и более поздних. Если вы сталкиваетесь с исключениями тайм-аута, вы можете настроить период тайм-аута, используя следующий оператор:

  ```
  ALTER MATERIALIZED VIEW mv2 SET ('session.insert_timeout' = '4000');
  ```

- **Проанализируйте узкие места производительности обновления материализованного представления**

  Обновление материализованных представлений со сложными вычислениями занимает много времени. Вы можете проанализировать узкие места производительности, проанализировав профиль запроса задачи обновления:

- Получите `query_id`, соответствующий задаче обновления, запросив `information_schema.task_runs`.
- Проанализируйте профиль запроса задачи обновления, используя следующие операторы:
  - GET_QUERY_PROFILE: Получите исходный профиль запроса на основе `query_id`.
  - ANALYZE PROFILE: Проанализируйте профиль запроса на основе фрагментов и отобразите его в виде древовидной структуры.

Состояние материализованного представления неактивно

Если материализованное представление не может перезаписывать запросы или обновляться, и состояние материализованного представления `is_active` равно `false`, это может быть результатом изменения схемы базовых таблиц. Чтобы решить эту проблему, вы можете вручную установить состояние материализованного представления как активное, выполнив следующий оператор:

```
ALTER MATERIALIZED VIEW mv1 ACTIVE;
```

Если установка состояния материализованного представления как активного не работает, вам необходимо удалить материализованное представление и создать его заново.

Задача обновления материализованного представления использует чрезмерные ресурсы

Если вы обнаружите, что задачи обновления используют чрезмерные системные ресурсы, рассмотрите следующие аспекты:

- **Проверьте, не создали ли вы слишком большое материализованное представление.**

  Если вы объединили слишком много таблиц, что вызывает значительный объем вычислений, задача обновления займет много ресурсов. Чтобы решить эту проблему, вам необходимо оценить размер материализованного представления и перепланировать его.

- **Проверьте, не установили ли вы излишне частый интервал обновления.**

Если вы используете стратегию обновления с фиксированным интервалом, вы можете установить более низкую частоту обновления, чтобы решить проблему. Если задачи обновления запускаются изменениями данных в базовых таблицах, слишком частая загрузка данных также может вызвать эту проблему. Чтобы решить эту проблему, вам необходимо определить подходящую стратегию обновления для вашего материализованного представления.

- **Проверьте, разбито ли материализованное представление на partition.**

  Обновление неразбитого материализованного представления может быть дорогостоящим, потому что Selena обновляет все материализованное представление каждый раз. Чтобы решить эту проблему, вам необходимо указать стратегию partitioning для материализованного представления для обновления одной partition за раз.

Чтобы остановить задачу обновления, которая занимает слишком много ресурсов, вы можете:

- Установить состояние материализованного представления как неактивное, чтобы все его задачи обновления были остановлены:

```
ALTER MATERIALIZED VIEW mv1 INACTIVE;
```

- Прервать выполняющуюся задачу обновления с помощью CANCEL REFRESH MATERIALIZED VIEW:

```
CANCEL REFRESH MATERIALIZED VIEW mv1;
```

Сбой перезаписи запроса материализованным представлением

Если ваше материализованное представление не может перезаписать соответствующие запросы, рассмотрите следующие аспекты:

- **Диагностируйте сбой перезаписи с помощью TRACE**

  Selena предоставляет оператор TRACE для диагностики сбоя перезаписи материализованного представления:

- `TRACE LOGS MV <query>`: Доступно в v1.5.2 и более поздних версиях, эта команда анализирует подробный процесс перезаписи и причины сбоя.
- `TRACE REASON MV <query>`: Доступно в v1.5.2 и более поздних версиях, эта команда предоставляет краткие причины сбоя перезаписи.

```
MySQL > TRACE REASON MV SELECT sum(c1) FROM
`glue_ice`.`iceberg_test`.`ice_test3`;
+-----------------------------------------------------------------------
-----------------------------------------------+
| Explain String
|
+-----------------------------------------------------------------------
-----------------------------------------------+
|     MV rewrite fail for mv1: Rewrite aggregate rollup sum(1: c1)
failed: only column-ref is supported after rewrite  |
|     MV rewrite fail for mv1: Rewrite aggregate function failed,
cannot get rollup function: sum(1: c1)               |
|     MV rewrite fail for mv1: Rewrite rollup aggregate failed: cannot
rewrite aggregate functions                     |
+-----------------------------------------------------------------------
-----------------------------------------------+
```

- **Проверьте, совпадают ли материализованное представление и запрос.**
  - Selena сопоставляет материализованное представление и запрос с помощью техники сопоставления на основе структуры, а не текстового сопоставления. Поэтому не гарантируется, что запрос может быть перезаписан только потому, что он выглядит похожим на запрос материализованного представления.
  - Материализованные представления могут перезаписывать только запросы типа SPJG (Select/Projection/Join/Aggregation). Запросы, включающие оконные функции, вложенную агрегацию или join плюс агрегацию, не поддерживаются.
  - Материализованные представления не могут перезаписывать запросы, которые включают сложные предикаты Join в Outer Joins. Например, в случаях типа `A LEFT JOIN B ON A.dt > '2023-01-01' AND A.id = B.id` рекомендуется указать предикат из предложения `JOIN ON` в предложении `WHERE`.

  Для получения дополнительной информации об ограничениях перезаписи запросов материализованным представлением см. Query rewrite with materialized views - Limitations.

- **Проверьте, активно ли состояние материализованного представления.**

Selena проверяет статус материализованного представления перед перезаписью запросов. Запросы могут быть перезаписаны только тогда, когда состояние материализованного представления активно. Чтобы решить эту проблему, вы можете вручную установить состояние материализованного представления как активное, выполнив следующий оператор:

```
ALTER MATERIALIZED VIEW mv1 ACTIVE;
```

- **Проверьте, соответствует ли материализованное представление требованиям согласованности данных.**

  Selena проверяет согласованность данных в материализованном представлении и в данных базовой таблицы. По умолчанию запросы могут быть перезаписаны только тогда, когда данные в материализованном представлении актуальны. Чтобы решить эту проблему, вы можете:

  - Добавить `PROPERTIES('query_rewrite_consistency'='LOOSE')` к материализованному представлению, чтобы отключить проверки согласованности.
  - Добавить `PROPERTIES('mv_rewrite_staleness_second'='5')`, чтобы допустить определенную степень несогласованности данных. Запросы могут быть перезаписаны, если последнее обновление было до этого временного интервала, независимо от того, изменились ли данные в базовых таблицах.
- **Проверьте, не отсутствуют ли в операторе запроса материализованного представления выходные столбцы.**

  Для перезаписи запросов диапазона и точечных запросов вы должны указать столбцы, используемые в качестве предикатов фильтрации, в выражении SELECT оператора запроса материализованного представления. Вам необходимо проверить оператор SELECT материализованного представления, чтобы убедиться, что он включает столбцы, на которые ссылаются предложения WHERE и ORDER BY запроса.

Пример 1: Материализованное представление `mv1` использует вложенную агрегацию. Таким образом, оно не может быть использовано для перезаписи запросов.

```
CREATE MATERIALIZED VIEW mv1 REFRESH ASYNC AS
select count(distinct cnt)
from (
    select c_city, count(*) cnt
    from customer
    group by c_city
) t;
```

Пример 2: Материализованное представление `mv2` использует join плюс агрегацию. Таким образом, оно не может быть использовано для перезаписи запросов. Чтобы решить эту проблему, вы можете создать материализованное представление с агрегацией, а затем вложенное материализованное представление с join на основе предыдущего.

```
CREATE MATERIALIZED VIEW mv2 REFRESH ASYNC AS
select *
from (
    select lo_orderkey, lo_custkey, p_partkey, p_name
    from lineorder
    join part on lo_partkey = p_partkey
) lo
join (
    select c_custkey
    from customer
    group by c_custkey
) cust
on lo.lo_custkey = cust.c_custkey;
```

Пример 3: Материализованное представление `mv3` не может перезаписать запросы в шаблоне `SELECT c_city, sum(tax) FROM tbl WHERE dt='2023-01-01' AND c_city = 'xxx'`, потому что столбец, на который ссылается предикат, отсутствует в выражении SELECT.

```
CREATE MATERIALIZED VIEW mv3 REFRESH ASYNC AS
SELECT c_city, sum(tax) FROM tbl GROUP BY c_city;
```

Чтобы решить эту проблему, вы можете создать материализованное представление следующим образом:

```
CREATE MATERIALIZED VIEW mv3 REFRESH ASYNC AS
SELECT dt, c_city, sum(tax) FROM tbl GROUP BY dt, c_city;
```

Часто задаваемые вопросы

Какая группа ресурсов контролирует ресурсы для асинхронных материализованных представлений? Как можно настроить её конфигурацию?

Если свойство `resource_group` не указано при создании асинхронного материализованного представления, система назначает его группе ресурсов по умолчанию `default_mv_wg`. Её конфигурация по умолчанию:

- `cpu_core_limit`: 1
- `mem_limit`: 80%
- `concurrency_limit`: 0
- `spill_mem_limit_threshold`: 80%

Вы можете настроить лимит CPU, лимит памяти, лимит параллелизма и порог сброса через следующие элементы конфигурации BE:

- `default_mv_resource_group_cpu_limit`
- `default_mv_resource_group_memory_limit`
- `default_mv_resource_group_concurrency_limit`
- `default_mv_resource_group_spill_mem_limit_threshold`

Вы также можете назначить выделенную группу ресурсов материализованному представлению, указав свойство `resource_group`.

Пример:

```
CREATE MATERIALIZED VIEW IF NOT EXISTS mv_test.test_1
PARTITION BY `metric_date`
REFRESH MANUAL
PROPERTIES (
  "replicated_storage" = "true",
  "partition_refresh_number" = "1",
  "force_external_table_query_rewrite" = "CHECKED",
  "query_rewrite_consistency" = "LOOSE",
  "replication_num" = "1",
  "storage_medium" = "HDD",
  # highlight-start
  "resource_group" = "rg_mv"
  # highlight-end
) AS ...
```

Если асинхронное материализованное представление запланировано обновляться каждую 1 минуту, что произойдет, если обновление займет больше 1 минуты?

Система обрабатывает это автоматически:

- Если в очереди есть ожидающие задачи, новая запущенная задача объединяется (статус `MERGED`).
- Если нет ожидающих задач (только выполняющиеся или успешно завершенные/неудачные задачи), новая запущенная задача ждет, пока текущая выполняющаяся задача не завершится, прежде чем выполняться.

Ограничение: одновременно может выполняться только одна задача обновления на материализованное представление.

После установки auto_refresh_partitions_limit, будет ли система всегда обновлять именно это количество partition?

Нет. Этот параметр представляет верхний лимит. Если система обнаруживает, что изменилось меньше partition базовой таблицы, будут обновлены только эти фактические partition.

Материализованное представление настроено на обновление каждые 5 минут, но в `information_schema.task_runs` оно, похоже, обновляется каждые 5 секунд. Почему?

Начиная с v1.5.2, значение по умолчанию `partition_refresh_number` изменилось с -1 на 1, что означает, что каждая операция обновления обрабатывает только одну partition за раз. Если необходимо обновить много partition, система разделяет работу на несколько

подзадач, что приводит к более частым записям планирования в `task_runs`, чем определенный пользователем интервал обновления.

Можно ли вкладывать несколько асинхронных материализованных представлений? Есть ли рекомендации?

Вложение поддерживается, но рекомендуется минимизировать глубину вложения. Чрезмерное вложение значительно увеличивает сложность устранения неполадок.

6.6.5 Поддержка функций: Асинхронные материализованные представления

Асинхронные материализованные представления поддерживаются начиная с Selena v1.5.2. Асинхронные материализованные представления предназначены для ускорения сложных запросов с соединениями или агрегациями к большим таблицам в Selena или в озерах данных. Разница в производительности может быть значительной, когда запрос выполняется часто или достаточно сложен. Кроме того, асинхронные материализованные представления особенно полезны для построения математических моделей на основе вашего хранилища данных.

Этот документ определяет границы компетенции для асинхронных материализованных представлений и поддерживаемую версию задействованных функций.

Функции DDL

Функция	Описание	Поддерживаемые версии
Auto Analyze	Автоматически собирает статистику после создания материализованного представления, чтобы избежать сбоев перезаписи.	v1.5.2+
Random Bucketing	По умолчанию включает стратегию случайного баскетинга для материализованных представлений.	v1.5.2+
Deferred Refresh	Поддерживает указание того, обновлять ли материализованное представление сразу после создания, используя DEFERRED или IMMEDIATE в CREATE MATERIALIZED VIEW.	v1.5.2+
Order By	Поддерживает указание ключа сортировки для материализованных представлений с помощью ORDER BY.	v1.5.2+
Window/CTE/Union/Subquery	Поддерживает использование оконных функций, CTE, Union и подзапросов в материализованных представлениях.	v1.5.2+
ALTER ACTIVE	Активирует недействительные материализованные представления после изменений схемы в базовых таблицах, используя ключевое слово ACTIVE в	v1.5.2+

Функция	Описание	Поддерживаемые версии
	ALTER MATERIALIZED VIEW.	
REFRESH SYNC MODE	Поддерживает синхронное выполнение задач обновления материализованного представления, используя ключевые слова WITH SYNC MODE в REFRESH MATERIALIZED VIEW.	v1.5.2+
Intermediate Result Spilling	Поддерживает включение Intermediate Result Spilling с помощью свойства `enable_spill`, чтобы избежать OOM во время построения материализованного представления.	v1.5.2+
Resource Group	Поддерживает указание групп ресурсов с помощью свойства `resource_group` для построения материализованного представления для достижения изоляции ресурсов.	v1.5.2+
Materialized View on View	Поддерживает создание материализованных представлений на основе логических представлений.	v1.5.2+
Swap Materialized View	Поддерживает атомарную замену материализованного представления с помощью ключевых слов SWAP WITH в ALTER MATERIALIZED VIEW.	v1.5.2+
CREATE INDEX ON Materialized View	Поддерживает создание индексов на материализованных представлениях для ускорения точечных запросов.	v1.5.2+
AUTO ACTIVE	Автоматически активирует недействительные материализованные представления в фоновом режиме с экспоненциальной задержкой, останавливаясь после достижения интервала 60 минут.	v1.5.2+
Backup and Restore	Поддерживает резервное копирование и восстановление материализованных представлений.	v1.5.2+
Object Dependencies	Предоставляет системное представление `sys.object_dependencies` для уточнения зависимости между материализованными представлениями и базовыми таблицами.	v1.5.2+

Переменные

Переменная	Описание	По умолчанию	Поддерживаемые версии
enable_materialized_view_rewrite	Включать ли перезапись запросов материализованного представления.	true	v1.5.2+
enable_materialized_view_for_insert	Включать ли перезапись запросов материализованного представления для инструкций INSERT.	false	v1.5.2+
materialized_view_rewrite_mode	Режим перезаписи запросов материализованного представления.	DEFAULT	v1.5.2+

Переменная	Описание	По умолчанию	Поддерживаемые версии
optimizer_materialized_view_timelimit	Максимальное время, которое может быть использовано для перезаписи запросов материализованного представления, после чего перезапись запросов отменяется, и процесс оптимизатора продолжается.	1000	v1.5.2+
analyze_mv	Метод сбора статистики после обновления материализованного представления.	SAMPLE	v1.5.2+
enable_materialized_view_plan_cache	Включать ли кэш планов для материализованных представлений. По умолчанию кэшируется 1000 планов материализованных представлений.	TRUE	v1.5.2+
query_including_mv_names	Белый список материализованных представлений, которые могут быть использованы для перезаписи запросов.		v1.5.2+
query_excluding_mv_names	Черный список материализованных представлений, которые могут быть использованы для перезаписи запросов.		v1.5.2+
cbo_materialized_view_rewrite_related_mvs_limit	Максимальное количество кандидатов материализованных представлений на этапе планирования.	64	v1.5.2+

Свойства

Свойство	Описание	Поддерживаемые версии
`session.<property_name>`	Префикс переменных сеанса, используемых для построения материализованного представления, например, `session.insert_timeout` и `session.query_mem_limit`.	v1.5.2+
auto_refresh_partitions_limit	Максимальное количество партиций материализованного представления, которые должны быть обновлены каждый раз при запуске автоматического обновления.	v1.5.2+
excluded_trigger_tables	Базовые таблицы, обновления которых не будут запускать автоматическое обновление материализованного представления.	v1.5.2+
partition_refresh_number	Количество партиций, которые должны быть обновлены в каждом пакете, когда задача обновления выполняется пакетами.	v1.5.2+
partition_ttl_number	Количество последних партиций материализованного представления для сохранения.	v1.5.2+
partition_ttl	Время жизни (TTL) для партиций материализованного представления. Это свойство рекомендуется вместо `partition_ttl_number`.	v1.5.2+

Свойство	Описание	Поддерживаемые версии
force_external_table_query_rewrite	Включать ли перезапись запросов для материализованных представлений на основе внешнего каталога.	v1.5.2+
query_rewrite_consistency	Правило перезаписи запросов для материализованных представлений, построенных на внутренних таблицах.	v1.5.2+
resource_group	Группа ресурсов, к которой относятся задачи обновления материализованного представления.	v1.5.2+
colocate_with	Группа colocation материализованного представления.	v1.5.2+
foreign_key_constraints	Ограничения Foreign Key при создании материализованного представления для перезаписи запросов в сценарии View Delta Join.	v1.5.2+
unique_constraints	Ограничения Unique Key при создании материализованного представления для перезаписи запросов в сценарии View Delta Join.	v1.5.2+
mv_rewrite_staleness_second	Допуск устаревания данных материализованного представления во время перезаписи запросов.	v1.5.2+
enable_query_rewrite	Может ли материализованное представление использоваться для перезаписи запросов.	v1.5.2+
excluded_refresh_tables	Базовые таблицы, которые не запускают синхронизацию данных во время обновления материализованного представления.	v1.5.2+

Партиционирование

Выравнивание	Вариант использования	Поддерживаемые версии
Выравнивание партиций один-к-одному (типы Date)	Создание материализованного представления, партиции которого соответствуют партициям базовой таблицы один-к-одному, используя один и тот же Partitioning Key. Partitioning Key должен быть типа DATE или DATETIME.	v1.5.2+
Выравнивание партиций один-к-одному (тип STRING)	Создание материализованного представления, партиции которого соответствуют партициям базовой таблицы один-к-одному, используя один и тот же Partitioning Key. Partitioning Key должен быть типа STRING.	v1.5.2+
Выравнивание партиций с временной детализацией rollup (типы Date)	Создание материализованного представления, детализация партиционирования которого больше, чем у базовой таблицы, используя функцию `date_trunc` на Partitioning Key. Partitioning Key должен быть типа DATE или DATETIME.	v1.5.2+

Выравнивание	Вариант использования	Поддерживаемые версии
Выравнивание партиций с временной детализацией rollup (тип STRING)	Создание материализованного представления, детализация партиционирования которого больше, чем у базовой таблицы, используя функцию `date_trunc` на Partitioning Key. Partitioning Key должен быть типа STRING.	v1.5.2+
Выравнивание партиций с пользовательской временной детализацией	Создание материализованного представления и настройка временной детализации для его партиций, используя функцию `date_trunc` с функцией `time_slice` или `date_slice`.	v1.5.2+
Выравнивание партиций с несколькими базовыми таблицами	Создание материализованного представления, партиции которого выровнены с партициями нескольких базовых таблиц, при условии, что базовые таблицы используют один и тот же тип Partitioning Key.	v1.5.2+
Выравнивание нескольких столбцов партиций	Указание нескольких столбцов партиций для материализованного представления и их сопоставление один-к-одному со столбцами партиций базовых таблиц.	v1.5.2+

Различные методы соединения

- **Одна таблица фактов (v1.5.2+)**: Установление отображения партиций между материализованным представлением и таблицей фактов обеспечивает автоматическое обновление партиций материализованного представления при обновлении таблицы фактов.
- **Несколько таблиц фактов (v1.5.2+)**: Установление отображения партиций между материализованным представлением и несколькими таблицами фактов, которые соединяются/объединяются с одинаковой временной детализацией, обеспечивает автоматическое обновление партиций материализованного представления при обновлении любой из таблиц фактов.
- **Временная таблица измерений (v1.5.2+)**: Предположим, таблица измерений хранит исторические данные версий и партиционирована с определенной временной детализацией, и таблица фактов соединяется с таблицей измерений с той же временной детализацией. Установление отображения партиций между материализованным представлением и таблицей фактов, и таблицей измерений обеспечивает автоматическое обновление партиций материализованного представления при обновлении любой таблицы.

Материализованные представления на внешних каталогах

Внешний источник данных	Поддерживаемый сценарий и версии	Стабильные версии
Hive	• Непартиционированная таблица: v1.5.2+ • Партиция типа DATE и DATETIME: v1.5.2+ • Преобразование Partition Key типа STRING в тип DATE: v1.5.2+ • Материализованные представления на Hive View: Будет поддерживаться • Многоуровневое партиционирование: Будет поддерживаться	v1.5.2+
Iceberg	• Непартиционированная таблица: v1.5.2+ • Партиция типа DATE и DATETIME: v1.5.2+ • Преобразование Partition Key типа STRING в тип DATE: v1.5.2+ • Материализованные представления на Iceberg View: Будет поддерживаться • Partition Transform: v1.5.2 • Обновление на уровне партиций: v1.5.2+ • Многоуровневое партиционирование: v1.5.2+	v1.5.2+
Hudi	• Непартиционированная таблица: v1.5.2+ • Партиция типа DATE и DATETIME: v1.5.2+ • Многоуровневое партиционирование: Будет поддерживаться	Не стабильно
Paimon	• Непартиционированная таблица: v1.5.2+ • Партиция типа DATE и DATETIME: Будет поддерживаться • Многоуровневое партиционирование: Будет поддерживаться	Не стабильно
DeltaLake	• Непартиционированная таблица: v1.5.2+ • Партиционированная таблица: Будет поддерживаться • Многоуровневое партиционирование: Будет поддерживаться	Не стабильно
JDBC	• Непартиционированная таблица: v1.5.2+ • Партиционированная таблица: MySQL RangeColumn Partition v1.5.2	Не стабильно

> **ПРИМЕЧАНИЕ**
>
> Материализованные представления внешних таблиц не поддерживают автоматическое обновление **запускаемое изменениями данных базовой таблицы**. Они поддерживают только асинхронное обновление с **фиксированным интервалом** и ручное обновление.

Перезапись запросов

Функция	Описание	Поддерживаемые версии
Single Table Rewrite	Перезапись запросов с материализованными представлениями, построенными на одной внутренней таблице.	v1.5.2+
Inner Join Rewrite	Перезапись запросов для INNER/CROSS JOIN на внутренних таблицах.	v1.5.2+
Aggregate Rewrite	Перезапись запросов для соединений с основными агрегациями.	v1.5.2+
UNION Rewrite	Перезапись компенсации предиката UNION и перезапись компенсации партиции UNION на внутренних таблицах.	v1.5.2+
Nested Materialized View Rewrite	Перезапись запросов с вложенными материализованными представлениями на внутренних таблицах.	v1.5.2+
Count Distinct Rewrite (bitmap/hll)	Перезапись запросов для вычислений COUNT DISTINCT в вычисления на основе bitmap или HLL.	v1.5.2+
View Delta Join Rewrite	Перезапись запросов, которые соединяют таблицы, являющиеся подмножеством таблиц, которые соединяет материализованное представление.	v1.5.2+
Join Derivability Rewrite	Перезапись запросов между различными типами соединений.	v1.5.2+
Full Outer Join and Other Joins	Перезапись запросов для Full Outer Join, Semi Join и Anti Join.	v1.5.2+
Avg to Sum/Count Rewrite	Перезапись запросов для avg() в sum() / count()	v1.5.2+
View-based Rewrite	Перезапись запросов с материализованными представлениями, построенными на представлениях, без транскрибирования запросов к представлению в запросы к базовым таблицам представления.	v1.5.2+

Функция	Описание	Поддерживаемые версии
Count Distinct Rewrite (ArrayAgg)	Перезапись запросов для вычислений COUNT DISTINCT в вычисления с функцией `array_agg_distinct`.	v1.5.2+
Text-based Query Rewrite	Перезапись запроса, который имеет идентичное абстрактное синтаксическое дерево с определением материализованного представления.	v1.5.2+

Функции диагностики

Функция	Сценарий использования	Поддерживаемые версии
TRACE REWRITE	Использование инструкции TRACE REWRITE для диагностики проблем перезаписи.	v1.5.2+
Query Dump	Дамп информации о материализованном представлении при его запросе.	v1.5.2+
Refresh Audit Log	Запись SQL, выполненного в Audit Log при обновлении материализованного представления.	v1.5.2+
Hit Audit Log	Запись попавшего материализованного представления и кандидатов материализованных представлений в Audit Log, когда запрос переписывается в материализованное представление.	v1.5.2+
Monitoring Metrics	Выделенные метрики мониторинга для материализованных представлений.	v1.5.2+

6.6.6 Понимание выполнения задач материализованного представления

Информация о выполнении задач обновления материализованного представления может помочь понять поведение обновления материализованного представления, устранить проблемы и отслеживать производительность.

Обзор

Когда система обновляет материализованное представление, она создает выполнение задачи, которое содержит подробную информацию об операции обновления. Эта информация хранится в объекте `MVTaskRunExtraMessage` и может быть получена путем запроса поля `EXTRA_MESSAGE` в системном представлении `information_schema.task_runs`.

Дополнительное сообщение выполнения задач

В этом разделе описываются поля, предоставляемые в `MVTaskRunExtraMessage`.

`forceRefresh`

- **Тип**: Boolean
- **Описание**: Указывает, является ли это принудительным обновлением, которое обходит нормальные условия обновления. `true` возвращается, когда принудительное полное обновление запускается вручную с помощью `REFRESH MATERIALIZED VIEW ... FORCE`.

`partitionStart`

- **Тип**: String
- **Описание**: Начальная граница партиции для этой операции обновления. Она определяет нижнюю границу диапазона партиций для обновления.
- **Формат**: Значение ключа партиции (например, `"2024-01-01"` для партиций на основе дат)
- **Пример**: Когда обновляются данные с января 2024 года и далее, это поле будет `"2024-01-01"`.

`partitionEnd`

- **Тип**: String
- **Описание**: Конечная граница партиции для этой операции обновления. Она определяет верхнюю границу диапазона партиций для обновления.
- **Формат**: Значение ключа партиции (например, `"2024-01-31"` для партиций на основе дат)
- **Пример**: Когда обновляются данные до января 2024 года, это поле будет `"2024-01-31"`.

`mvPartitionsToRefresh`

- **Тип**: Set of Strings
- **Описание**: Список партиций материализованного представления, которые запланированы для обновления в выполнении задачи. Этот элемент помогает отслеживать, какие партиции материализованного представления будут обновлены.
- **Ограничение размера**: Система автоматически обрезает количество партиций до `max_mv_task_run_meta_message_values_length` (По умолчанию: 100), чтобы

предотвратить чрезмерное хранение метаданных, если фактическое количество превышает значение.

- **Пример**: `["p20240101", "p20240102", "p20240103"]`
- **Примечание**: Это представляет собственные партиции материализованного представления, а не партиции базовой таблицы.

`refBasePartitionsToRefreshMap`

- **Тип**: Map of String to Set of Strings
- **Описание**: Отображение между справочными базовыми таблицами и набором их партиций, которые должны быть обновлены.
- **Использование**: Значение устанавливается на этапе планировщика плана материализованного представления. Оно может использоваться для отслеживания того, какие партиции базовой таблицы необходимо просканировать.
- **Формат**: `{tableName -> Set<partitionName>}`
- **Ограничение размера**: Система автоматически обрезает количество партиций до `max_mv_task_run_meta_message_values_length` (По умолчанию: 100), чтобы предотвратить чрезмерное хранение метаданных, если фактическое количество превышает значение.
- **Пример**:

```
{
  "orders": ["p20240101", "p20240102"],
  "customers": ["p202401"]
}
```

- **Примечание**: Это **запланированный** набор партиций до оптимизации.

`basePartitionsToRefreshMap`

- **Тип**: Map of String to Set of Strings
- **Описание**: Отображение между справочными базовыми таблицами и набором их партиций, которые были обновлены во время выполнения.

- **Использование**: Значение устанавливается после фиксации карты версий материализованного представления. Оно отражает реальные партиции, используемые оптимизатором.

- **Формат**: `{tableName -> Set<partitionName>}`

- **Ограничение размера**: Система автоматически обрезает количество партиций до `max_mv_task_run_meta_message_values_length` (По умолчанию: 100), чтобы предотвратить чрезмерное хранение метаданных, если фактическое количество превышает значение.

- **Пример**:

```
{
  "orders": ["p20240101", "p20240102"],
  "line_items": ["p20240101_batch1", "p20240101_batch2"]
}
```

- **Примечание**: Это **фактический** набор партиций после оптимизации и выполнения запроса.

> **ПРИМЕЧАНИЕ**
>
> **Разница между refBasePartitionsToRefreshMap и basePartitionsToRefreshMap:**
>
> - `refBasePartitionsToRefreshMap`: Запланированные партиции до оптимизации (обычно для основной справочной таблицы)
> - `basePartitionsToRefreshMap`: Фактические партиции после оптимизации (включает все таблицы и оптимизированные наборы партиций)

`nextPartitionStart`

- **Тип**: String
- **Описание**: Начальная граница партиции для следующего инкрементного обновления. Это значение определяет нижнюю границу диапазона партиций, которые будут обновлены в следующем выполнении задачи, когда операция обновления разделена на несколько выполнений задач из-за ограничений ресурсов или больших объемов данных.

- **Пример**: Если текущее выполнение задачи обновляет данные до `"2024-01-15"`, значение этого поля может быть `"2024-01-16"`.

`nextPartitionEnd`

- **Тип**: String
- **Описание**: Конечная граница партиции для следующего инкрементного обновления. Это значение определяет верхнюю границу диапазона партиций, которые будут обновлены в следующем выполнении задачи, когда операция обновления разделена на несколько выполнений задач из-за ограничений ресурсов или больших объемов данных.
- **Пример**: `"2024-01-31"` для следующего пакета партиций для обработки.

`nextPartitionValues`

- **Тип**: String
- **Описание**: Сериализованные значения партиций для следующего обновления (используется для list-партиционирования или сложных схем партиционирования). Он хранит конкретные значения партиций, когда простых диапазонов начала/конца недостаточно.
- **Пример**: `"('US', 'ACTIVE'), ('UK', 'ACTIVE')"` для многостолбцовых list-партиций.

`processStartTime`

- **Тип**: Integer (временная метка в миллисекундах)
- **Описание**: Временная метка, когда выполнение задачи фактически начало обработку (время ожидания исключено). Она может использоваться для вычисления фактического времени обработки по формуле `Время обработки = Время завершения - Время начала процесса` (время ожидания в очереди исключено).
- **Пример**: `1704067200000` (2024-01-01 00:00:00 UTC)

`executeOption`

- **Тип**: ExecuteOption object
- **Описание**: Параметры конфигурации для выполнения задачи.
- **По умолчанию**: `Priority` = `LOWEST`, `isMergeRedundant` = `false`
- **Поля**:
  - `priority`: Приоритет выполнения задачи (значения из `Constants.TaskRunPriority`)
    - `HIGHEST`: 0

        - `HIGH`: 32
        - `NORMAL`: 64
        - `LOW`: 96
        - `LOWEST`: 127
    - `isMergeRedundant`: Объединять ли избыточные операции обновления.
    - `properties`: Дополнительные свойства выполнения в формате `Map<String, String>`.

`planBuilderMessage`

- **Тип**: Map of String to String
- **Описание**: Диагностические сообщения и метаданные от построителя плана запроса. Он содержит информацию о планировании запроса, решениях оптимизации и возможных проблемах.
- **Ограничение размера**: Система автоматически обрезает количество партиций до `max_mv_task_run_meta_message_values_length` (По умолчанию: 100), чтобы предотвратить чрезмерное хранение метаданных, если фактическое количество превышает значение.

`refreshMode`

- **Тип**: String
- **Описание**: Режим обновления этого выполнения задачи. Он указывает, как было обновлено материализованное представление.
- **Допустимые значения**:
    - `"COMPLETE"`: Полное обновление всех партиций
    - `"PARTIAL"`: Инкрементное обновление конкретных партиций
    - `"FORCE"`: Принудительное обновление, обходящее проверки устаревания
    - `""` (пусто): По умолчанию или не указано

`adaptivePartitionRefreshNumber`

- **Тип**: Integer
- **Описание**: Количество партиций, которые должны быть обновлены в каждой итерации, когда используется адаптивное обновление партиций. Это значение автоматически определяется на основе системных ресурсов и объема данных для оптимизации производительности обновления.
- **По умолчанию**: `-1` (указывает, что адаптивное обновление не установлено или не используется)
- **Пример**: `10` (указывает на обновление 10 партиций за раз)

Запрос деталей выполнения задачи

Вы можете запрашивать информацию о выполнении задач материализованного представления через системное представление `information_schema.task_runs`.

```
SELECT
    TASK_NAME,
    CREATE_TIME,
    FINISH_TIME,
    STATE,
    EXTRA_MESSAGE
FROM information_schema.task_runs
WHERE TASK_NAME LIKE 'mv-%'
ORDER BY CREATE_TIME DESC
LIMIT 10;
```

Столбец `EXTRA_MESSAGE` содержит JSON-представление `MVTaskRunExtraMessage`.

Вы можете дополнительно разобрать JSON-строку `EXTRA_MESSAGE` для лучшей читаемости.

```
SELECT
    TASK_NAME,
    CREATE_TIME,
    get_json_string(EXTRA_MESSAGE, '$.refreshMode') AS refresh_mode,
    get_json_string(EXTRA_MESSAGE, '$.forceRefresh') AS force_refresh,
    get_json_string(EXTRA_MESSAGE, '$.mvPartitionsToRefresh') AS
mv_partitions,
    get_json_int(EXTRA_MESSAGE, '$.processStartTime') AS process_start_ms,
    get_json_int(EXTRA_MESSAGE, '$.adaptivePartitionRefreshNumber') AS
adaptive_batch_size
FROM information_schema.task_runs
WHERE TASK_NAME = 'mv-12345'
ORDER BY CREATE_TIME DESC;
```

Понимание производительности обновления

Вычисление времени обработки

```
SELECT
    TASK_NAME,
    FINISH_TIME,
    get_json_bigint(EXTRA_MESSAGE, '$.processStartTime') AS
process_start_time,
    (unix_timestamp(FINISH_TIME) * 1000 -
     get_json_bigint(EXTRA_MESSAGE, '$.processStartTime')) / 1000 AS
processing_seconds
FROM information_schema.task_runs
WHERE TASK_NAME LIKE 'mv-%' AND STATE = 'SUCCESS';
```

Анализ шаблонов обновления партиций

```
SELECT
    TASK_NAME,
    CREATE_TIME,
    get_json_string(EXTRA_MESSAGE, '$.partitionStart') AS start_partition,
    get_json_string(EXTRA_MESSAGE, '$.partitionEnd') AS end_partition,
    get_json_string(EXTRA_MESSAGE, '$.nextPartitionStart') AS next_start,
    get_json_string(EXTRA_MESSAGE, '$.nextPartitionEnd') AS next_end
FROM information_schema.task_runs
WHERE TASK_NAME = 'mv-12345'
ORDER BY CREATE_TIME DESC;
```

Элементы конфигурации

max_mv_task_run_meta_message_values_length

- **Тип**: Integer
- **По умолчанию**: 100
- **Область действия**: Конфигурация FE
- **Описание**: Максимальное количество элементов для хранения в полях set или MAP, чтобы предотвратить чрезмерный рост метаданных. Он ограничивает размер `mvPartitionsToRefresh`, `refBasePartitionsToRefreshMap`, `basePartitionsToRefreshMap` и `planBuilderMessage`.

Лучшие практики

Мониторинг производительности обновления

- Сравните `processStartTime` с фактическим временем завершения, чтобы выявить проблемы с очередью.
- Используйте `adaptivePartitionRefreshNumber` для оптимизации размеров пакетов.

Отладка неудачных обновлений

- Проверьте `planBuilderMessage` на наличие проблем с оптимизатором.
- Сравните `refBasePartitionsToRefreshMap` с `basePartitionsToRefreshMap` на предмет проблем с обрезкой партиций.

Оптимизация инкрементных обновлений

- Отслеживайте `nextPartitionStart` и `nextPartitionEnd`, чтобы понять шаблоны обновления с несколькими итерациями.
- Отрегулируйте детализацию партиций, если обновления часто охватывают несколько выполнений.

Понимание покрытия партиций

- Сравните `mvPartitionsToRefresh` с `basePartitionsToRefreshMap`, чтобы проверить отображение партиций материализованного представления на базовую таблицу.
- Убедитесь, что партиции базовой таблицы соответствуют ожидаемым диапазонам обновления.

Устранение неполадок

Проблема: Обновление занимает слишком много времени

Проверьте:

1. `processStartTime` - Значительная разница со временем создания указывает, что выполнение задачи находилось в очереди.
2. `basePartitionsToRefreshMap` - Значительное значение указывает на слишком много партиций, которые сканируются.
3. `adaptivePartitionRefreshNumber` - Возможно, вам нужно настроить рабочую нагрузку.

Проблема: Обновлены неожиданные партиции

Проверьте:

1. `forceRefresh` - Если возвращается `true`, это указывает, что выполняется принудительное полное обновление.
2. `refBasePartitionsToRefreshMap` - Это поле показывает запланированные партиции.
3. `basePartitionsToRefreshMap` - Это поле показывает фактические партиции после оптимизации.
4. Сравните две вышеуказанные карты, чтобы увидеть, изменил ли оптимизатор план.

Проблема: Обновление застряло в нескольких итерациях

Проверьте:

1. `nextPartitionStart` и `nextPartitionEnd` - Это поле показывает неполное состояние обновления.
2. `adaptivePartitionRefreshNumber` - Возможно, вам нужно настроить рабочую нагрузку.
3. Рассмотрите возможность увеличения размера пакета или уменьшения детализации партиций.

См. также

- CREATE MATERIALIZED VIEW
- REFRESH MATERIALIZED VIEW
- Information Schema.task_runs

6.7 Colocate Join

Для shuffle join и broadcast join, если условие соединения выполняется, строки данных из двух соединяемых таблиц объединяются на одном узле для завершения соединения. Ни один из этих двух методов соединения не может избежать задержки или накладных расходов, вызванных сетевой передачей данных между узлами.

Основная идея заключается в том, чтобы сохранить ключ бакетинга, количество копий и размещение копий согласованными для таблиц в одной группе Colocation Group. Если столбец соединения является ключом бакетинга, вычислительному узлу нужно выполнить только локальное соединение без получения данных с других узлов. Colocate Join поддерживает equi joins.

Этот документ знакомит с принципом, реализацией, использованием и соображениями по использованию Colocate Join.

6.7.1 Терминология

- **Colocation Group (CG)**: CG будет содержать одну или несколько таблиц. Таблицы в CG имеют одинаковый бакетинг и размещение реплик, и описываются с помощью Colocation Group Schema.
- **Colocation Group Schema (CGS)**: CGS содержит ключ бакетинга, количество buckets и количество реплик CG.

6.7.2 Принцип

Colocate Join заключается в формировании CG с набором таблиц, имеющих одинаковый CGS, и обеспечении того, чтобы соответствующие копии bucket этих таблиц находились на одном и том же наборе узлов BE. Когда таблицы в CG выполняют операции Join по столбцам бакетинга, локальные данные могут быть соединены напрямую, что экономит время на передачу данных между узлами.

Bucket Seq получается путём `hash(key) mod buckets`. Предположим, таблица имеет 8 buckets, тогда есть [0, 1, 2, 3, 4, 5, 6, 7] 8 buckets, и каждый Bucket имеет один или несколько подтаблиц, количество подтаблиц зависит от количества партиций. Если это мультипартиционная таблица, будет несколько tablets.

Чтобы иметь одинаковое распределение данных, таблицы в пределах одной CG должны соответствовать следующему.

1. Таблицы в пределах одной CG должны иметь идентичный ключ бакетинга (тип, количество, порядок) и одинаковое количество buckets, чтобы фрагменты данных нескольких таблиц могли распределяться и контролироваться один за другим. Ключ бакетинга — это столбцы, указанные в операторе создания таблицы `DISTRIBUTED BY HASH(col1, col2, ...)`. Ключ бакетинга определяет, какие столбцы данных хешируются в разные Bucket Seqs. Имя ключа бакетинга может отличаться для таблиц в пределах одной CG. Столбцы бакетинга могут отличаться в операторе создания, но порядок соответствующих типов данных в `DISTRIBUTED BY HASH(col1, col2, ...)` должен быть точно таким же.
2. Таблицы в пределах одной CG могут иметь разное количество партиций и разные ключи партиционирования.

При создании таблицы CG указывается атрибутом `"colocate_with" = "group_name"` в PROPERTIES таблицы. Если CG не существует, это означает, что таблица является первой таблицей CG и называется Parent Table. Распределение данных Parent Table (тип, количество и порядок ключей разделения bucket, количество копий и количество разделённых buckets) определяет CGS. Если CG существует, проверяется, согласуется ли распределение данных таблицы с CGS.

Размещение копий таблиц в пределах одной CG удовлетворяет:

1. Сопоставление между Bucket Seq и узлами BE всех таблиц такое же, как у Parent Table.
2. Сопоставление между Bucket Seq и узлами BE всех партиций в Parent Table такое же, как у первой Partition.
3. Сопоставление между Bucket Seq и узлами BE первой Partition Parent Table определяется с использованием нативного алгоритма Round Robin.

Согласованное распределение данных и сопоставление гарантируют, что строки данных с одинаковым значением, взятым ключом бакетинга, попадают на один и тот же BE. Поэтому при использовании ключа бакетинга для соединения столбцов требуются только локальные соединения.

6.7.3 Использование

Создание таблицы

При создании таблицы вы можете указать атрибут `"colocate_with" = "group_name"` в PROPERTIES, чтобы указать, что таблица является таблицей Colocate Join и принадлежит указанной Colocation Group.

> **ПРИМЕЧАНИЕ**
>
> Начиная с версии 2.5.4, Colocate Join может выполняться для таблиц из разных баз данных. Вам нужно только указать одно и то же свойство `colocate_with` при создании таблиц.

Например:

```
CREATE TABLE tbl (k1 int, v1 int sum)
DISTRIBUTED BY HASH(k1)
BUCKETS 8
PROPERTIES(
    "colocate_with" = "group1"
);
```

Если указанная Group не существует, Selena автоматически создаёт Group, содержащую только текущую таблицу. Если Group существует, Selena проверяет, соответствует ли текущая таблица Colocation Group Schema. Если да, она создаёт таблицу и добавляет её в Group. В то же время таблица создаёт партицию и tablet на основе правил распределения данных существующей Group.

Colocation Group принадлежит базе данных. Имя Colocation Group уникально в пределах базы данных. Во внутреннем хранилище полное имя Colocation Group — это `dbId_groupName`, но вы воспринимаете только `groupName`.

> **ПРИМЕЧАНИЕ**
>
> Если вы указываете одну и ту же Colocation Group для связывания таблиц из разных баз данных для Colocate Join, Colocation Group существует в каждой из этих баз данных. Вы можете выполнить `show proc "/colocation_group"`, чтобы проверить Colocation Groups в разных базах данных.

Удаление

Полное удаление — это удаление из Корзины. Обычно после удаления таблицы командой `DROP TABLE` по умолчанию она остаётся в корзине в течение дня, прежде чем быть удалённой. Когда последняя таблица в Group полностью удалена, Group также будет удалена автоматически.

Просмотр информации о группе

Следующая команда позволяет вам просмотреть информацию о Group, которая уже существует в кластере.

```
SHOW PROC '/colocation_group';

+-------------+--------------+--------------+------------+----------------
+----------+----------+
| GroupId     | GroupName    | TableIds     | BucketsNum | ReplicationNum |
DistCols | IsStable |
+-------------+--------------+--------------+------------+----------------
+----------+----------+
| 10005.10008 | 10005_group1 | 10007, 10040 | 10         | 3              |
int(11)  | true     |
+-------------+--------------+--------------+------------+----------------
+----------+----------+
```

- **GroupId**: уникальный для всего кластера идентификатор Group, первая половина — это db id, вторая половина — это group id.
- **GroupName**: полное имя Group.
- **TabletIds**: список id таблиц в Group.
- **BucketsNum**: количество buckets.
- **ReplicationNum**: количество реплик.
- **DistCols**: столбцы распределения, т.е. тип столбца бакетинга.
- **IsStable**: стабильна ли Group (определение стабильности см. в разделе "Балансировка и восстановление реплик Colocation").

Вы можете дополнительно просмотреть распределение данных Group с помощью следующей команды.

```
SHOW PROC '/colocation_group/10005.10008';

+-------------+---------------------+
| BucketIndex | BackendIds          |
+-------------+---------------------+
| 0           | 10004, 10002, 10001 |
| 1           | 10003, 10002, 10004 |
| 2           | 10002, 10004, 10001 |
| 3           | 10003, 10002, 10004 |
| 4           | 10002, 10004, 10003 |
| 5           | 10003, 10002, 10001 |
| 6           | 10003, 10004, 10001 |
| 7           | 10003, 10004, 10002 |
+-------------+---------------------+
```

- **BucketIndex**: индекс последовательности buckets.
- **BackendIds**: идентификаторы узлов BE, где расположены фрагменты данных бакетинга.

> Примечание: Вышеуказанная команда требует привилегии NODE или роли `cluster_admin`. Обычные пользователи не могут получить к ней доступ.

Изменение свойств Group таблицы

Вы можете изменить свойства Colocation Group таблицы. Например:

```
ALTER TABLE tbl SET ("colocate_with" = "group2");
```

Если таблица ранее не была назначена Group, команда проверит Schema и добавит таблицу в Group (Group будет создана сначала, если она не существует). Если таблица была ранее назначена другой Group, команда удалит таблицу из исходной Group и добавит её в новую Group (Group будет создана сначала, если она не существует).

Вы также можете удалить свойства Colocation таблицы с помощью следующей команды.

```
ALTER TABLE tbl SET ("colocate_with" = "");
```

Другие связанные операции

При добавлении партиции с помощью `ADD PARTITION` или изменении количества копий в таблице с атрибутом Colocation, Selena проверяет, будет ли операция нарушать Colocation Group Schema, и отклоняет её в случае нарушения.

6.7.4 Балансировка и восстановление реплик Colocation

Распределение реплик таблицы Colocation должно следовать правилам распределения, указанным в схеме Group, поэтому оно отличается от обычного шардинга с точки зрения восстановления и балансировки реплик.

Сама Group имеет свойство `stable`. Когда `stable` равно `true`, это означает, что изменения в фрагментах таблиц в Group не производятся, и функция Colocation работает правильно. Когда `stable` равно `false`, это означает, что некоторые фрагменты таблиц в

текущей Group восстанавливаются или мигрируются, и Colocate Join затронутых таблиц деградирует до обычного Join.

Восстановление реплик

Реплики могут храниться только на указанном узле BE. Selena будет искать наименее загруженный BE для замены недоступного BE (например, down, decommission). После замены все фрагменты данных бакетинга на старом BE восстанавливаются. Во время миграции Group отмечается как **Unstable**.

Балансировка реплик

Selena пытается распределить фрагменты таблиц Colocation равномерно по всем узлам BE. Балансировка для обычных таблиц находится на уровне реплики, то есть каждая реплика индивидуально находит узел BE с меньшей нагрузкой. Балансировка для таблиц Colocation находится на уровне Bucket, то есть все реплики в пределах Bucket мигрируют вместе. Мы используем простой алгоритм балансировки, который распределяет `BucketsSequnce` равномерно по всем узлам BE, не учитывая фактический размер реплик, а только количество реплик. Точный алгоритм можно найти в комментариях к коду в `ColocateTableBalancer.java`.

> Примечание 1: Текущий алгоритм балансировки и восстановления реплик Colocation может не работать хорошо для кластеров Selena с гетерогенным развёртыванием. Так называемое гетерогенное развёртывание означает, что ёмкость диска, количество дисков и тип диска (SSD и HDD) узлов BE не согласуются. В случае гетерогенного развёртывания может случиться так, что узел BE малой ёмкости хранит такое же количество реплик, как узел BE большой ёмкости.
>
> Примечание 2: Когда Group находится в состоянии Unstable, Join её таблиц деградирует до обычного Join, что может значительно снизить производительность запросов кластера. Если вы не хотите, чтобы система автоматически балансировалась, установите конфигурацию FE `disable_colocate_balance`, чтобы отключить автоматическую балансировку, и включите её обратно в подходящее время. (Подробнее см. раздел "Продвинутые операции" (#Advanced Operations))

6.7.5 Запрос

Таблица Colocation запрашивается так же, как обычная таблица. Если Group, в которой находится таблица Colocation, находится в состоянии Unstable, она автоматически

деградирует до обычного Join, как показано в следующем примере.

Таблица 1:

```
CREATE TABLE `tbl1` (
    `k1` date NOT NULL COMMENT "",
    `k2` int(11) NOT NULL COMMENT "",
    `v1` int(11) SUM NOT NULL COMMENT ""
) ENGINE=OLAP
AGGREGATE KEY(`k1`, `k2`)
PARTITION BY RANGE(`k1`)
(
    PARTITION p1 VALUES LESS THAN ('2019-05-31'),
    PARTITION p2 VALUES LESS THAN ('2019-06-30')
)
DISTRIBUTED BY HASH(`k2`)  BUCKETS 6
PROPERTIES (
    "colocate_with" = "group1"
);
INSERT INTO tbl1
VALUES
    ("2015-09-12",1000,1),
    ("2015-09-13",2000,2);
```

Таблица 2:

```
CREATE TABLE `tbl2` (
    `k1` datetime NOT NULL COMMENT "",
    `k2` int(11) NOT NULL COMMENT "",
    `v1` double SUM NOT NULL COMMENT ""
) ENGINE=OLAP
AGGREGATE KEY(`k1`, `k2`)
DISTRIBUTED BY HASH(`k2`)  BUCKETS 6
PROPERTIES (
    "colocate_with" = "group1"
);
INSERT INTO tbl2
VALUES
    ("2015-09-12 00:00:00",3000,3),
    ("2015-09-12 00:00:00",4000,4);
```

Просмотр плана запроса:

```
EXPLAIN SELECT * FROM tbl1 INNER JOIN tbl2 ON (tbl1.k2 = tbl2.k2);
+------------------------------------------------------------------------------+
| Explain String                                                               |
+------------------------------------------------------------------------------+
| PLAN FRAGMENT 0                                                              |
|  OUTPUT EXPRS:1: k1 | 2: k2 | 3: v1 | 4: k1 | 5: k2 | 6: v1                |
|   PARTITION: UNPARTITIONED                                                   |
|                                                                              |
|   RESULT SINK                                                                |
|                                                                              |
|   3:EXCHANGE                                                                 |
|                                                                              |
| PLAN FRAGMENT 1                                                              |
|  OUTPUT EXPRS:                                                               |
|   PARTITION: RANDOM                                                          |
|                                                                              |
|   STREAM DATA SINK                                                           |
|     EXCHANGE ID: 03                                                          |
|     UNPARTITIONED                                                            |
|                                                                              |
|   2:HASH JOIN                                                                |
|   |  join op: INNER JOIN (COLOCATE)                                          |
|   |  colocate: true                                                          |
|   |  equal join conjunct: 5: k2 = 2: k2                                      |
|   |                                                                          |
|   |----1:OlapScanNode                                                        |
|   |       TABLE: tbl1                                                        |
|   |       PREAGGREGATION: OFF. Reason: Has can not pre-aggregation Join |
|   |       partitions=1/2                                                     |
|   |       rollup: tbl1                                                       |
|   |       tabletRatio=6/6                                                    |
|   |       tabletList=15344,15346,15348,15350,15352,15354                     |
|   |       cardinality=1                                                      |
|   |       avgRowSize=3.0                                                     |
|   |                                                                          |
|   0:OlapScanNode                                                             |
|      TABLE: tbl2                                                             |
|      PREAGGREGATION: OFF. Reason: None aggregate function                    |
|      partitions=1/1                                                          |
|      rollup: tbl2                                                            |
|      tabletRatio=6/6                                                         |
|      tabletList=15373,15375,15377,15379,15381,15383                          |
|      cardinality=1                                                           |
|      avgRowSize=3.0                                                          |
```

```
+-----------------------------------------------------------------------+
40 rows in set (0.03 sec)
```

Если Colocate Join действует, узел Hash Join отображает `colocate: true`.

Если он не действует, план запроса следующий:

```
+----------------------------------------------------+
| Explain String                                     |
+----------------------------------------------------+
| PLAN FRAGMENT 0                                    |
|  OUTPUT EXPRS:`tbl1`.`k1` |                        |
|   PARTITION: RANDOM                                |
|                                                    |
|   RESULT SINK                                      |
|                                                    |
|   2:HASH JOIN                                      |
|   |  join op: INNER JOIN (BROADCAST)               |
|   |  hash predicates:                              |
|   |  colocate: false, reason: group is not stable  |
|   |    `tbl1`.`k2` = `tbl2`.`k2`                   |
|   |  tuple ids: 0 1                                |
|   |                                                |
|   |----3:EXCHANGE                                  |
|   |       tuple ids: 1                             |
|   |                                                |
|   0:OlapScanNode                                   |
|      TABLE: tbl1                                   |
|      PREAGGREGATION: OFF. Reason: No AggregateInfo |
|      partitions=0/2                                |
|      rollup: null                                  |
|      buckets=0/0                                   |
|      cardinality=-1                                |
|      avgRowSize=0.0                                |
|      numNodes=0                                    |
|      tuple ids: 0                                  |
|                                                    |
| PLAN FRAGMENT 1                                    |
|  OUTPUT EXPRS:                                     |
|   PARTITION: RANDOM                                |
|                                                    |
|   STREAM DATA SINK                                 |
|     EXCHANGE ID: 03                                |
|     UNPARTITIONED                                  |
|                                                    |
|   1:OlapScanNode                                   |
|      TABLE: tbl2                                   |
|      PREAGGREGATION: OFF. Reason: null             |
|      partitions=0/1                                |
|      rollup: null                                  |
|      buckets=0/0                                   |
|      cardinality=-1                                |
```

```
|      avgRowSize=0.0                                |
|      numNodes=0                                    |
|      tuple ids: 1                                  |
+----------------------------------------------------+
```

Узел HASH JOIN покажет соответствующую причину: `colocate: false, reason: group is not stable`. Одновременно будет сгенерирован узел EXCHANGE.

6.7.6 Продвинутые операции

Элементы конфигурации FE

- **disable_colocate_relocate**

Отключать ли автоматическое восстановление реплик Colocation для Selena. По умолчанию false, что означает, что оно включено. Этот параметр влияет только на восстановление реплик для таблиц Colocation, а не для обычных таблиц.

- **disable_colocate_balance**

Отключать ли автоматическую балансировку реплик Colocation для Selena. По умолчанию false, что означает, что она включена. Этот параметр влияет только на балансировку реплик таблиц Colocation, а не для обычных таблиц.

- **disable_colocate_join**

  Вы можете отключить Colocate join на уровне сессии, изменив эту переменную.

- **disable_colocate_join**

  Функцию Colocate join можно отключить, изменив эту переменную.

HTTP Restful API

Selena предоставляет несколько HTTP Restful API, связанных с Colocate Join, для просмотра и изменения Colocation Groups.

Этот API реализован на FE и доступен с использованием `fe_host:fe_http_port` с правами `db_admin` и `user_admin`.

1. Просмотр всей информации Colocation кластера

```
curl --location-trusted -u<username>:<password> 'http://<fe_host>:
<fe_http_port>/api/colocate'
```

```
// Возвращает внутреннюю информацию Colocation в формате Json.
{
    "colocate_meta": {
        "groupName2Id": {
            "g1": {
                "dbId": 10005,
                "grpId": 10008
            }
        },
        "group2Tables": {},
        "table2Group": {
            "10007": {
                "dbId": 10005,
                "grpId": 10008
            },
            "10040": {
                "dbId": 10005,
                "grpId": 10008
            }
        },
        "group2Schema": {
            "10005.10008": {
                "groupId": {
                    "dbId": 10005,
                    "grpId": 10008
                },
                "distributionColTypes": [{
                    "type": "INT",
                    "len": -1,
                    "isAssignedStrLenInColDefinition": false,
                    "precision": 0,
                    "scale": 0
                }],
                "bucketsNum": 10,
                "replicationNum": 2
            }
        },
        "group2BackendsPerBucketSeq": {
            "10005.10008": [
                [10004, 10002],
                [10003, 10002],
                [10002, 10004],
                [10003, 10002],
                [10002, 10004],
                [10003, 10002],
```

```
                [10003, 10004],
                [10003, 10004],
                [10003, 10004],
                [10002, 10004]
            ]
        },
        "unstableGroups": []
    },
    "status": "OK"
}
```

2. Пометить Group как Stable или Unstable

```
# Пометить как Stable
curl -XPOST --location-trusted -u<username>:<password>
'http://<fe_host>:<fe_http_port>/api/colocate/group_stable?db_id=
<dbId>&group_id=<grpId>'
# Пометить как Unstable
curl -XPOST --location-trusted -u<username>:<password>
'http://<fe_host>:<fe_http_port>/api/colocate/group_unstable?db_id=
<dbId>&group_id=<grpId>'
```

Если возвращённый результат — `200`, Group успешно помечена как Stable или Unstable.

3. Установить распределение данных Group

Этот интерфейс позволяет вам принудительно задать числовое распределение Group.

`POST /api/colocate/bucketseq?db_id=10005&group_id= 10008`

`Body:`

`[[10004,10002],[10003,10002],[10002,10004],[10003,10002],[10002,10004],[10003,10002],[10003,10004],[10003,10004],[10003,10004],[10002,10004]]`

`returns: 200`

Где `Body` — это `BucketsSequence`, представленная как вложенный массив, и идентификаторы BE, в которых расположены фрагменты бакетинга.

> Обратите внимание, что для использования этой команды вам может потребоваться установить конфигурацию FE `disable_colocate_relocate` и `disable_colocate_balance` в true, то есть отключить автоматическое восстановление и балансировку реплик Colocation в системе. В противном случае она может быть автоматически сброшена системой после изменения.

6.8 Использование Lateral Join для преобразования столбцов в строки

Преобразование столбцов в строки - это распространенная операция в обработке ETL. Lateral - это специальное ключевое слово Join, которое может ассоциировать строку с внутренним подзапросом или табличной функцией. Используя Lateral в сочетании с unnest(), вы можете развернуть одну строку в несколько строк. Для получения дополнительной информации см. unnest.

6.8.1 Ограничения

- В настоящее время Lateral Join используется только с unnest() для достижения преобразования столбцов в строки. Другие табличные функции и UDTF будут поддерживаться позже.
- В настоящее время Lateral Join не поддерживает подзапросы.

6.8.2 Использование Lateral Join

Синтаксис:

```
from table_reference join [lateral] table_reference;
```

Примеры:

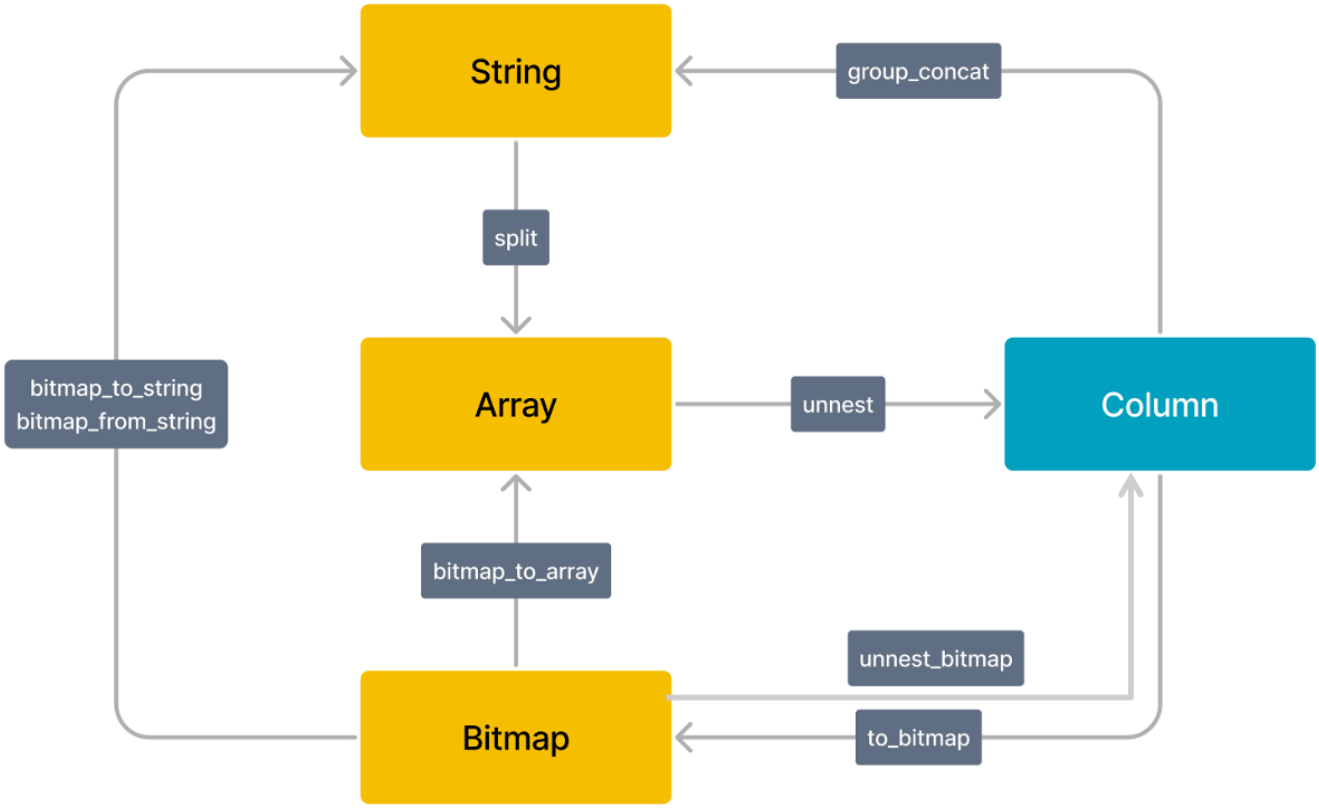
```
SELECT student, score
FROM tests
CROSS JOIN LATERAL UNNEST(scores) AS t (score);

SELECT student, score
FROM tests, UNNEST(scores) AS t (score);
```

Второй синтаксис здесь - это сокращенная версия первого, где ключевое слово Lateral может быть опущено при использовании ключевого слова UNNEST. Ключевое слово

UNNEST - это табличная функция, которая преобразует массив в несколько строк. Вместе с Lateral Join она может реализовать общую логику развертывания строк.

> **ПРИМЕЧАНИЕ**
>
> Если вы хотите выполнить unnest для нескольких столбцов, вы должны указать псевдоним для каждого столбца, например, `select v1, t1.unnest as v2, t2.unnest as v3 from lateral_test, unnest(v2) t1, unnest(v3) t2;`.

Selena поддерживает преобразование типов между BITMAP, STRING, ARRAY и Column.



6.8.3 Примеры использования

Вместе с unnest() вы можете достичь следующих функций преобразования столбцов в строки:

Развертывание строки в несколько строк

1. Создайте таблицу и вставьте данные в эту таблицу.

```
CREATE TABLE lateral_test2 (
    `v1` bigint(20) NULL COMMENT "",
    `v2` string NULL COMMENT ""
)
DUPLICATE KEY(v1)
DISTRIBUTED BY HASH(`v1`)
PROPERTIES (
    "replication_num" = "3",
    "storage_format" = "DEFAULT"
);

INSERT INTO lateral_test2 VALUES (1, "1,2,3"), (2, "1,3");
```

2. Запросите данные до развертывания.

```
select * from lateral_test2;

+------+-------+
| v1   | v2    |
+------+-------+
|    1 | 1,2,3 |
|    2 | 1,3   |
+------+-------+
```

3. Разверните `v2` в несколько строк.

```
-- Выполнить unnest для одного столбца.

select v1,unnest from lateral_test2, unnest(split(v2, ",")) as unnest;

+------+--------+
| v1   | unnest |
+------+--------+
|    1 | 1      |
|    1 | 2      |
|    1 | 3      |
|    2 | 1      |
|    2 | 3      |
+------+--------+

-- Выполнить unnest для нескольких столбцов. Вы должны указать
псевдоним для каждой операции.

select v1, t1.unnest as v2, t2.unnest as v3 from lateral_test2,
unnest(split(v2, ",")) t1, unnest(split(v3, ",")) t2;

+------+------+------+
| v1   | v2   | v3   |
+------+------+------+
|    1 | 1    | 1    |
|    1 | 1    | 2    |
|    1 | 2    | 1    |
|    1 | 2    | 2    |
|    1 | 3    | 1    |
|    1 | 3    | 2    |
|    2 | 1    | 1    |
|    2 | 1    | 3    |
|    2 | 3    | 1    |
|    2 | 3    | 3    |
+------+------+------+
```

Развертывание массива в несколько строк

Начиная с v1.5.2, unnest() может принимать несколько массивов разных типов и длин. Для получения дополнительной информации см. unnest().

1. Создайте таблицу и вставьте данные в эту таблицу.

```
CREATE TABLE lateral_test (
    `v1` bigint(20) NULL COMMENT "",
    `v2` ARRAY NULL COMMENT ""
)
DUPLICATE KEY(v1)
DISTRIBUTED BY HASH(`v1`)
PROPERTIES (
    "replication_num" = "3",
    "storage_format" = "DEFAULT"
);

INSERT INTO lateral_test VALUES (1, [1,2]), (2, [1, null, 3]), (3,
null);
```

2. Запросите данные до развертывания.

```
select * from lateral_test;

+------+------------+
| v1   | v2         |
+------+------------+
|    1 | [1,2]      |
|    2 | [1,null,3] |
|    3 | NULL       |
+------+------------+
```

3. Разверните `v2` в несколько строк.

```
select v1,v2,unnest from lateral_test , unnest(v2) as unnest;

+------+------------+--------+
| v1   | v2         | unnest |
+------+------------+--------+
|    1 | [1,2]      |      1 |
|    1 | [1,2]      |      2 |
|    2 | [1,null,3] |      1 |
|    2 | [1,null,3] |   NULL |
|    2 | [1,null,3] |      3 |
+------+------------+--------+
```

Развертывание данных Bitmap

Вы можете использовать Lateral Join с функцией unnest_bitmap для развертывания данных Bitmap.

1. Создайте таблицу и вставьте данные в эту таблицу.

```
CREATE TABLE lateral_test3 (
`v1` bigint(20) NULL COMMENT "",
`v2` Bitmap BITMAP_UNION COMMENT ""
)
AGGREGATE KEY(v1)
DISTRIBUTED BY HASH(`v1`);

INSERT INTO lateral_test3 VALUES (1, bitmap_from_string('1, 2')), (2,
to_bitmap(3));
```

2. Запросите данные до развертывания.

```
select v1, bitmap_to_string(v2) from lateral_test3;

+------+------------------------+
| v1   | bitmap_to_string(`v2`) |
+------+------------------------+
|    1 | 1,2                    |
|    2 | 3                      |
+------+------------------------+
```

3. Вставьте новую строку.

```
insert into lateral_test3 values (1, to_bitmap(3));

select v1, bitmap_to_string(v2) from lateral_test3;

+------+------------------------+
| v1   | bitmap_to_string(`v2`) |
+------+------------------------+
|    1 | 1,2,3                  |
|    2 | 3                      |
+------+------------------------+
```

4. Разверните данные в `v2` в несколько строк.

```
mysql> select v1, unnest_bitmap from lateral_test3, unnest_bitmap(v2)
as unnest_bitmap;

+------+---------------+
| v1   | unnest_bitmap |
+------+---------------+
|    1 |             1 |
|    1 |             2 |
|    1 |             3 |
|    2 |             3 |
+------+---------------+
```

6.9 Caching

6.9.1 Кэш запросов

Кэш запросов — это мощная функция Selena, которая может значительно повысить производительность агрегатных запросов. Сохраняя промежуточные результаты локальных агрегаций в памяти, кэш запросов может избежать ненужного доступа к диску и вычислений для новых запросов, которые идентичны или похожи на предыдущие. Благодаря кэшу запросов Selena может быстро и точно возвращать результаты агрегатных запросов, экономя время и ресурсы и обеспечивая лучшую масштабируемость. Кэш запросов особенно полезен в сценариях с высокой параллельностью, когда много пользователей выполняют похожие запросы к большим и сложным наборам данных.

Эта функция поддерживается в cluster с раздельным хранением (shared-nothing) начиная с версии v1.5.2 и в cluster с разделяемыми данными (shared-data) начиная с версии v1.5.2.

В версии v1.5.2 кэш запросов поддерживает только агрегатные запросы к отдельным плоским таблицам. Начиная с версии v1.5.2, кэш запросов также поддерживает агрегатные запросы к нескольким таблицам, объединённым по схеме «звезда».

Сценарии применения

Рекомендуется использовать кэш запросов в следующих сценариях:

- Вы часто выполняете агрегатные запросы к отдельным плоским таблицам или к нескольким объединённым таблицам, соединённым по схеме «звезда».

- Большинство ваших агрегатных запросов — это агрегатные запросы без GROUP BY и агрегатные запросы с GROUP BY с низкой кардинальностью.
- Ваши данные загружаются в режиме добавления по временным partition и могут быть классифицированы как горячие и холодные данные на основе частоты доступа.

Кэш запросов поддерживает запросы, соответствующие следующим условиям:

- Используется движок Pipeline. Чтобы включить движок Pipeline, установите сессионную переменную `enable_pipeline_engine` в значение `true`.

  > **ПРИМЕЧАНИЕ**
  >
  > Другие движки запросов не поддерживают кэш запросов.

- Запросы выполняются к нативным OLAP-таблицам (с версии v1.5.2) или облачным нативным таблицам (с версии v1.5.2). Кэш запросов не поддерживает запросы к внешним таблицам. Кэш запросов также поддерживает запросы, планы которых требуют доступа к синхронным материализованным представлениям. Однако кэш запросов не поддерживает запросы, планы которых требуют доступа к асинхронным материализованным представлениям.

- Запросы являются агрегатными запросами к отдельным таблицам или к нескольким объединённым таблицам.

  ПРИМЕЧАНИЕ

  > - Кэш запросов поддерживает Broadcast Join и Bucket Shuffle Join.
  > - Кэш запросов поддерживает две древовидные структуры, содержащие операторы Join: Aggregation-Join и Join-Aggregation. Shuffle join не поддерживаются в древовидной структуре Aggregation-Join, а Hash join не поддерживаются в древовидной структуре Join-Aggregation.

- Запросы не включают недетерминированные функции, такие как `rand`, `random`, `uuid` и `sleep`.

Кэш запросов поддерживает запросы к таблицам, использующим любую из следующих политик партиционирования: без партиционирования, многостолбцовое партиционирование и одностолбцовое партиционирование.

Ограничения функции

- Кэш запросов основан на вычислениях по tablet движка Pipeline. Вычисление по tablet означает, что pipeline driver может обрабатывать целые tablet по одному, а не обрабатывать часть tablet или много tablet вперемешку. Если количество tablet, которые должны быть обработаны каждым отдельным BE для запроса, больше или равно количеству pipeline driver, которые вызываются для выполнения этого запроса, кэш запросов работает. Количество вызываемых pipeline driver представляет фактическую степень параллелизма (DOP). Если количество tablet меньше количества pipeline driver, каждый pipeline driver обрабатывает только часть конкретного tablet. В этой ситуации результаты вычислений по tablet не могут быть получены, и поэтому кэш запросов не работает.
- В Selena агрегатный запрос состоит как минимум из четырёх стадий. Результаты вычислений по tablet, генерируемые AggregateNode на первой стадии, могут быть кэшированы только когда OlapScanNode и AggregateNode вычисляют данные из одного фрагмента. Результаты вычислений по tablet, генерируемые AggregateNode на других стадиях, не могут быть кэшированы. Для некоторых агрегатных запросов DISTINCT, если сессионная переменная `cbo_cte_reuse` установлена в `true`, кэш запросов не работает, когда OlapScanNode, который производит данные, и AggregateNode первой стадии, который потребляет произведённые данные, вычисляют данные из разных фрагментов и связаны через ExchangeNode. Следующие два примера показывают сценарии, в которых выполняются CTE-оптимизации и поэтому кэш запросов не работает:
  - Выходные столбцы вычисляются с использованием агрегатной функции `avg(distinct)`.
  - Выходные столбцы вычисляются несколькими агрегатными функциями DISTINCT.
- Если ваши данные перетасовываются перед агрегацией, кэш запросов не может ускорить запросы к этим данным.
- Если столбцы group-by или столбцы дедупликации таблицы имеют высокую кардинальность, для агрегатных запросов к этой таблице будут генерироваться большие результаты. В этих случаях запросы будут обходить кэш запросов во время выполнения.
- Кэш запросов занимает небольшой объём памяти, предоставляемой BE, для сохранения результатов вычислений. Размер кэша запросов по умолчанию составляет 512 МБ. Поэтому кэш запросов не подходит для сохранения элементов данных большого размера. Кроме того, после включения кэша запросов производительность запросов снижается, если коэффициент попадания в кэш низкий.

Поэтому, если размер результатов вычислений, сгенерированных для tablet, превышает порог, указанный параметром `query_cache_entry_max_bytes` или `query_cache_entry_max_rows`, кэш запросов больше не работает для запроса, и запрос переключается в режим Passthrough.

Принцип работы

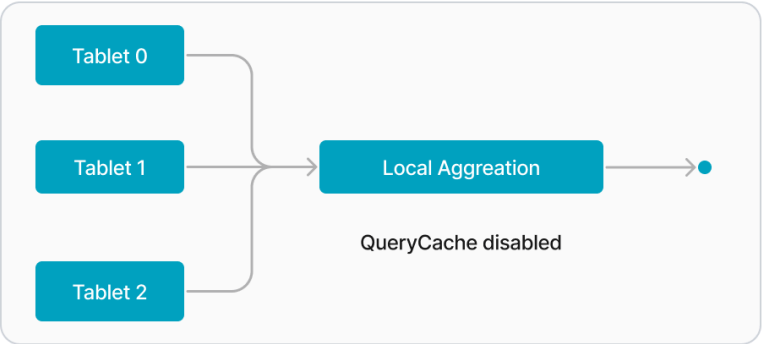
Когда кэш запросов включён, каждый BE разделяет локальную агрегацию запроса на следующие две стадии:

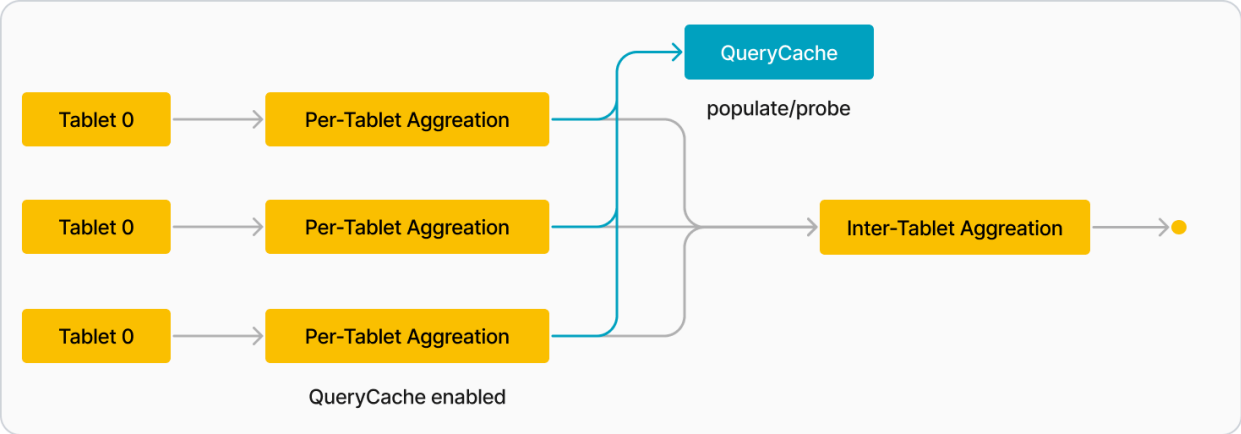
1. Агрегация по tablet

   BE обрабатывает каждый tablet индивидуально. Когда BE начинает обработку tablet, он сначала проверяет кэш запросов, чтобы узнать, находится ли промежуточный результат агрегации по этому tablet в кэше запросов. Если да (попадание в кэш), BE напрямую извлекает промежуточный результат из кэша запросов. Если нет (промах кэша), BE обращается к данным на диске и выполняет локальную агрегацию для вычисления промежуточного результата. Когда BE заканчивает обработку tablet, он заполняет кэш запросов промежуточным результатом агрегации по этому tablet.

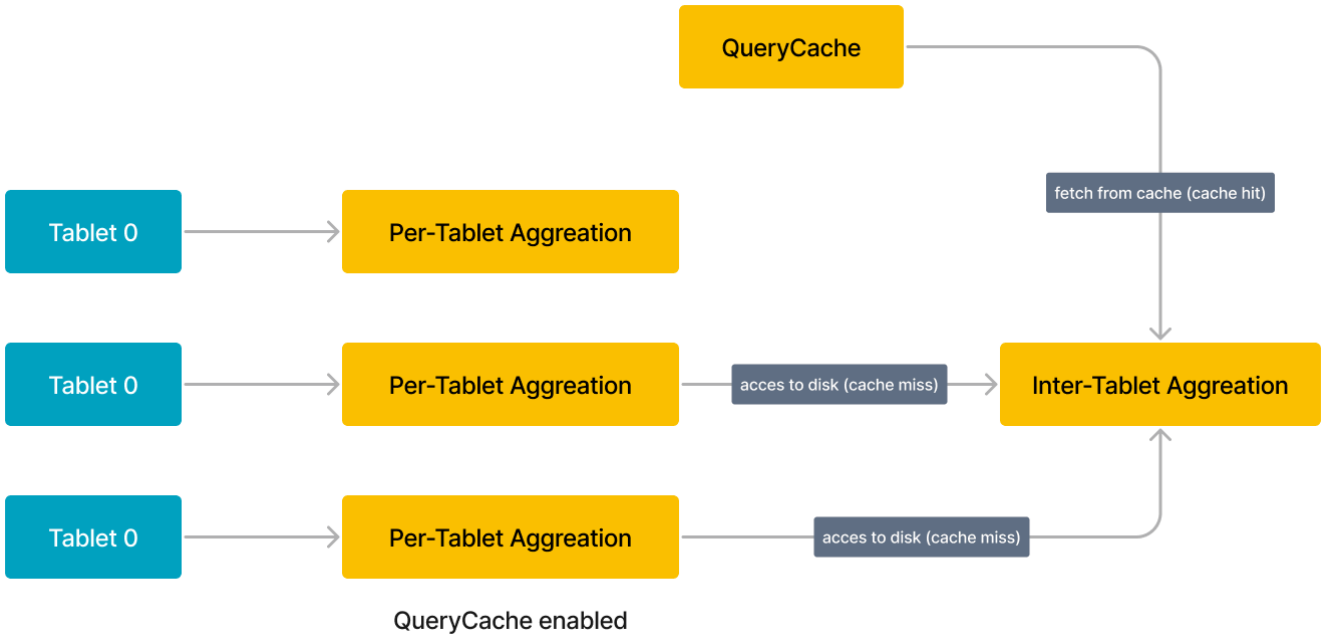
2. Межtablet агрегация

   BE собирает промежуточные результаты со всех tablet, участвующих в запросе, и объединяет их в окончательный результат.





Когда в будущем выдаётся похожий запрос, он может повторно использовать кэшированный результат предыдущего запроса. Например, запрос, показанный на следующем рисунке, включает три tablet (Tablet 0–2), и промежуточный результат для первого tablet (Tablet 0) уже находится в кэше запросов. В этом примере BE может напрямую извлечь результат для Tablet 0 из кэша запросов вместо обращения к данным на диске. Если кэш запросов полностью прогрет, он может содержать промежуточные результаты для всех трёх tablet, и поэтому BE не нужно обращаться к каким-либо данным на диске.



Для освобождения дополнительной памяти кэш запросов использует политику вытеснения на основе LRU (Least Recently Used) для управления записями кэша в нём. Согласно этой политике вытеснения, когда объём памяти, занимаемой кэшем запросов, превышает его предопределённый размер (`query_cache_capacity`), наименее недавно использованные записи кэша вытесняются из кэша запросов.

> **ПРИМЕЧАНИЕ**
>
> В будущем Selena также будет поддерживать политику вытеснения на основе TTL (Time to Live), согласно которой записи кэша могут быть вытеснены из кэша запросов.

FE определяет, нужно ли ускорять каждый запрос с помощью кэша запросов, и нормализует запросы для устранения тривиальных литеральных деталей, которые не влияют на семантику запросов.

Для предотвращения снижения производительности, вызванного плохими случаями кэша запросов, BE применяет адаптивную политику для обхода кэша запросов во время выполнения.

Включение кэша запросов

В этом разделе описываются параметры и сессионные переменные, используемые для включения и настройки кэша запросов.

Сессионные переменные FE

Переменная	Значение по умолчанию	Динамическая настройка	Описание
enable_query_cache	false	Да	Указывает, включать ли кэш запросов. Допустимые значения: `true` и `false`. `true` включает эту функцию, а `false` отключает её. Когда кэш запросов включён, он работает только для запросов, соответствующих условиям, указанным в разделе «Сценарии применения» этой темы.
query_cache_entry_max_bytes	4194304	Да	Указывает порог для запуска режима Passthrough. Допустимые значения: от `0` до `9223372036854775807`. Когда количество байтов или строк из результатов вычислений конкретного tablet, к которому обращается запрос, превышает порог, указанный параметром `query_cache_entry_max_bytes` или `query_cache_entry_max_rows`, запрос переключается в режим Passthrough. Если параметр `query_cache_entry_max_bytes` или `query_cache_entry_max_rows` установлен в `0`, режим Passthrough используется даже когда результаты вычислений не генерируются из задействованных tablet.
query_cache_entry_max_rows	409600	Да	То же, что и выше.

Параметры BE

Вам нужно настроить следующий параметр в файле конфигурации BE **be.conf**. После перенастройки этого параметра для BE вы должны перезапустить BE, чтобы новая настройка параметра вступила в силу.

Параметр	Обязательный	Описание
query_cache_capacity	Нет	Указывает размер кэша запросов. Единица измерения: байты. Размер по умолчанию — 512 МБ. Каждый BE имеет свой собственный локальный кэш запросов в памяти и заполняет и проверяет только свой собственный кэш запросов. Обратите внимание, что размер кэша запросов не может быть меньше 4 МБ. Если объём памяти BE недостаточен для обеспечения ожидаемого размера кэша запросов, вы можете увеличить объём памяти BE.

Разработан для максимального коэффициента попадания в кэш во всех сценариях

Рассмотрим три сценария, в которых кэш запросов всё ещё эффективен, даже когда запросы не идентичны буквально. Эти три сценария:

- Семантически эквивалентные запросы
- Запросы с перекрывающимися сканируемыми partition
- Запросы к данным только с добавлением (без операций UPDATE или DELETE)

Семантически эквивалентные запросы

Когда два запроса похожи, что не означает, что они должны быть буквально эквивалентны, но означает, что они содержат семантически эквивалентные фрагменты в своих планах выполнения, они считаются семантически эквивалентными и могут повторно использовать результаты вычислений друг друга. В широком смысле два запроса семантически эквивалентны, если они запрашивают данные из одного источника, используют один и тот же метод вычислений и имеют один и тот же план выполнения. Selena применяет следующие правила для оценки семантической эквивалентности двух запросов:

- Если два запроса содержат несколько агрегаций, они оцениваются как семантически эквивалентные, если их первые агрегации семантически эквивалентны. Например, следующие два запроса, Q1 и Q2, оба содержат несколько агрегаций, но их первые агрегации семантически эквивалентны. Поэтому Q1 и Q2 оцениваются как семантически эквивалентные.

    - Q1

```
SELECT
    (
        ifnull(sum(murmur_hash3_32(hour)), 0) +
ifnull(sum(murmur_hash3_32(k0)), 0) +
ifnull(sum(murmur_hash3_32(__c_0)), 0)
    ) AS fingerprint
FROM
    (
        SELECT
            date_trunc('hour', ts) AS hour,
            k0,
            sum(v1) AS __c_0
        FROM
            t0
        WHERE
            ts between '2022-01-03 00:00:00'
            and '2022-01-03 23:59:59'
        GROUP BY
            date_trunc('hour', ts),
            k0
    ) AS t;
```

- Q2

```
SELECT
    date_trunc('hour', ts) AS hour,
    k0,
    sum(v1) AS __c_0
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59'
GROUP BY
    date_trunc('hour', ts),
    k0
```

- Если оба запроса относятся к одному из следующих типов запросов, они могут быть оценены как семантически эквивалентные. Обратите внимание, что запросы, включающие предложение HAVING, не могут быть оценены как семантически эквивалентные запросам, не включающим предложение HAVING. Однако включение

предложения ORDER BY или LIMIT не влияет на оценку семантической эквивалентности двух запросов.

- Агрегации GROUP BY

```
SELECT <GroupByItems>, <AggFunctionItems>
FROM <Table>
WHERE <Predicates> [and <PartitionColumnRangePredicate>]
GROUP BY <GroupByItems>
[HAVING <HavingPredicate>]
```

> **ПРИМЕЧАНИЕ**
>
> В предыдущем примере предложение HAVING является необязательным.

- Агрегации GROUP BY DISTINCT

```
SELECT DISTINCT <GroupByItems>, <Items>
FROM <Table>
WHERE <Predicates> [and <PartitionColumnRangePredicate>]
GROUP BY <GroupByItems>
HAVING <HavingPredicate>
```

> **ПРИМЕЧАНИЕ**
>
> В предыдущем примере предложение HAVING является необязательным.

- Агрегации без GROUP BY

```
SELECT <AggFunctionItems> FROM <Table>
WHERE <Predicates> [and <PartitionColumnRangePredicate>]
```

- Агрегации без GROUP BY DISTINCT

```
SELECT DISTINCT <Items> FROM <Table>
WHERE <Predicates> [and <PartitionColumnRangePredicate>]
```

- Если любой запрос включает `PartitionColumnRangePredicate`, он удаляется перед оценкой двух запросов на семантическую эквивалентность. `PartitionColumnRangePredicate` указывает один из следующих типов предикатов, ссылающихся на столбец партиционирования:
  - `col between v1 and v2`: Значения столбца партиционирования находятся в диапазоне [v1, v2], где `v1` и `v2` — константные выражения.
  - `v1 < col and col < v2`: Значения столбца партиционирования находятся в диапазоне (v1, v2), где `v1` и `v2` — константные выражения.
  - `v1 < col and col <= v2`: Значения столбца партиционирования находятся в диапазоне (v1, v2], где `v1` и `v2` — константные выражения.
  - `v1 <= col and col < v2`: Значения столбца партиционирования находятся в диапазоне [v1, v2), где `v1` и `v2` — константные выражения.
  - `v1 <= col and col <= v2`: Значения столбца партиционирования находятся в диапазоне [v1, v2], где `v1` и `v2` — константные выражения.
- Если выходные столбцы предложений SELECT двух запросов одинаковы после их перестановки, два запроса оцениваются как семантически эквивалентные.
- Если выходные столбцы предложений GROUP BY двух запросов одинаковы после их перестановки, два запроса оцениваются как семантически эквивалентные.
- Если оставшиеся предикаты предложений WHERE двух запросов семантически эквивалентны после удаления `PartitionColumnRangePredicate`, два запроса оцениваются как семантически эквивалентные.
- Если предикаты в предложениях HAVING двух запросов семантически эквивалентны, два запроса оцениваются как семантически эквивалентные.

Используйте следующую таблицу `lineorder_flat` в качестве примера:

```
CREATE TABLE `lineorder_flat`
(
    `lo_orderdate` date NOT NULL COMMENT "",
    `lo_orderkey` int(11) NOT NULL COMMENT "",
    `lo_linenumber` tinyint(4) NOT NULL COMMENT "",
    `lo_custkey` int(11) NOT NULL COMMENT "",
    `lo_partkey` int(11) NOT NULL COMMENT "",
    `lo_suppkey` int(11) NOT NULL COMMENT "",
    `lo_orderpriority` varchar(100) NOT NULL COMMENT "",
    `lo_shippriority` tinyint(4) NOT NULL COMMENT "",
    `lo_quantity` tinyint(4) NOT NULL COMMENT "",
    `lo_extendedprice` int(11) NOT NULL COMMENT "",
    `lo_ordtotalprice` int(11) NOT NULL COMMENT "",
    `lo_discount` tinyint(4) NOT NULL COMMENT "",
    `lo_revenue` int(11) NOT NULL COMMENT "",
    `lo_supplycost` int(11) NOT NULL COMMENT "",
    `lo_tax` tinyint(4) NOT NULL COMMENT "",
    `lo_commitdate` date NOT NULL COMMENT "",
    `lo_shipmode` varchar(100) NOT NULL COMMENT "",
    `c_name` varchar(100) NOT NULL COMMENT "",
    `c_address` varchar(100) NOT NULL COMMENT "",
    `c_city` varchar(100) NOT NULL COMMENT "",
    `c_nation` varchar(100) NOT NULL COMMENT "",
    `c_region` varchar(100) NOT NULL COMMENT "",
    `c_phone` varchar(100) NOT NULL COMMENT "",
    `c_mktsegment` varchar(100) NOT NULL COMMENT "",
    `s_name` varchar(100) NOT NULL COMMENT "",
    `s_address` varchar(100) NOT NULL COMMENT "",
    `s_city` varchar(100) NOT NULL COMMENT "",
    `s_nation` varchar(100) NOT NULL COMMENT "",
    `s_region` varchar(100) NOT NULL COMMENT "",
    `s_phone` varchar(100) NOT NULL COMMENT "",
    `p_name` varchar(100) NOT NULL COMMENT "",
    `p_mfgr` varchar(100) NOT NULL COMMENT "",
    `p_category` varchar(100) NOT NULL COMMENT "",
    `p_brand` varchar(100) NOT NULL COMMENT "",
    `p_color` varchar(100) NOT NULL COMMENT "",
    `p_type` varchar(100) NOT NULL COMMENT "",
    `p_size` tinyint(4) NOT NULL COMMENT "",
    `p_container` varchar(100) NOT NULL COMMENT ""
)
ENGINE=OLAP
DUPLICATE KEY(`lo_orderdate`, `lo_orderkey`)
COMMENT "olap"
PARTITION BY RANGE(`lo_orderdate`)
```

```
(PARTITION p1 VALUES [('0000-01-01'), ('1993-01-01')),
PARTITION p2 VALUES [('1993-01-01'), ('1994-01-01')),
PARTITION p3 VALUES [('1994-01-01'), ('1995-01-01')),
PARTITION p4 VALUES [('1995-01-01'), ('1996-01-01')),
PARTITION p5 VALUES [('1996-01-01'), ('1997-01-01')),
PARTITION p6 VALUES [('1997-01-01'), ('1998-01-01')),
PARTITION p7 VALUES [('1998-01-01'), ('1999-01-01')))
DISTRIBUTED BY HASH(`lo_orderkey`)
PROPERTIES
(
    "replication_num" = "3",
    "colocate_with" = "groupxx1",
    "storage_format" = "DEFAULT",
    "enable_persistent_index" = "true",
    "compression" = "LZ4"
);
```

Следующие два запроса, Q1 и Q2, к таблице `lineorder_flat` семантически эквивалентны после их обработки следующим образом:

1. Перестановка выходных столбцов оператора SELECT.
2. Перестановка выходных столбцов предложения GROUP BY.
3. Удаление выходных столбцов предложения ORDER BY.
4. Перестановка предикатов в предложении WHERE.
5. Добавление `PartitionColumnRangePredicate`.

- Q1

```
SELECT sum(lo_revenue), year(lo_orderdate) AS year,p_brand
FROM lineorder_flat
WHERE p_category = 'MFGR#12' AND s_region = 'AMERICA'
GROUP BY year,p_brand
ORDER BY year,p_brand;
```

- Q2

```
SELECT year(lo_orderdate) AS year, p_brand, sum(lo_revenue)
FROM lineorder_flat
WHERE s_region = 'AMERICA' AND p_category = 'MFGR#12' AND
   lo_orderdate >= '1993-01-01' AND lo_orderdate <= '1993-12-31'
GROUP BY p_brand, year(lo_orderdate)
```

Семантическая эквивалентность оценивается на основе физических планов запросов. Поэтому буквальные различия в запросах не влияют на оценку семантической эквивалентности. Кроме того, константные выражения удаляются из запросов, а выражения `cast` удаляются во время оптимизации запросов. Поэтому эти выражения не влияют на оценку семантической эквивалентности. В-третьих, псевдонимы столбцов и отношений также не влияют на оценку семантической эквивалентности.

Запросы с перекрывающимися сканируемыми partition

Кэш запросов поддерживает разделение запросов на основе предикатов.

Разделение запросов на основе семантики предикатов помогает реализовать повторное использование частичных результатов вычислений. Когда запрос содержит предикат, ссылающийся на столбец партиционирования таблицы, и предикат указывает диапазон значений, Selena может разделить диапазон на несколько интервалов на основе партиционирования таблицы. Результаты вычислений из каждого отдельного интервала могут быть отдельно повторно использованы другими запросами.

Используйте следующую таблицу `t0` в качестве примера:

```
CREATE TABLE if not exists t0
(
    ts DATETIME NOT NULL,
    k0 VARCHAR(10) NOT NULL,
    k1 BIGINT NOT NULL,
    v1 DECIMAL64(7, 2) NOT NULL
)
ENGINE=OLAP
DUPLICATE KEY(`ts`, `k0`, `k1`)
COMMENT "OLAP"
PARTITION BY RANGE(ts)
(
  START ("2022-01-01 00:00:00") END ("2022-02-01 00:00:00") EVERY (INTERVAL
1 day)
)
DISTRIBUTED BY HASH(`ts`, `k0`, `k1`)
PROPERTIES
(
    "replication_num" = "3",
    "storage_format" = "default"
);
```

Таблица `t0` партиционирована по дням, и столбец `ts` является столбцом партиционирования таблицы. Среди следующих четырёх запросов Q2, Q3 и Q4 могут повторно использовать части результатов вычислений, кэшированных для Q1:

- Q1

  ```
  SELECT date_trunc('day', ts) as day, sum(v0)
  FROM t0
  WHERE ts BETWEEN '2022-01-02 12:30:00' AND '2022-01-14 23:59:59'
  GROUP BY day;
  ```

  Диапазон значений, указанный предикатом `ts between '2022-01-02 12:30:00' and '2022-01-14 23:59:59'` запроса Q1, может быть разделён на следующие интервалы:

```
1. [2022-01-02 12:30:00, 2022-01-03 00:00:00),
2. [2022-01-03 00:00:00, 2022-01-04 00:00:00),
3. [2022-01-04 00:00:00, 2022-01-05 00:00:00),
...
12. [2022-01-13 00:00:00, 2022-01-14 00:00:00),
13. [2022-01-14 00:00:00, 2022-01-15 00:00:00),
```

- Q2

```sql
SELECT date_trunc('day', ts) as day, sum(v0)
FROM t0
WHERE ts >= '2022-01-02 12:30:00' AND  ts < '2022-01-05 00:00:00'
GROUP BY day;
```

Q2 может повторно использовать результаты вычислений в следующих интервалах Q1:

```
1. [2022-01-02 12:30:00, 2022-01-03 00:00:00),
2. [2022-01-03 00:00:00, 2022-01-04 00:00:00),
3. [2022-01-04 00:00:00, 2022-01-05 00:00:00),
```

- Q3

```sql
SELECT date_trunc('day', ts) as day, sum(v0)
FROM t0
WHERE ts >= '2022-01-01 12:30:00' AND  ts <= '2022-01-10 12:00:00'
GROUP BY day;
```

Q3 может повторно использовать результаты вычислений в следующих интервалах Q1:

```
2. [2022-01-03 00:00:00, 2022-01-04 00:00:00),
3. [2022-01-04 00:00:00, 2022-01-05 00:00:00),
...
8. [2022-01-09 00:00:00, 2022-01-10 00:00:00),
```

- Q4

```
SELECT date_trunc('day', ts) as day, sum(v0)
FROM t0
WHERE ts BETWEEN '2022-01-02 12:30:00' and '2022-01-02 23:59:59'
GROUP BY day;
```

Q4 может повторно использовать результаты вычислений в следующих интервалах Q1:

```
1. [2022-01-02 12:30:00, 2022-01-03 00:00:00),
```

Поддержка повторного использования частичных результатов вычислений варьируется в зависимости от используемой политики партиционирования, как описано в следующей таблице.

Политика партиционирования	Поддержка повторного использования частичных результатов вычислений
Без партиционирования	Не поддерживается
Многостолбцовое партиционирование	Не поддерживается **ПРИМЕЧАНИЕ** Эта функция может быть поддержана в будущем.
Одностолбцовое партиционирование	Поддерживается

Запросы к данным только с добавлением

Кэш запросов поддерживает многоверсионное кэширование.

По мере выполнения загрузок данных генерируются новые версии tablet. Следовательно, кэшированные результаты вычислений, которые генерируются из предыдущих версий tablet, становятся устаревшими и отстают от последних версий tablet. В этой ситуации механизм многоверсионного кэширования пытается объединить устаревшие результаты, сохранённые в кэше запросов, и инкрементальные версии tablet, хранящиеся на диске, в окончательные результаты tablet, чтобы новые запросы могли содержать последние версии tablet. Многоверсионное кэширование ограничено типами таблиц, типами запросов и типами обновления данных.

Поддержка многоверсионного кэширования варьируется в зависимости от типов таблиц и типов запросов, как описано в следующей таблице.

Тип таблицы	Тип запроса	Поддержка многоверсионного кэширования
Duplicate Key table	• Запросы к базовым таблицам • Запросы к синхронным материализованным представлениям	• Запросы к базовым таблицам: поддерживается во всех ситуациях, кроме случаев, когда инкрементальные версии tablet содержат записи об удалении данных. • Запросы к синхронным материализованным представлениям: поддерживается во всех ситуациях, кроме случаев, когда предложения GROUP BY, HAVING или WHERE запросов ссылаются на столбцы агрегации.
Aggregate table	Запросы к базовым таблицам или запросы к синхронным материализованным представлениям	Поддерживается во всех ситуациях, кроме следующих: схемы базовых таблиц содержат агрегатную функцию `replace`. Предложения GROUP BY, HAVING или WHERE запросов ссылаются на столбцы агрегации. Инкрементальные версии tablet содержат записи об удалении данных.
Unique Key table	Н/Д	Не поддерживается. Однако кэш запросов поддерживается.
Primary Key table	Н/Д	Не поддерживается. Однако кэш запросов поддерживается.

Влияние типов обновления данных на многоверсионное кэширование следующее:

- Удаление данных

  Многоверсионное кэширование не может работать, если инкрементальные версии tablet содержат операции удаления.

- Вставка данных

  - Если для tablet генерируется пустая версия, существующие данные tablet в кэше запросов остаются действительными и всё ещё могут быть извлечены.
  - Если для tablet генерируется непустая версия, существующие данные tablet в кэше запросов остаются действительными, но их версия отстаёт от последней версии tablet. В этой ситуации Selena читает инкрементальные данные, сгенерированные от версии существующих данных до последней версии tablet, объединяет существующие данные с инкрементальными данными и заполняет объединённые данные в кэш запросов.

- Изменения схемы и усечение tablet

  Если схема таблицы изменяется или конкретные tablet таблицы усекаются, для таблицы генерируются новые tablet. В результате существующие данные tablet таблицы в кэше запросов становятся недействительными.

Метрики

Профили запросов, для которых работает кэш запросов, содержат статистику `CacheOperator`.

В исходном плане запроса, если pipeline содержит `OlapScanOperator`, имена `OlapScanOperator` и операторов агрегации имеют префикс `ML_`, чтобы обозначить, что pipeline использует `MultilaneOperator` для выполнения вычислений по tablet. `CacheOperator` вставляется перед `ML_CONJUGATE_AGGREGATE` для обработки логики, управляющей работой кэша запросов в режимах Passthrough, Populate и Probe. Профиль запроса содержит следующие метрики `CacheOperator`, которые помогают понять использование кэша запросов.

Метрика	Описание
CachePassthroughBytes	Количество байтов, сгенерированных в режиме Passthrough.
CachePassthroughChunkNum	Количество chunks, сгенерированных в режиме Passthrough.
CachePassthroughRowNum	Количество строк, сгенерированных в режиме Passthrough.
CachePassthroughTabletNum	Количество tablet, сгенерированных в режиме Passthrough.
CachePassthroughTime:	Количество времени вычислений в режиме Passthrough.
CachePopulateBytes	Количество байтов, сгенерированных в режиме Populate.
CachePopulateChunkNum	Количество chunks, сгенерированных в режиме Populate.
CachePopulateRowNum	Количество строк, сгенерированных в режиме Populate.
CachePopulateTabletNum	Количество tablet, сгенерированных в режиме Populate.
CachePopulateTime	Количество времени вычислений в режиме Populate.
CacheProbeBytes	Количество байтов, сгенерированных для попаданий в кэш в режиме Probe.
CacheProbeChunkNum	Количество chunks, сгенерированных для попаданий в кэш в режиме Probe.
CacheProbeRowNum	Количество строк, сгенерированных для попаданий в кэш в режиме Probe.
CacheProbeTabletNum	Количество tablet, сгенерированных для попаданий в кэш в режиме Probe.
CacheProbeTime	Количество времени вычислений в режиме Probe.

Метрики `CachePopulate`*`XXX`* предоставляют статистику о промахах кэша запросов, для которых кэш запросов обновляется.

Метрики `CachePassthrough` *XXX* предоставляют статистику о промахах кэша запросов, для которых кэш запросов не обновляется, потому что размер результатов вычислений по tablet большой.

Метрики `CacheProbe` *XXX* предоставляют статистику о попаданиях в кэш запросов.

В механизме многоверсионного кэширования метрики `CachePopulate` и метрики `CacheProbe` могут содержать одну и ту же статистику tablet, а метрики `CachePassthrough` и метрики `CacheProbe` также могут содержать одну и ту же статистику tablet. Например, когда Selena вычисляет данные каждого tablet, она попадает в результаты вычислений, сгенерированные на исторической версии tablet. В этой ситуации Selena читает инкрементальные данные, сгенерированные от исторической версии до последней версии tablet, вычисляет данные и объединяет инкрементальные данные с кэшированными данными. Если размер результатов вычислений, сгенерированных после объединения, не превышает порог, указанный параметром `query_cache_entry_max_bytes` или `query_cache_entry_max_rows`, статистика tablet собирается в метрики `CachePopulate`. В противном случае статистика tablet собирается в метрики `CachePassthrough`.

Операции RESTful API

- `metrics |grep query_cache`

  Эта операция API используется для запроса метрик, связанных с кэшем запросов.

  ```
  curl -s  http://<be_host>:<be_http_port>/metrics |grep query_cache

  # TYPE selena_be_query_cache_capacity gauge
  selena_be_query_cache_capacity 536870912
  # TYPE selena_be_query_cache_hit_count gauge
  selena_be_query_cache_hit_count 5084393
  # TYPE selena_be_query_cache_hit_ratio gauge
  selena_be_query_cache_hit_ratio 0.984098
  # TYPE selena_be_query_cache_lookup_count gauge
  selena_be_query_cache_lookup_count 5166553
  # TYPE selena_be_query_cache_usage gauge
  selena_be_query_cache_usage 0
  # TYPE selena_be_query_cache_usage_ratio gauge
  selena_be_query_cache_usage_ratio 0.000000
  ```

- `api/query_cache/stat`

Эта операция API используется для запроса использования кэша запросов.

```
curl  http://<be_host>:<be_http_port>/api/query_cache/stat
{
    "capacity": 536870912,
    "usage": 0,
    "usage_ratio": 0.0,
    "lookup_count": 5025124,
    "hit_count": 4943720,
    "hit_ratio": 0.983800598751394
}
```

- `api/query_cache/invalidate_all`

  Эта операция API используется для очистки кэша запросов.

```
curl  -XPUT http://<be_host>:
<be_http_port>/api/query_cache/invalidate_all

{
    "status": "OK"
}
```

Параметры в предыдущих операциях API следующие:

- `be_host`: IP-адрес узла, на котором находится BE.
- `be_http_port`: номер HTTP-порта узла, на котором находится BE.

Предостережения

- Selena необходимо заполнить кэш запросов результатами вычислений запросов, которые инициируются впервые. В результате производительность запросов может быть немного ниже ожидаемой, и задержка запросов увеличивается.
- Если вы настроите большой размер кэша запросов, объём памяти, который может быть предоставлен для оценки запросов на BE, уменьшается. Рекомендуется, чтобы размер кэша запросов не превышал 1/6 объёма памяти, предоставляемой для оценки запросов.
- Если количество tablet, которые необходимо обработать, меньше значения `pipeline_dop`, кэш запросов не работает. Чтобы кэш запросов работал, вы можете

установить `pipeline_dop` в меньшее значение, например `1`. Начиная с версии v1.5.2, Selena адаптивно регулирует этот параметр на основе параллелизма запросов.

Примеры

Набор данных

1. Войдите в свой cluster Selena, перейдите в целевую базу данных и выполните следующую команду для создания таблицы с именем `t0`:

```
CREATE TABLE t0
(
      `ts` datetime NOT NULL COMMENT "",
      `k0` varchar(10) NOT NULL COMMENT "",
      `k1` char(6) NOT NULL COMMENT "",
      `v0` bigint(20) NOT NULL COMMENT "",
      `v1` decimal64(7, 2) NOT NULL COMMENT ""
)
ENGINE=OLAP
DUPLICATE KEY(`ts`, `k0`, `k1`)
COMMENT "OLAP"
PARTITION BY RANGE(`ts`)
(
    START ("2022-01-01 00:00:00") END ("2022-02-01 00:00:00") EVERY
(INTERVAL 1 DAY)
)
DISTRIBUTED BY HASH(`ts`, `k0`, `k1`)
PROPERTIES
(
    "replication_num" = "3",
    "storage_format" = "DEFAULT",
    "enable_persistent_index" = "true"
);
```

2. Вставьте следующие записи данных в `t0`:

```
INSERT INTO t0
VALUES
    ('2022-01-11 20:42:26', 'n4AGcEqYp', 'hhbawx', '799393174109549',
'8029.42'),
    ('2022-01-27 18:17:59', 'i66lt', 'mtrtzf', '100400167',
'10000.88'),
    ...
```

Примеры запросов

Статистика метрик, связанных с кэшем запросов, в этом разделе является примерами и предназначена только для справки.

Кэш запросов работает для локальных агрегаций на стадии 1

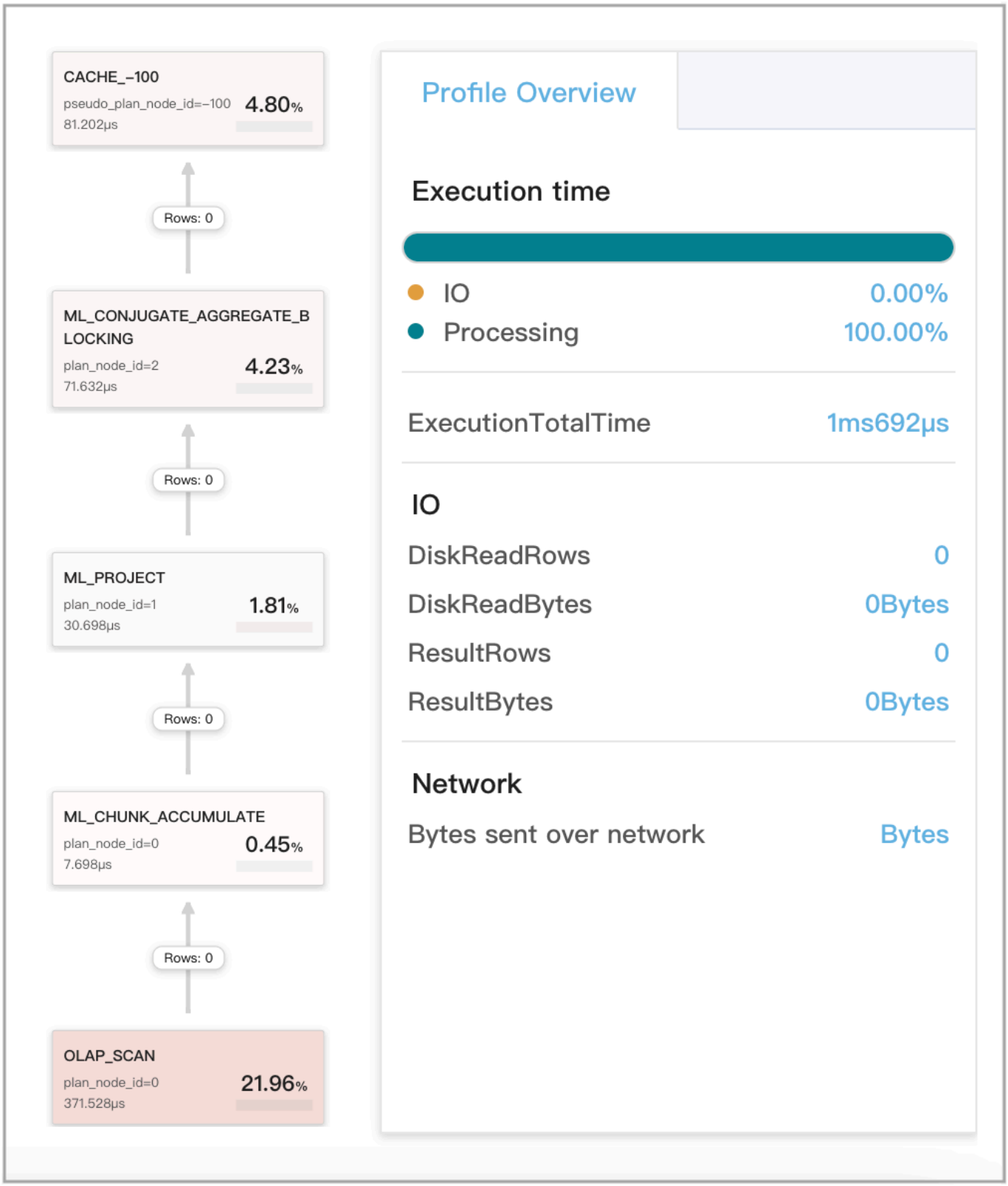
Это включает три ситуации:

- Запрос обращается только к одному tablet.
- Запрос обращается к нескольким tablet из нескольких partition таблицы, которая сама является colocated-группой, и данные не нужно перетасовывать для агрегаций.
- Запрос обращается к нескольким tablet из одного partition таблицы, и данные не нужно перетасовывать для агрегаций.

Пример запроса:

```
SELECT
    date_trunc('hour', ts) AS hour,
    k0,
    sum(v1) AS __c_0
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59'
GROUP BY
    date_trunc('hour', ts),
    k0
```

На следующем рисунке показаны метрики, связанные с кэшем запросов, в профиле запроса.



Кэш запросов не работает для удалённых агрегаций на стадии 1

Когда агрегации на нескольких tablet принудительно выполняются на стадии 1, данные сначала перетасовываются, а затем агрегируются.

Пример запроса:

```
SET new_planner_agg_stage = 1;

SELECT
    date_trunc('hour', ts) AS hour,
    v0 % 2 AS is_odd,
    sum(v1) AS __c_0
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59'
GROUP BY
    date_trunc('hour', ts),
    is_odd
```

Кэш запросов работает для локальных агрегаций на стадии 2

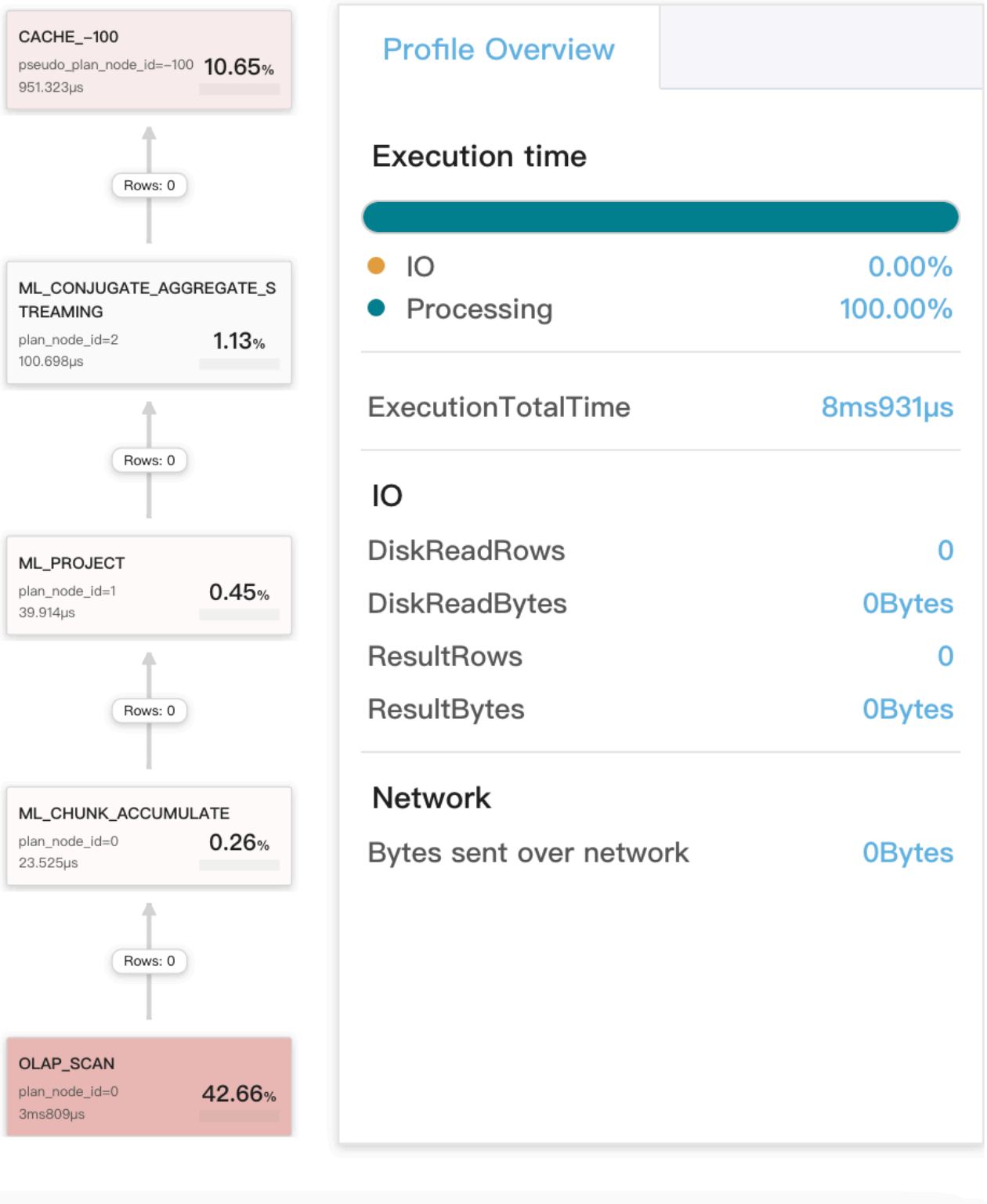
Это включает три ситуации:

- Агрегации на стадии 2 запроса компилируются для сравнения данных одного типа. Первая агрегация — локальная. После завершения первой агрегации результаты, сгенерированные из первой агрегации, вычисляются для выполнения второй агрегации, которая является глобальной.
- Запрос является запросом SELECT DISTINCT.
- Запрос включает одну из следующих агрегатных функций DISTINCT: `sum(distinct)`, `count(distinct)` и `avg(distinct)`. В большинстве случаев агрегации выполняются на стадии 3 или 4 для такого запроса. Однако вы можете выполнить `set new_planner_agg_stage = 1`, чтобы принудительно выполнить агрегации на стадии 2 для запроса. Если запрос содержит `avg(distinct)` и вы хотите выполнить агрегации на стадии, вам также нужно выполнить `set cbo_cte_reuse = false`, чтобы отключить CTE-оптимизации.

Пример запроса:

```
SELECT
    date_trunc('hour', ts) AS hour,
    v0 % 2 AS is_odd,
    sum(v1) AS __c_0
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59'
GROUP BY
    date_trunc('hour', ts),
    is_odd
```

На следующем рисунке показаны метрики, связанные с кэшем запросов, в профиле запроса.



Кэш запросов работает для локальных агрегаций на стадии 3

Запрос является агрегатным запросом GROUP BY, который включает одну агрегатную функцию DISTINCT.

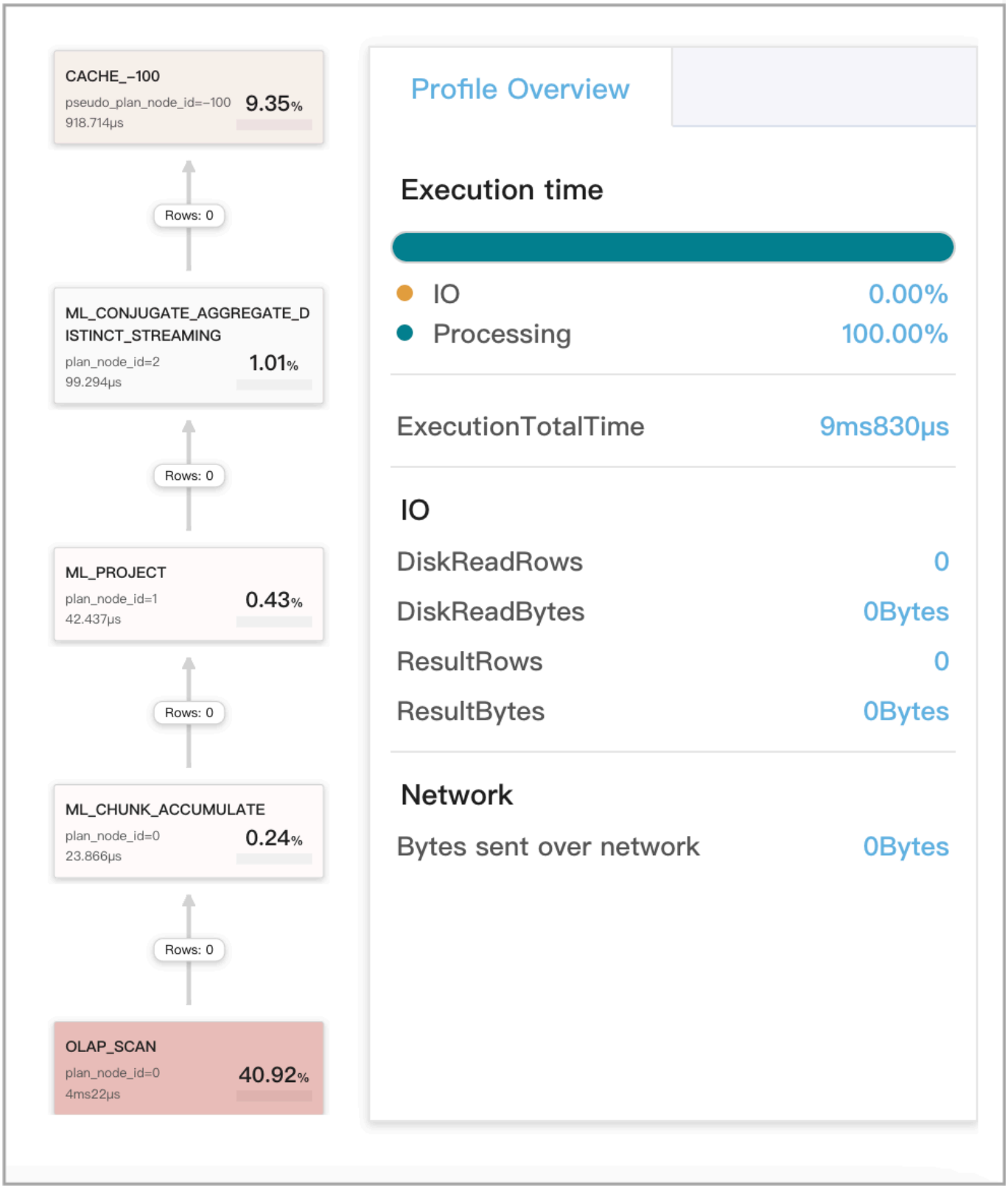
Поддерживаемые агрегатные функции DISTINCT: `sum(distinct)`, `count(distinct)` и `avg(distinct)`.

> **ВНИМАНИЕ**
>
> Если запрос включает `avg(distinct)`, вам также нужно выполнить `set cbo_cte_reuse = false`, чтобы отключить CTE-оптимизации.

Пример запроса:

```
SELECT
    date_trunc('hour', ts) AS hour,
    v0 % 2 AS is_odd,
    sum(distinct v1) AS __c_0
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59'
GROUP BY
    date_trunc('hour', ts),
    is_odd;
```

На следующем рисунке показаны метрики, связанные с кэшем запросов, в профиле запроса.



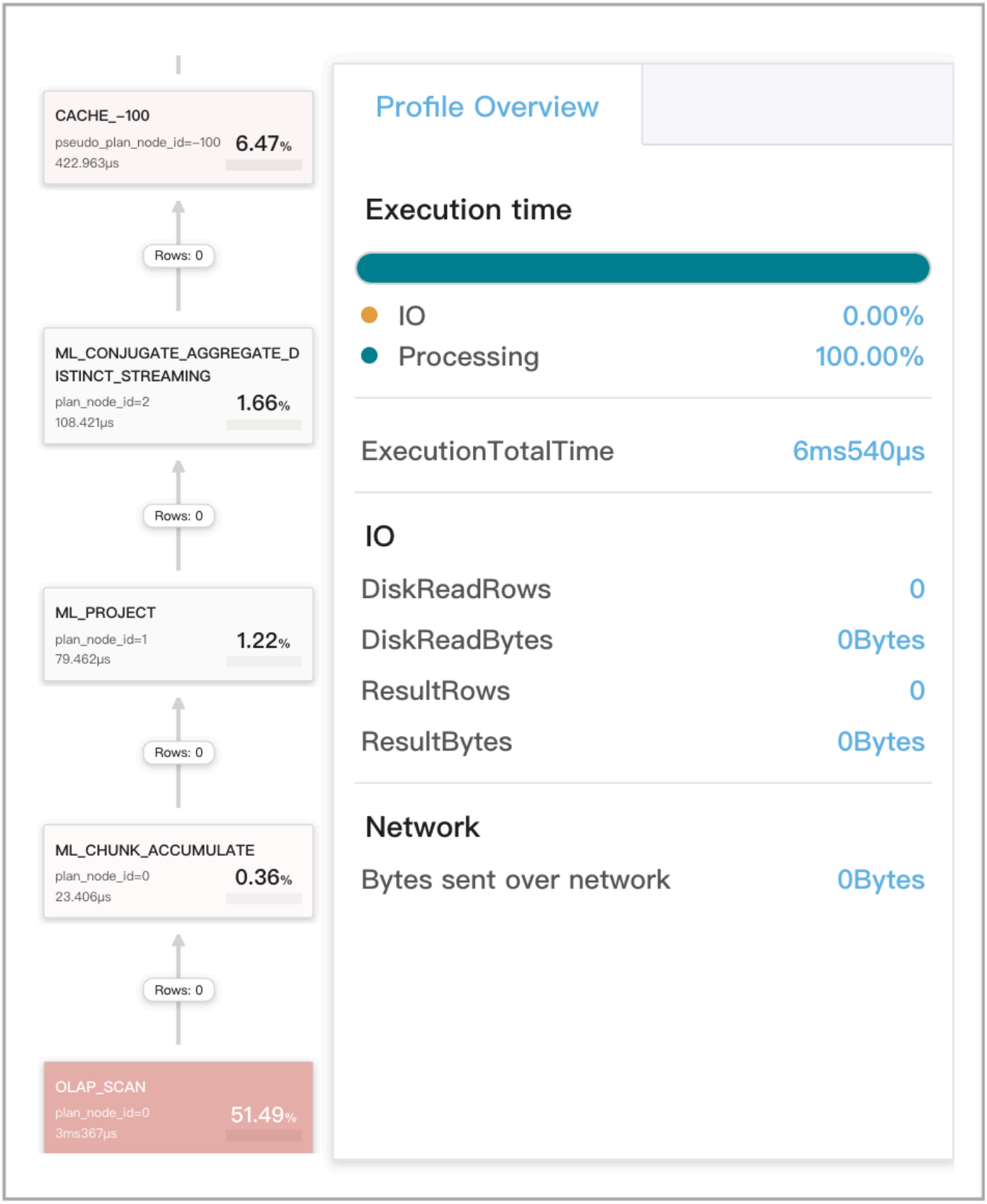
Кэш запросов работает для локальных агрегаций на стадии 4

Запрос является агрегатным запросом без GROUP BY, который включает одну агрегатную функцию DISTINCT. Такие запросы включают классические запросы на удаление дублирующих данных.

Пример запроса:

```
SELECT
    count(distinct v1) AS __c_0
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59'
```

На следующем рисунке показаны метрики, связанные с кэшем запросов, в профиле запроса.



Кэшированные результаты повторно используются для двух запросов, первые агрегации которых семантически эквивалентны

Используйте следующие два запроса, Q1 и Q2, в качестве примера. Q1 и Q2 оба включают несколько агрегаций, но их первые агрегации семантически эквивалентны.

Поэтому Q1 и Q2 оцениваются как семантически эквивалентные и могут повторно использовать результаты вычислений друг друга, сохранённые в кэше запросов.

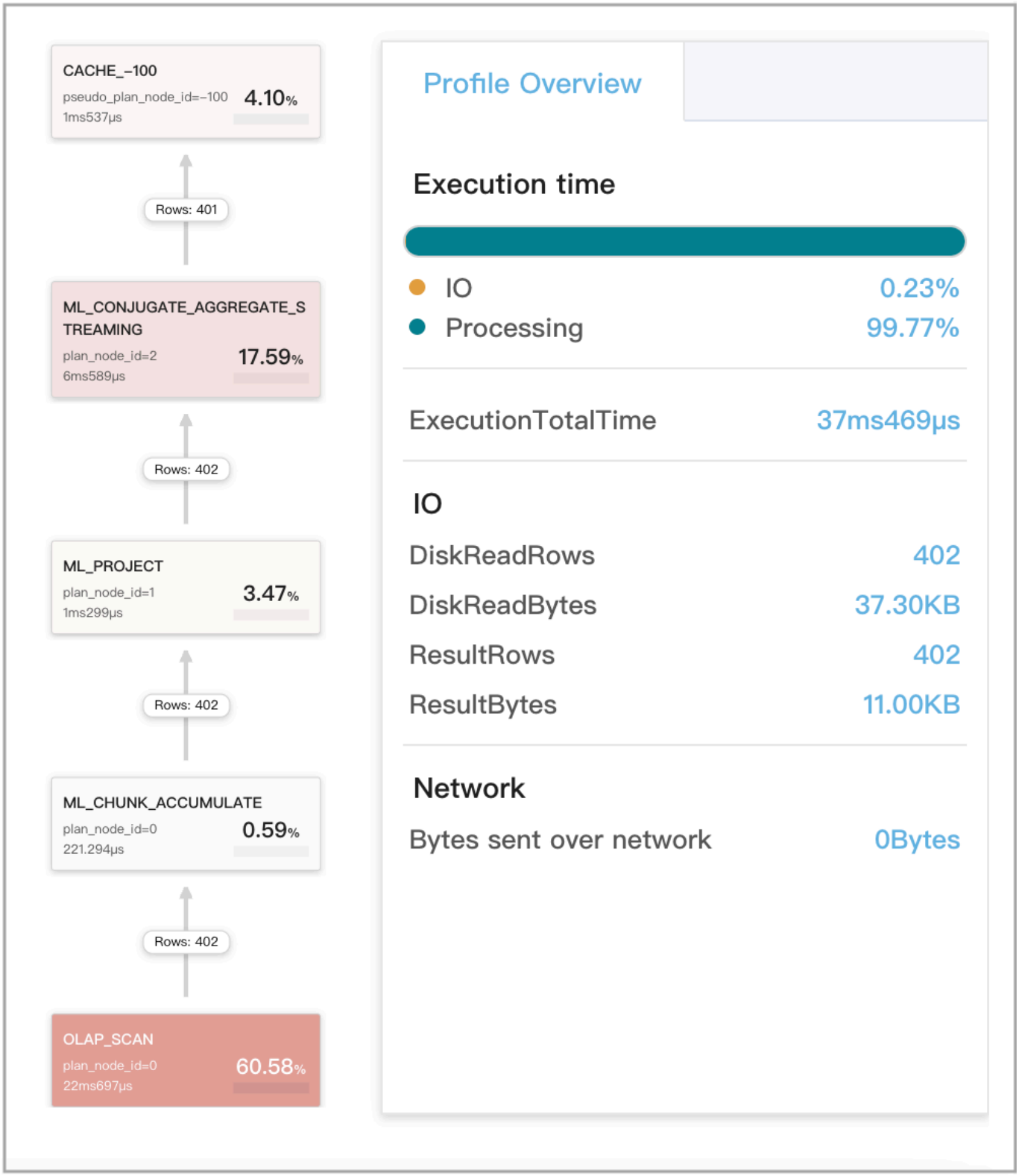
- Q1

```
SELECT
    (
        ifnull(sum(murmur_hash3_32(hour)), 0) +
ifnull(sum(murmur_hash3_32(k0)), 0) +
ifnull(sum(murmur_hash3_32(__c_0)), 0)
      ) AS fingerprint
FROM
    (
        SELECT
            date_trunc('hour', ts) AS hour,
            k0,
            sum(v1) AS __c_0
        FROM
            t0
        WHERE
            ts between '2022-01-03 00:00:00'
            and '2022-01-03 23:59:59'
        GROUP BY
            date_trunc('hour', ts),
            k0
    ) AS t;
```

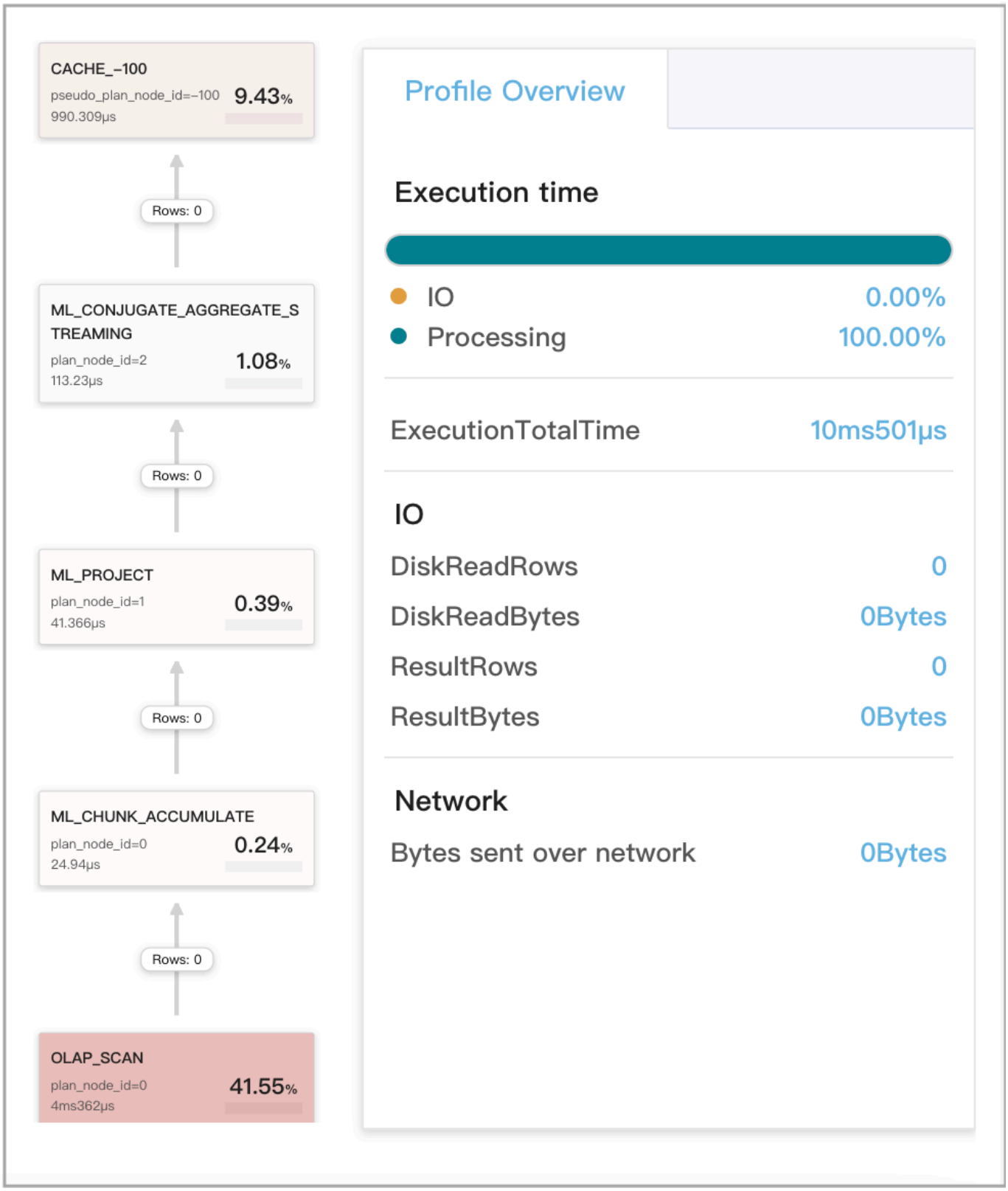
- Q2

```
SELECT
    date_trunc('hour', ts) AS hour,
    k0,
    sum(v1) AS __c_0
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59'
GROUP BY
    date_trunc('hour', ts),
    k0
```

На следующем рисунке показаны метрики `CachePopulate` для Q1.



На следующем рисунке показаны метрики `CacheProbe` для Q2.

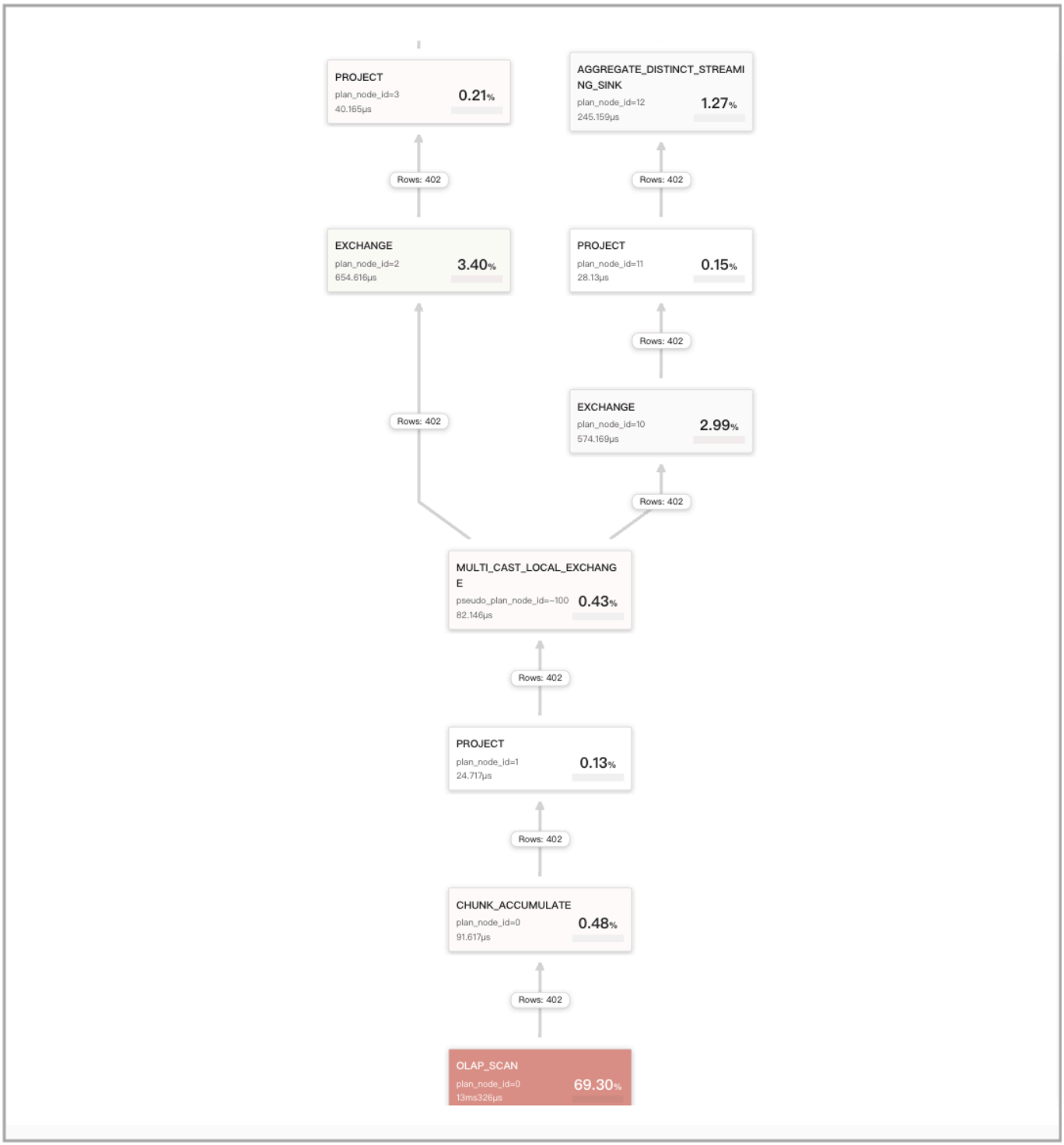


Кэш запросов не работает для запросов DISTINCT, для которых включены CTE-оптимизации

После выполнения `set cbo_cte_reuse = true` для включения CTE-оптимизаций результаты вычислений для определённых запросов, включающих агрегатные функции DISTINCT, не могут быть кэшированы. Несколько примеров:

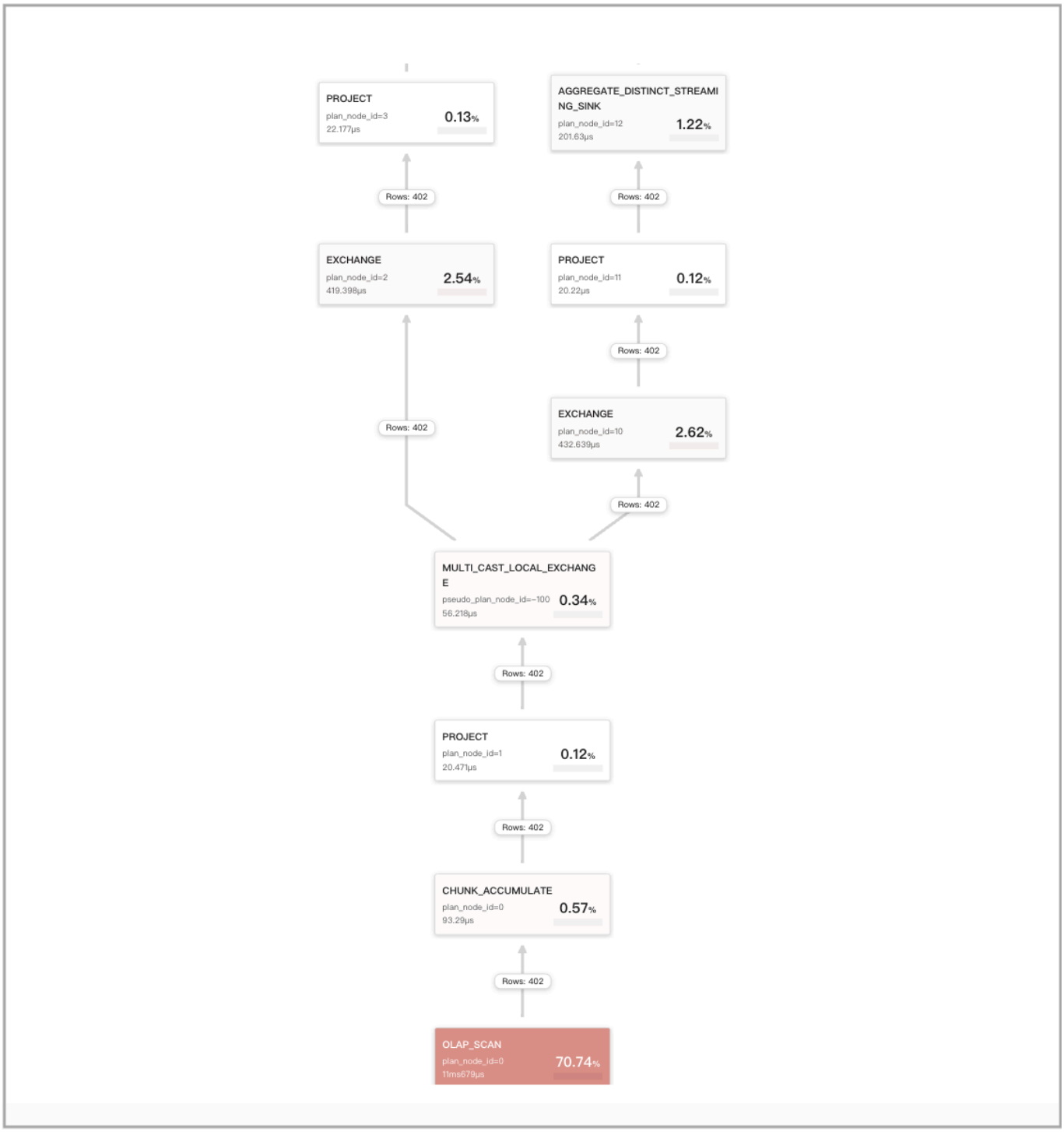
- Запрос содержит одну агрегатную функцию DISTINCT `avg(distinct)`:

```
SELECT
    avg(distinct v1) AS __c_0
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59';
```



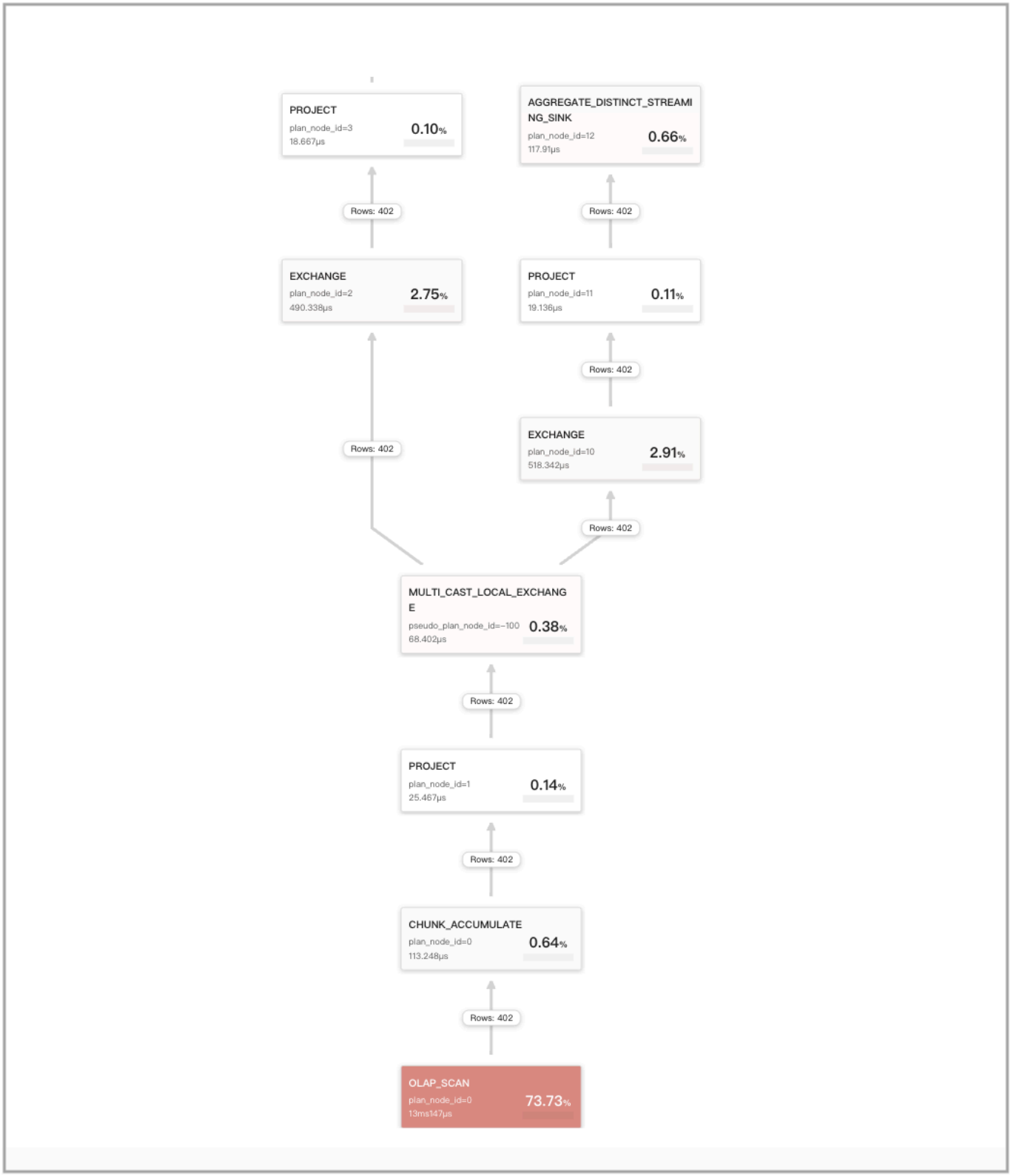
- Запрос содержит несколько агрегатных функций DISTINCT, ссылающихся на один и тот же столбец:

```sql
SELECT
    avg(distinct v1) AS __c_0,
    sum(distinct v1) AS __c_1,
    count(distinct v1) AS __c_2
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59';
```



- Запрос содержит несколько агрегатных функций DISTINCT, каждая из которых ссылается на разный столбец:

```sql
SELECT
    sum(distinct v1) AS __c_1,
    count(distinct v0) AS __c_2
FROM
    t0
WHERE
    ts between '2022-01-03 00:00:00'
    and '2022-01-03 23:59:59';
```



Лучшие практики

При создании таблицы укажите разумное описание partition и разумный метод распределения, включая:

- Выберите один столбец типа DATE в качестве столбца партиционирования. Если таблица содержит более одного столбца типа DATE, выберите столбец, значения которого продвигаются вперёд по мере инкрементального приёма данных и который используется для определения интересующих временных диапазонов запросов.
- Выберите правильную ширину partition. Данные, полученные недавно, могут изменять последние partition таблицы. Поэтому записи кэша, связанные с последними partition, нестабильны и склонны к аннулированию.
- Укажите количество bucket в несколько десятков в описании распределения оператора создания таблицы. Если количество bucket слишком мало, кэш запросов не может работать, когда количество tablet, которые необходимо обработать BE, меньше значения `pipeline_dop`.

6.9.2 Кэш данных (Data Cache)

Начиная с версий v1.5.2, Selena представила Data Cache для ускорения запросов в cluster с разделяемыми данными (shared-data), заменив File Cache в более ранних версиях. Data Cache загружает данные из удалённого хранилища блоками (порядка МБ) по мере необходимости, тогда как File Cache загружает целые файлы данных каждый раз в фоновом режиме, независимо от того, сколько строк данных фактически требуется.

По сравнению с File Cache, Data Cache имеет следующие преимущества:

- Меньше чтений из объектного хранилища, что означает меньшую стоимость доступа к объектному хранилищу (если ваше объектное хранилище взимает плату на основе частоты доступа).
- Меньше нагрузки на запись на локальные диски и использование CPU, следовательно, меньше влияния на другие задачи загрузки или запросов (поскольку фоновые потоки загрузки больше не нужны).
- Оптимизированная эффективность кэша (поскольку File Cache может загружать менее часто используемые данные в файле).
- Оптимизированный контроль над кэшированными данными, что позволяет избежать переполнения локальных дисков из-за избыточных данных, которые не удалось вытеснить File Cache.

Включение Data Cache

Начиная с версии v1.5.2, Selena использует единый экземпляр Data Cache для запросов к внешним catalog и облачным нативным таблицам (в cluster с разделяемыми данными).

Настройка Data Cache

Вы можете настроить Data Cache, используя следующие параметры конфигурации CN(BE):

Каталог кэша

- storage_root_path (В cluster с разделяемыми данными этот параметр используется для указания корневого пути, где хранятся кэшированные данные.)

Размер диска кэша

- datacache_disk_size
- starlet_star_cache_disk_size_percent

Размер диска кэша в cluster с разделяемыми данными принимает большее значение между `datacache_disk_size` и `starlet_star_cache_disk_size_percent`.

Просмотр статуса Data Cache

- Выполните следующий оператор для просмотра корневого пути, где хранятся кэшированные данные:

  ```
  SELECT * FROM information_schema.be_configs
  WHERE NAME LIKE "%storage_root_path%";
  ```

  Обычно кэшированные данные хранятся в подкаталоге `datacache/` вашего `storage_root_path`.

- Выполните следующий оператор для просмотра максимального процента хранилища, который может использовать Data Cache:

  ```
  SELECT * FROM information_schema.be_configs
  WHERE NAME LIKE "%starlet_star_cache_disk_size_percent% or
  %datacache_disk_size%";
  ```

Мониторинг Data Cache

Selena предоставляет различные метрики для мониторинга Data Cache.

Шаблоны панелей мониторинга

Вы можете скачать следующие шаблоны Grafana Dashboard в зависимости от вашей среды Selena:

- Шаблон панели мониторинга для Selena shared-data cluster на виртуальных машинах
- Шаблон панели мониторинга для Selena shared-data cluster на Kubernetes

Важные метрики

fslib read io_latency

Записывает задержку чтения Data Cache.

fslib write io_latency

Записывает задержку записи Data Cache.

fslib star cache meta memory size

Записывает приблизительное использование памяти Data Cache.

fslib star cache data disk size

Записывает фактическое использование диска Data Cache.

Отключение Data Cache

Чтобы отключить Data Cache, вам нужно добавить следующую конфигурацию в файл конфигурации CN **cn.conf** и перезапустить узлы CN:

```
datacache_enable = false
storage_root_path =
```

Очистка кэшированных данных

Вы можете очистить кэшированные данные в экстренных случаях. Это не повлияет на исходные данные в вашем удалённом хранилище.

Следуйте этим шагам для очистки кэшированных данных на узле CN:

1. Удалите подкаталог, в котором хранятся данные.

   Пример:

```
# Предположим, что `storage_root_path = /data/disk1;/data/disk2`
rm -rf /data/disk1/datacache/
rm -rf /data/disk2/datacache/
```

2. Перезапустите узел CN.

Примечания по использованию

- Если свойство `datacache.enable` установлено в `false` для облачной нативной таблицы, Data Cache не будет включён для этой таблицы.
- Если свойство `datacache.partition_duration` установлено на определённый временной диапазон, данные за пределами этого временного диапазона не будут кэшироваться.
- После понижения версии cluster с разделяемыми данными с v1.5.2 и более ранних версий, если вы хотите повторно использовать кэшированные данные в Data Cache, вам нужно вручную переименовать Blockfile в каталоге `starlet_cache`, изменив формат имени файла с `blockfile_{n}.{version}` на `blockfile_{n}`, то есть удалить суффикс с информацией о версии. Более поздние версии совместимы с форматом имени файла, поэтому вам не нужно выполнять эту операцию вручную. Вы можете изменить имя, запустив следующий shell-скрипт:

```
#!/bin/bash

# Замените <starlet_cache_path> на каталог Data Cache вашего cluster,
например, /usr/be/storage/starlet_cache.
starlet_cache_path="<starlet_cache_path>"

for blockfile in ${starlet_cache_path}/blockfile_*; do
    if [ -f "$blockfile" ]; then
        new_blockfile=$(echo "$blockfile" | cut -d'.' -f1)
        mv "$blockfile" "$new_blockfile"
    fi
done
```

Известные проблемы

Высокое использование памяти

- Описание: Когда cluster работает в условиях низкой нагрузки, общее использование памяти узла CN значительно превышает сумму использования памяти каждого

модуля.

- Идентификация: Если сумма `fslib star cache meta memory size` и `fslib star cache data memory size` занимает значительную долю от общего использования памяти узла CN, это может указывать на эту проблему.
- Затронутые версии: v1.5.2 и более ранние патч-версии, и v1.5.2 и более ранние патч-версии
- Исправленные версии: v1.5.2
- Решения:
  - Обновите cluster до исправленных версий.
  - Если вы не хотите обновлять cluster, вы можете очистить каталог `${storage_root_path}/starlet_cache/star_cache/meta` узлов CN и перезапустить узлы.

Вопросы и ответы

B1: Почему каталог Data Cache занимает гораздо больше места на диске (наблюдаемого через команды `du` и `ls`), чем фактический размер кэшированных данных?

Дисковое пространство, занимаемое Data Cache, представляет историческое пиковое использование и не связано с текущим фактическим размером кэшированных данных. Например, если кэшировано 100 ГБ данных, размер данных станет 200 ГБ после compaction. Затем, после сборки мусора (GC), размер данных был уменьшен до 100 ГБ. Однако дисковое пространство, занимаемое Data Cache, останется на пиковом уровне 200 ГБ, хотя фактические кэшированные данные внутри составляют 100 ГБ.

B2: Автоматически ли Data Cache вытесняет данные?

Нет. Data Cache вытесняет данные только при достижении лимита использования диска (`80%` дискового пространства по умолчанию). Процесс вытеснения не удаляет данные; он просто помечает дисковое пространство, хранящее старый кэш, как пустое. Новый кэш затем перезапишет старый кэш. Поэтому, даже если происходит вытеснение, использование диска не уменьшится и не повлияет на фактическое использование.

B3: Почему использование диска остаётся на настроенном максимальном уровне и не уменьшается?

См. B2. Механизм вытеснения Data Cache не удаляет кэшированные данные, а помечает старые данные как перезаписываемые. Следовательно, использование диска не уменьшится.

B4: Почему использование диска Data Cache остаётся на том же уровне после того, как я удалил таблицу и файлы таблицы удалены из удалённого хранилища?

Удаление таблицы не запускает удаление данных в Data Cache. Кэш удалённой таблицы будет постепенно вытеснен со временем на основе логики LRU (Least Recently Used) Data Cache, и это не влияет на фактическое использование.

B5: Почему фактическое использование диска достигает 90% или более, хотя настроено 80% ёмкости диска?

Использование диска Data Cache точное и не превысит настроенный лимит. Избыточное использование диска может быть вызвано:

- Файлами журналов, генерируемыми во время работы.
- Core-файлами, генерируемыми при сбоях CN.
- Постоянными индексами таблиц Primary Key (хранятся в `${storage_root_path}/persist/`).
- Смешанными развёртываниями экземпляров BE/CN/FE, использующих один и тот же диск.
- Проблемами с диском, например, используется файловая система ext3.

Вы можете выполнить `du -h . -d 1` в корневом каталоге диска или подкаталогах, чтобы проверить конкретные каталоги, занимающие пространство, а затем удалить неожиданные части. Вы также можете уменьшить лимит ёмкости диска Data Cache, настроив `starlet_star_cache_disk_size_percent`.

B6: Почему существует значительная разница в использовании диска Data Cache между узлами?

Невозможно обеспечить согласованное использование кэша между узлами из-за врождённых ограничений одноузлового кэширования. Пока это не влияет на задержку запросов, различия в использовании кэша допустимы. Это расхождение может быть вызвано:

- Различным временем добавления узлов в cluster.
- Различиями в количестве tablet, принадлежащих разным узлам.
- Различиями в размере данных, принадлежащих разным tablet.
- Различиями в ситуациях compaction и GC между узлами.
- Узлами, испытывающими сбои или проблемы нехватки памяти (OOM).

В итоге, различия в использовании кэша зависят от множества факторов.

6.10 Computing distinct value count

6.10.1 Bitmap для Count Distinct

Использование bitmap для вычисления количества уникальных значений.

Bitmap — полезный инструмент для вычисления количества уникальных значений в массиве. Этот метод занимает меньше места для хранения и может ускорить вычисления по сравнению с традиционным Count Distinct. Предположим, есть массив с именем A с диапазоном значений [0, n). Используя bitmap размером (n+7)/8 байт, можно вычислить количество уникальных элементов в массиве. Для этого инициализируйте все биты значением 0, установите значения элементов как индексы битов, а затем установите все биты в 1. Количество единиц в bitmap — это количество уникальных элементов в массиве.

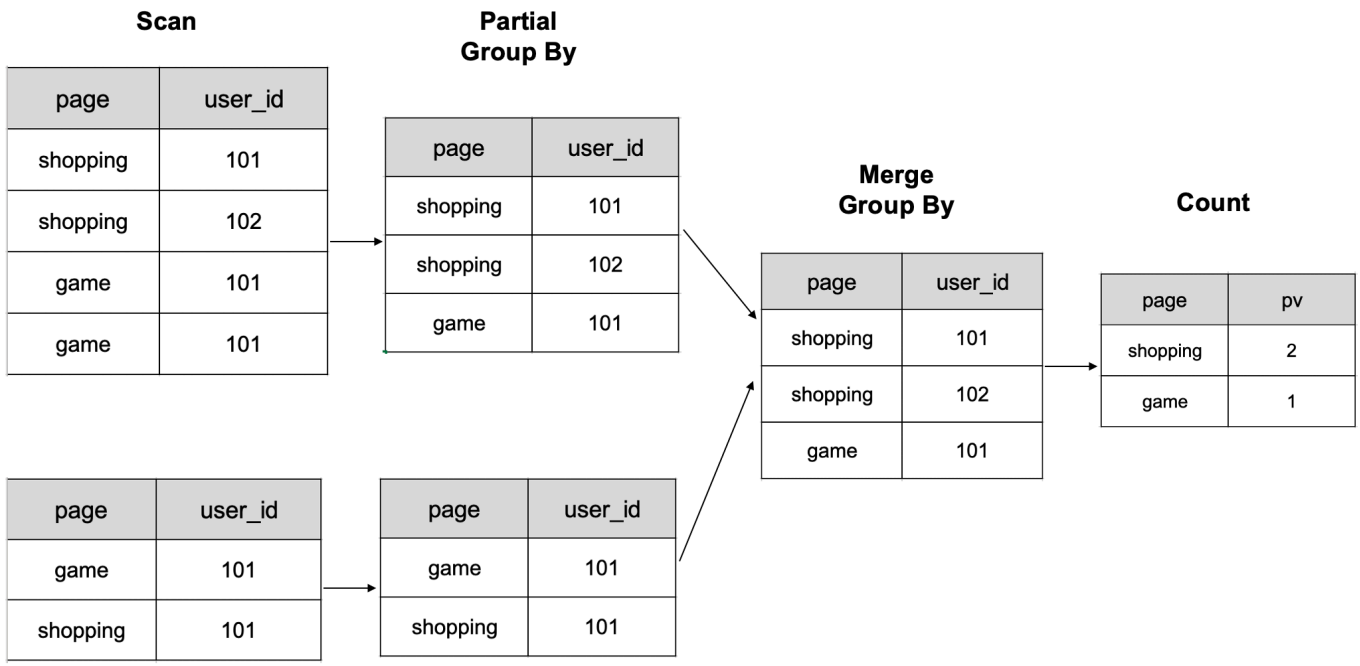
Традиционный Count Distinct

Архитектура MPP базы данных позволяет сохранять детализированные данные при использовании Count Distinct. Однако функция Count Distinct требует множественных операций shuffle данных во время обработки запроса, что потребляет больше ресурсов и приводит к линейному снижению производительности по мере увеличения объёма данных.

Следующий сценарий рассчитывает UV на основе детализированных данных в таблице (`dt`, `page`, `user_id`).

dt	page	user_id
20191206	game	101
20191206	shopping	102
20191206	game	101
20191206	shopping	101
20191206	game	101
20191206	shopping	101

Данные вычисляются согласно следующему рисунку. Сначала данные группируются по столбцам `page` и `user_id`, затем подсчитывается обработанный результат.



> **ПРИМЕЧАНИЕ**
>
> На рисунке показана схема вычисления 6 строк данных на двух узлах BE.

При работе с большими объёмами данных, требующими множественных операций shuffle, необходимые вычислительные ресурсы могут значительно увеличиться. Это замедляет запросы. Однако использование технологии Bitmap может помочь решить эту проблему и улучшить производительность запросов в таких сценариях.

Подсчёт `uv` с группировкой по `page`:

```
select page, count(distinct user_id) as uv from table group by page;
```

page	uv
game	1
shopping	2

Преимущества Count Distinct с Bitmap

Вы можете получить следующие преимущества от использования bitmap по сравнению с `COUNT(DISTINCT expr)`:

- Меньше места для хранения: Если вы используете bitmap для вычисления количества уникальных значений для данных `INT32`, требуемое пространство для хранения составляет всего 1/32 от `COUNT(DISTINCT expr)`. Для выполнения вычислений используются сжатые roaring bitmap, что ещё больше сокращает использование пространства для хранения по сравнению с традиционными bitmap.
- Более быстрые вычисления: Bitmap используют побитовые операции, что приводит к более быстрым вычислениям по сравнению с `COUNT(DISTINCT expr)`. Вычисление количества уникальных значений может обрабатываться параллельно, что приводит к дальнейшему улучшению производительности запросов.

Для реализации Roaring Bitmap см. соответствующую статью и реализацию.

Примечания по использованию

- Как bitmap-индексирование, так и bitmap Count Distinct используют технологию bitmap. Однако цель их введения и решаемые ими проблемы совершенно различны. Первое используется для фильтрации перечисляемых столбцов с низкой кардинальностью, а второе — для подсчёта количества уникальных элементов в столбцах значений строки данных.
- Вы можете определить столбец значений как `BITMAP` независимо от типа таблицы (Aggregate table, Duplicate Key table, Primary Key table или Unique Key table). Однако ключ сортировки таблицы не может иметь тип `BITMAP`.
- При создании таблицы вы можете определить столбец значений как `BITMAP` и агрегатную функцию как `BITMAP_UNION`.
- Вы можете использовать roaring bitmap только для вычисления количества уникальных значений для данных следующих типов: `TINYINT`, `SMALLINT`, `INT` и `BIGINT`. Для данных других типов необходимо построить глобальный словарь.

Count Distinct с Bitmap

Рассмотрим расчёт UV страниц в качестве примера.

1. Создайте Aggregate-таблицу со столбцом `BITMAP` `visit_users`, который использует агрегатную функцию `BITMAP_UNION`.

```
CREATE TABLE `page_uv` (
  `page_id` INT NOT NULL COMMENT 'ID страницы',
  `visit_date` datetime NOT NULL COMMENT 'время доступа',
  `visit_users` BITMAP BITMAP_UNION NOT NULL COMMENT 'ID пользователя'
) ENGINE=OLAP
AGGREGATE KEY(`page_id`, `visit_date`)
DISTRIBUTED BY HASH(`page_id`)
PROPERTIES (
  "replication_num" = "3",
  "storage_format" = "DEFAULT"
);
```

2. Загрузите данные в эту таблицу.

   Используйте `INSERT INTO` для загрузки данных:

```
INSERT INTO page_uv VALUES
(1, '2020-06-23 01:30:30', to_bitmap(13)),
(1, '2020-06-23 01:30:30', to_bitmap(23)),
(1, '2020-06-23 01:30:30', to_bitmap(33)),
(1, '2020-06-23 02:30:30', to_bitmap(13)),
(2, '2020-06-23 01:30:30', to_bitmap(23));
```

   После загрузки данных:

   - В строке `page_id = 1, visit_date = '2020-06-23 01:30:30'` поле `visit_users` содержит три элемента bitmap (13, 23, 33).
   - В строке `page_id = 1, visit_date = '2020-06-23 02:30:30'` поле `visit_users` содержит один элемент bitmap (13).
   - В строке `page_id = 2, visit_date = '2020-06-23 01:30:30'` поле `visit_users` содержит один элемент bitmap (23).

3. Рассчитайте UV страниц.

```
SELECT page_id, count(distinct visit_users)
FROM page_uv GROUP BY page_id;
```

```
+-----------+-----------------------------+
|  page_id  | count(DISTINCT `visit_users`)|
+-----------+-----------------------------+
|         1 |                           3 |
|         2 |                           1 |
+-----------+-----------------------------+
2 row in set (0.00 sec)
```

Глобальный словарь

В настоящее время механизм Count Distinct на основе Bitmap требует, чтобы входные данные были целыми числами. Если пользователю нужно использовать другие типы данных в качестве входных данных для Bitmap, то пользователю необходимо самостоятельно построить глобальный словарь для сопоставления других типов данных (например, строковых) с целочисленными типами. Существует несколько подходов к построению глобального словаря:

Глобальный словарь на основе таблицы Hive

Глобальный словарь в этой схеме сам по себе является таблицей Hive с двумя столбцами: один для исходных значений, другой для закодированных Int-значений. Шаги по генерации глобального словаря следующие:

1. Дедуплицируйте столбцы словаря таблицы фактов для создания временной таблицы
2. Выполните left join временной таблицы и глобального словаря, добавьте `new value` во временную таблицу.
3. Закодируйте `new value` и вставьте его в глобальный словарь.
4. Выполните left join таблицы фактов и обновлённого глобального словаря, замените элементы словаря идентификаторами.

Таким образом, глобальный словарь можно обновлять, а столбцы значений в таблице фактов можно заменять с помощью Spark или MR. По сравнению с глобальным словарём на основе trie-дерева, этот подход может быть распределённым, и глобальный словарь можно использовать повторно.

Однако следует учитывать несколько моментов: исходная таблица фактов читается несколько раз, и есть два соединения, которые потребляют много дополнительных ресурсов при вычислении глобального словаря.

Построение глобального словаря на основе trie-дерева

Пользователи также могут самостоятельно строить глобальные словари с использованием trie-деревьев (также известных как префиксные деревья или деревья словарей). Trie-дерево имеет общие префиксы для потомков узлов, что можно использовать для сокращения времени запроса и минимизации сравнений строк, и поэтому хорошо подходит для реализации словарного кодирования. Однако реализация trie-дерева нелегко распределяется и может создавать узкие места производительности при относительно большом объёме данных.

Построив глобальный словарь и преобразовав данные других типов в целочисленные данные, вы можете использовать Bitmap для выполнения точного анализа Count Distinct для столбцов нецелочисленных данных.

6.10.2 Использование HLL для приблизительного count distinct

Предпосылки

В реальных сценариях нагрузка на дедупликацию данных возрастает по мере увеличения объёма данных. Когда размер данных достигает определённого уровня, стоимость точной дедупликации становится относительно высокой. В этом случае пользователи обычно используют приблизительные алгоритмы для снижения вычислительной нагрузки. HyperLogLog (HLL), который будет представлен в этом разделе, является приблизительным алгоритмом дедупликации с превосходной пространственной сложностью O(mloglogn) и временной сложностью O(n). Более того, погрешность результата вычислений может контролироваться на уровне около 1%-10%, в зависимости от размера набора данных и используемой hash-функции.

Что такое HyperLogLog

HyperLogLog — это приблизительный алгоритм дедупликации, который потребляет очень мало места для хранения. Тип **HLL** используется для реализации алгоритма HyperLogLog. Он содержит промежуточные результаты вычисления HyperLogLog и может использоваться только как тип столбца-индикатора для таблиц данных.

Поскольку алгоритм HLL включает много математических знаний, мы проиллюстрируем его на практическом примере. Предположим, мы разрабатываем рандомизированный эксперимент A, то есть независимо повторяем подбрасывание монеты до первого

выпадения орла; записываем количество подбрасываний монеты до первого орла как случайную величину X, тогда:

- X=1, P(X=1)=1/2
- X=2, P(X=2)=1/4
- ...
- X=n, P(X=n)=$(1/2)^n$

Мы используем тест A для построения рандомизированного теста B, который заключается в N независимых повторениях теста A, генерируя N независимых одинаково распределённых случайных величин X_1, X_2, X_3, ..., X_N. Берём максимальное значение случайных величин как X_{max}. Используя оценку максимального правдоподобия, оценочное значение N равно $2^{X_{max}}$.

Теперь мы моделируем вышеуказанный эксперимент с использованием hash-функции на заданном наборе данных:

- Тест A: Вычисляем hash-значение элементов набора данных и преобразуем hash-значение в двоичное представление. Записываем появление bit=1, начиная с младшего бита двоичного представления.
- Тест B: Повторяем процесс теста A для элементов набора данных теста B. Обновляем максимальную позицию "m" первого появления бита 1 для каждого теста;
- Оцениваем количество неповторяющихся элементов в наборе данных как m^2.

Фактически, алгоритм HLL разделяет элементы на $K=2^k$ bucket на основе младших k битов hash-значения элемента. Подсчитывает максимальное значение первого появления бита 1, начиная с k+1-го бита, как m_1, m_2,..., m_k, и оценивает количество неповторяющихся элементов в bucket как 2^{m_1}, 2^{m_2},..., 2^{m_k}. Количество неповторяющихся элементов в наборе данных — это суммированное среднее количества bucket, умноженное на количество неповторяющихся элементов в bucket: $N = K(K/(2^{-m_1}+2^{-m_2},..., 2^{-m_K}))$.

HLL умножает поправочный коэффициент на результат оценки, чтобы сделать результат более точным.

Обратитесь к статье https://gist.github.com/avibryant/8275649 о реализации алгоритма дедупликации HLL с помощью SQL-операторов Selena:

```
SELECT floor((0.721 * 1024 * 1024) / (sum(pow(2, m * -1)) + 1024 -
count(*))) AS estimate
FROM(select(murmur_hash3_32(c2) & 1023) AS bucket,
     max((31 - CAST(log2(murmur_hash3_32(c2) & 2147483647) AS INT))) AS m
     FROM db0.table0
     GROUP BY bucket) bucket_values
```

Этот алгоритм дедуплицирует col2 таблицы db0.table0.

- Использует hash-функцию `murmur_hash3_32` для вычисления hash-значения col2 как 32-битного знакового целого числа.
- Использует 1024 bucket, поправочный коэффициент 0.721, и берёт младшие 10 битов hash-значения как индекс bucket.
- Игнорирует знаковый бит hash-значения, начинает со следующего старшего бита к младшему биту и определяет позицию первого появления бита 1.
- Группирует вычисленные hash-значения по bucket и использует агрегацию `MAX` для нахождения максимальной позиции первого появления бита 1 в bucket.
- Результат агрегации используется как подзапрос, и суммированное среднее всех оценок bucket умножается на количество bucket и поправочный коэффициент.
- Обратите внимание, что количество пустых bucket равно 1.

Вышеуказанный алгоритм имеет очень низкую погрешность при большом объёме данных.

Это основная идея алгоритма HLL. Если вас интересует, обратитесь к статье о HyperLogLog.

Как использовать HyperLogLog

1. Чтобы использовать дедупликацию HyperLogLog, вам нужно установить тип целевого столбца-индикатора как `HLL` и агрегатную функцию как `HLL_UNION` в операторе создания таблицы.
2. В настоящее время только Aggregate-таблица поддерживает HLL как тип столбца-индикатора.
3. При использовании `count distinct` для столбцов типа HLL Selena автоматически преобразует его в вычисление `HLL_UNION_AGG`.

Пример

Сначала создайте таблицу со столбцами **HLL**, где uv — агрегированный столбец, тип столбца — `HLL`, и агрегатная функция — HLL_UNION.

```
CREATE TABLE test(
        dt DATE,
        id INT,
        uv HLL HLL_UNION
)
DISTRIBUTED BY HASH(ID);
```

> - Примечание: При большом объёме данных лучше создать соответствующую rollup-таблицу для часто выполняемых HLL-запросов

Загрузите данные с помощью Stream Load:

```
curl --location-trusted -u <username>:<password> -H
"label:label_1600997542287" \
    -H "column_separator:," \
    -H "columns:dt,id,user_id, uv=hll_hash(user_id)" -T /root/test.csv
http://selena_be0:8040/api/db0/test/_stream_load
{
    "TxnId": 2504748,
    "Label": "label_1600997542287",
    "Status": "Success",
    "Message": "OK",
    "NumberTotalRows": 5,
    "NumberLoadedRows": 5,
    "NumberFilteredRows": 0,
    "NumberUnselectedRows": 0,
    "LoadBytes": 120,
    "LoadTimeMs": 46,
    "BeginTxnTimeMs": 0,
    "StreamLoadPutTimeMs": 1,
    "ReadDataTimeMs": 0,
    "WriteDataTimeMs": 29,
    "CommitAndPublishTimeMs": 14
}
```

Режим Broker Load:

```
LOAD LABEL test_db.label
 (
    DATA INFILE("hdfs://<hdfs_host>:
<hdfs_port>/user/selena/data/input/file")
    INTO TABLE `test`
    COLUMNS TERMINATED BY ","
    (dt, id, user_id)
    SET (
      uv = HLL_HASH(user_id)
    )
 );
```

Запрос данных

- Столбец HLL не позволяет напрямую запрашивать его исходное значение, используйте функцию HLL_UNION_AGG для запроса.
- Чтобы найти общий uv,

`SELECT HLL_UNION_AGG(uv) FROM test;`

Этот оператор эквивалентен

`SELECT COUNT(DISTINCT uv) FROM test;`

- Запрос uv за каждый день

`SELECT COUNT(DISTINCT uv) FROM test GROUP BY ID;`

Предостережения

Как выбрать между Bitmap и HLL? Если кардинальность набора данных составляет миллионы или десятки миллионов, и у вас есть несколько десятков машин, используйте `count distinct`. Если кардинальность составляет сотни миллионов и требуется точная дедупликация, используйте `Bitmap`; если приемлема приблизительная дедупликация, используйте тип `HLL`.

Bitmap поддерживает только TINYINT, SMALLINT, INT и BIGINT. Обратите внимание, что LARGEINT не поддерживается. Для других типов наборов данных, которые нужно дедуплицировать, необходимо построить словарь для сопоставления исходного типа с целочисленным типом. Построение словаря сложно и требует компромисса между

объёмом данных, частотой обновления, эффективностью запросов, хранением и другими факторами. HLL не требует словаря, но требует, чтобы соответствующий тип данных поддерживал hash-функцию. Даже в аналитической системе, которая не поддерживает HLL внутренне, всё ещё возможно использовать hash-функцию и SQL для реализации дедупликации HLL.

Для обычных столбцов пользователи могут использовать функцию NDV для приблизительной дедупликации. Эта функция возвращает приблизительную агрегацию результатов COUNT(DISTINCT col), и базовая реализация преобразует тип хранения данных в тип HyperLogLog для вычисления. Функция NDV потребляет много ресурсов при вычислении и поэтому не очень подходит для сценариев с высокой параллельностью.

Если вы хотите выполнять анализ поведения пользователей, вы можете рассмотреть IntersectCount или пользовательские UDAF.

6.11 Потоковая агрегация с сортировкой

Распространённые методы агрегации в системах баз данных включают hash-агрегацию и агрегацию с сортировкой.

Начиная с версии v1.5.2, Selena поддерживает **потоковую агрегацию с сортировкой (sorted streaming aggregate)**.

6.11.1 Принципы работы

Узлы агрегации (AGG) в основном отвечают за обработку GROUP BY и агрегатных функций.

Потоковая агрегация с сортировкой может группировать данные путём сравнения ключей GROUP BY в соответствии с последовательностью ключей, без необходимости создания hash-таблицы. Это эффективно снижает потребление памяти при агрегации. Для запросов с высокой кардинальностью агрегации потоковая агрегация с сортировкой улучшает производительность агрегации и уменьшает использование памяти.

Вы можете включить потоковую агрегацию с сортировкой, установив следующую переменную:

```
set enable_sort_aggregate=true;
```

6.11.2 Ограничения

- Cluster Selena с разделяемыми данными (shared-data) не поддерживают потоковую агрегацию с сортировкой.
- Ключи в GROUP BY должны следовать определённой последовательности. Например, если ключ сортировки — `k1, k2, k3`, тогда:
  - `GROUP BY k1` и `GROUP BY k1, k2` разрешены.
  - `GROUP BY k1, k3` не соответствует последовательности ключа сортировки. Поэтому потоковая агрегация с сортировкой не сможет работать с таким выражением.
- Выбранный partition должен быть единственным (поскольку один и тот же ключ может быть распределён на разных машинах в разных partition).
- Ключи GROUP BY должны иметь такое же распределение, как и bucket-ключи, указанные при создании таблицы. Например, если таблица имеет три столбца `k1, k2, k3`, bucket-ключ может быть `k1` или `k1, k2`.
  - Если bucket-ключ — `k1`, ключ `GROUP BY` может быть `k1`, `k1, k2` или `k1, k2, k3`.
  - Если bucket-ключ — `k1, k2`, ключ GROUP BY может быть `k1, k2` или `k1, k2, k3`.
  - Если планы запросов не соответствуют этому требованию, функция потоковой агрегации с сортировкой не будет работать, даже если она включена.
- Потоковая агрегация с сортировкой работает только для первой стадии агрегации (то есть под узлом AGG находится только один узел Scan).

6.11.3 Примеры

1. Создайте таблицу и вставьте данные.

```
CREATE TABLE `test_sorted_streaming_agg_basic`
(
    `id_int` int(11) NOT NULL COMMENT "",
    `id_string` varchar(100) NOT NULL COMMENT ""
)
ENGINE=OLAP
DUPLICATE KEY(`id_int`)COMMENT "OLAP"
DISTRIBUTED BY HASH(`id_int`)
PROPERTIES
("replication_num" = "3");

INSERT INTO test_sorted_streaming_agg_basic VALUES
(1, 'v1'),
(2, 'v2'),
(3, 'v3'),
(1, 'v4');
```

2. Включите потоковую агрегацию с сортировкой и используйте EXPLAIN для просмотра профиля SQL.

```
set enable_sort_aggregate = true;

explain costs select id_int, max(id_string)
from test_sorted_streaming_agg_basic
group by id_int;
```

6.11.4 Проверка включения потоковой агрегации с сортировкой

Просмотрите результаты `EXPLAIN costs`. Если поле `sorted streaming` в узле AGG имеет значение `true`, функция включена.

```
|
|
|   1:AGGREGATE (update finalize)
|
|   |  aggregate: max[([2: id_string, VARCHAR, false]); args: VARCHAR;
result: VARCHAR; args nullable: false; result nullable: true] |
|   |  group by: [1: id_int, INT, false]
|
|   |  sorted streaming: true
|
|   |  cardinality: 1
|
|   |  column statistics:
|
|   |  * id_int-->[-Infinity, Infinity, 0.0, 1.0, 1.0] UNKNOWN
|
|   |  * max-->[-Infinity, Infinity, 0.0, 1.0, 1.0] UNKNOWN
|
|   |
|
|   0:OlapScanNode
|
|      table: test_sorted_streaming_agg_basic, rollup:
test_sorted_streaming_agg_basic
|
|      preAggregation: on
|
|      partitionsRatio=1/1, tabletsRatio=10/10
|
|
tabletList=30672,30674,30676,30678,30680,30682,30684,30686,30688,30690
|
|      actualRows=0, avgRowSize=2.0
|
|      cardinality: 1
|
|      column statistics:
|
|      * id_int-->[-Infinity, Infinity, 0.0, 1.0, 1.0] UNKNOWN
|
|      * id_string-->[-Infinity, Infinity, 0.0, 1.0, 1.0] UNKNOWN
|
```

6.12 Ускорение COUNT(DISTINCT) и Join с помощью AUTO INCREMENT и глобального словаря

В этой теме описывается, как ускорить вычисление COUNT(DISTINCT) и операции Join с использованием столбцов AUTO INCREMENT и глобального словаря.

6.12.1 Сценарии использования

- **Сценарий 1**: Предположим, вам нужно выполнить точную дедупликацию на массивных данных (например, розничные заказы или заказы на доставку). Однако столбец для дедупликации имеет тип STRING, что может привести к неоптимальной производительности при подсчёте. Например, в таблице `orders` столбец `order_uuid`, представляющий идентификатор заказа, имеет тип STRING, обычно размером от 32 до 36 байт, генерируемый функцией `UUID()` или аналогичными. В этом случае COUNT(DISTINCT) по STRING-столбцу `order_uuid` с запросом вида `SELECT count(DISTINCT order_uuid) FROM orders WHERE create_date >= CURDATE();` может не обеспечить удовлетворительную производительность. Использование INTEGER-столбца для точной дедупликации значительно повысит производительность.

- **Сценарий 2**: Предположим, вы хотите ускорить точную дедупликацию в многомерном анализе с использованием bitmap-функций. Функция `bitmap_count()` требует INTEGER на входе, но если столбец для дедупликации имеет тип STRING, вам нужно использовать функцию `bitmap_hash()`. Это может привести к приблизительным, немного заниженным результатам дедупликации, а также может снизить производительность запросов и увеличить требования к хранилищу, поскольку INTEGER-значения, генерируемые bitmap_hash(), более рассеяны по сравнению с последовательно присвоенными INTEGER-значениями.

- **Сценарий 3**: Предположим, вам нужно запросить количество заказов с коротким промежутком времени между размещением и оплатой заказа, где время размещения и оплаты заказа могут храниться в разных таблицах, поддерживаемых разными бизнес-командами. Вам может потребоваться объединить эти таблицы по идентификатору заказа, а затем выполнить дедупликацию заказов. Например:

```
SELECT count(distinct order_uuid)
FROM orders_t1 as t1 JOIN orders_t2 as t2
    ON t1.order_uuid = t2.order_uuid
WHERE t2.payment_time - t1.create_time <= 3600
    AND create_date >= CURDATE();
```

Однако использование STRING-столбца `order_uuid` для соединений менее эффективно, чем использование INTEGER-столбца.

6.12.2 Подход к оптимизации

Для решения проблем в вышеуказанных сценариях подход к оптимизации включает загрузку данных заказов в целевую таблицу и установление сопоставления между STRING- и INTEGER-значениями. Последующий анализ запросов будет основан на INTEGER-столбце. Этот подход можно разделить на следующие этапы:

1. Этап 1: Создание глобального словаря и установление сопоставления между STRING- и INTEGER-значениями. В этом словаре ключевой столбец имеет тип STRING, а столбец значений — тип AUTO INCREMENT INTEGER. При загрузке данных система автоматически генерирует уникальный идентификатор для каждого STRING-значения, создавая сопоставление между STRING- и INTEGER-значениями.

2. Этап 2: Загрузка связи между данными заказов и глобальным словарём в целевую таблицу.

3. Этап 3: Использование INTEGER-столбца из целевой таблицы для точной дедупликации или соединений при последующем анализе запросов, что может значительно повысить производительность.

4. Этап 4: Для дальнейшей оптимизации производительности можно использовать bitmap-функции на INTEGER-столбце для ускорения точной дедупликации.

6.12.3 Решение

До версии v1.5.2 этап 2 мог быть реализован двумя способами:

- Использование внешней таблицы или внутренней таблицы в качестве промежуточной таблицы для соединения с таблицей словаря для получения соответствующего идентификатора словаря перед загрузкой.

- Использование таблицы Primary Key для загрузки данных, а затем использование оператора UPDATE с операцией JOIN для обновления идентификатора словаря. Однако этот процесс загрузки данных может быть неудобным и имеет много ограничений.

Начиная с версии v1.5.2, Selena представила функцию `dict_mapping()`, позволяющую определять столбец идентификатора словаря в целевой таблице как вычисляемый столбец с использованием выражения `dict_mapping()`. Последующие задачи загрузки данных обрабатываются как обычная загрузка данных, без необходимости использования операторов UPDATE с операциями JOIN для записи идентификаторов словаря. Во время загрузки данных система автоматически связывает исходную таблицу с таблицей словаря и вставляет соответствующий идентификатор словаря, что значительно упрощает процесс загрузки данных с глобальной таблицей словаря, независимо от типов таблиц и с поддержкой различных методов загрузки.

Бизнес-сценарий

В следующем примере используются два примера CSV-файлов, `batch1.csv` и `batch2.csv`, каждый из которых содержит два столбца: `id` и `order_uuid`.

- `batch1.csv`

```
1, a1
2, a2
3, a3
11, a1
11, a2
12, a1
```

- `batch2.csv`

```
1, a2
2, a2
3, a2
11, a2
12, a101
12, a102
13, a102
```

Процесс

Первый этап

Создайте таблицу глобального словаря и загрузите значения столбца идентификатора заказа из CSV-файлов для установления сопоставления между STRING- и INTEGER-значениями.

1. Создайте таблицу Primary Key для использования в качестве глобального словаря. Определите Primary Key `order_uuid` (тип STRING) и столбец значений `order_id_int` (тип AUTO INCREMENT INTEGER).

   > **К СВЕДЕНИЮ**
   >
   > Функция `dict_mapping` требует, чтобы таблица глобального словаря была таблицей Primary Key.

   ```
   CREATE TABLE dict (
       order_uuid STRING,
       order_id_int BIGINT AUTO_INCREMENT  -- Автоматически присваивает
   идентификатор каждому значению order_uuid.
   )
   PRIMARY KEY (order_uuid)
   DISTRIBUTED BY HASH (order_uuid)
   PROPERTIES("replicated_storage" = "true");
   ```

2. Используйте Stream Load для пакетной загрузки столбца `order_uuid` из двух CSV-файлов в столбец `order_uuid` таблицы словаря `dict`. Убедитесь, что вы используете частичное обновление в режиме столбцов.

```
curl --location-trusted -u root: \
    -H "partial_update: true" \
    -H "format: CSV" -H "column_separator:," -H "columns: id,
order_uuid" \
    -T batch1.csv \
    -XPUT http://<fe_host>:
<fe_http_port>/api/example_db/dict/_stream_load

curl --location-trusted -u root: \
    -H "partial_update: true" \
    -H "format: CSV" -H "column_separator:," -H "columns: id,
order_uuid" \
    -T batch2.csv \
    -XPUT http://<fe_host>:
<fe_http_port>/api/example_db/dict/_stream_load
```

> **ПРИМЕЧАНИЕ**
>
> Если новые данные добавляются в источник данных до перехода к следующему этапу, все новые данные должны быть загружены в таблицу словаря для обеспечения существования сопоставления.

Второй этап

Создайте целевую таблицу, включающую столбец идентификатора словаря с атрибутом `dict_mapping`. При загрузке данных заказов в целевую таблицу система автоматически свяжет их с таблицей словаря и вставит соответствующий идентификатор словаря.

1. Создайте таблицу `dest_table`, включающую все столбцы из CSV-файлов. Вам также нужно определить INTEGER-столбец `order_id_int` (обычно BIGINT) для сопоставления со STRING-столбцом `order_uuid`, имеющий атрибут столбца `dict_mapping`. Будущий анализ запросов будет основан на этом столбце `order_id_int`.

```
-- В целевой таблице определите BIGINT dict_mapping столбец
`order_id_int` для сопоставления со STRING-столбцом `order_uuid`.
CREATE TABLE dest_table (
    id BIGINT,
    order_uuid STRING, -- Этот столбец записывает STRING-идентификатор
заказа.
    batch INT comment 'Используется для различения разных пакетов
загрузки',
    order_id_int BIGINT AS dict_mapping('dict', order_uuid) -- Столбец
dict_mapping идентификатора словаря соответствует `order_uuid`.
)
DUPLICATE KEY (id, order_uuid)
DISTRIBUTED BY HASH(id);
```

2. Загрузите данные в целевую таблицу с помощью Stream Load или любого другого доступного метода. Поскольку столбец `order_id_int` имеет атрибут `dict_mapping`, система автоматически получит идентификатор словаря из `dict` во время загрузки.

```
curl --location-trusted -u root: \
    -H "format: CSV" -H "column_separator:," -H "columns: id,
order_uuid, batch=1" \
    -T batch1.csv \
    -XPUT http://<fe_host>:
<fe_http_port>/api/example_db/dest_table/_stream_load

curl --location-trusted -u root: \
    -H "format: CSV" -H "column_separator:," -H "columns: id,
order_uuid, batch=2" \
    -T batch2.csv \
    -XPUT http://<fe_host>:
<fe_http_port>/api/example_db/dest_table/_stream_load
```

Третий этап

Во время анализа запросов вы можете выполнять точную дедупликацию или соединения по INTEGER-столбцу `order_id_int`, что значительно повышает производительность по сравнению с использованием STRING-столбца `order_uuid`.

```
-- Точная дедупликация на основе BIGINT-типа order_id_int.
SELECT id, COUNT(DISTINCT order_id_int) FROM dest_table GROUP BY id ORDER
BY id;
-- Точная дедупликация на основе STRING-типа order_uuid.
SELECT id, COUNT(DISTINCT order_uuid) FROM dest_table GROUP BY id ORDER BY
id;
```

Вы также можете использовать bitmap-функции для ускорения точной дедупликации.

Использование bitmap-функций для ускорения точной дедупликации

Для дальнейшего ускорения вычислений вы можете вставить INTEGER-значения столбцов таблицы словаря непосредственно в bitmap-столбец после создания глобального словаря. Впоследствии вы можете использовать bitmap-функции на этом bitmap-столбце для точной дедупликации.

Способ 1

Если вы создали глобальный словарь и уже импортировали данные заказов в `dest_table`, выполните следующие шаги:

1. Создайте Aggregate-таблицу `dest_table_bitmap` с двумя столбцами: BITMAP-столбец `order_id_bitmap` для агрегации с использованием функции `bitmap_union()` и INTEGER-столбец `id`. Эта таблица не включает исходный STRING-столбец, иначе каждый bitmap содержал бы только одно значение, что сводит на нет преимущество ускорения.

   ```
   CREATE TABLE dest_table_bitmap (
       id BIGINT,
       order_id_bitmap BITMAP BITMAP_UNION
   )
   AGGREGATE KEY (id)
   DISTRIBUTED BY HASH(id) BUCKETS 6;
   ```

2. Вставьте данные в `dest_table_bitmap`. Вставьте данные из столбца `id` таблицы `dest_table` в столбец `id`, а INTEGER-столбец `order_id_int` из таблицы словаря `dict` (обработанный функцией `to_bitmap()`) — в столбец `order_id_bitmap`.

```
INSERT INTO dest_table_bitmap (id, order_id_bitmap)
SELECT id,  to_bitmap(dict_mapping('dict', order_uuid))
FROM dest_table
WHERE dest_table.batch = 1; -- Указывает разные пакеты.

INSERT INTO dest_table_bitmap (id, order_id_bitmap)
SELECT id, to_bitmap(dict_mapping('dict', order_uuid))
FROM dest_table
WHERE dest_table.batch = 2;
```

3. Используйте функцию `BITMAP_UNION_COUNT()` на BITMAP-столбце для точной дедупликации.

```
SELECT id, BITMAP_UNION_COUNT(order_id_bitmap) FROM dest_table_bitmap
GROUP BY id ORDER BY id;
```

Способ 2

Если вам не нужно сохранять конкретные данные заказов после создания глобального словаря и вы хотите загружать данные непосредственно в таблицу `dest_table_bitmap`, выполните следующие шаги:

1. Создайте Aggregate-таблицу `dest_table_bitmap` с двумя столбцами: BITMAP-столбец `order_id_bitmap` для агрегации с использованием функции `bitmap_union()` и INTEGER-столбец `id`. Эта таблица не включает исходный STRING-столбец, иначе каждый bitmap содержал бы только одно значение, что сводит на нет преимущество ускорения.

```
CREATE TABLE dest_table_bitmap (
    id BIGINT,
    order_id_bitmap BITMAP BITMAP_UNION
)
AGGREGATE KEY (id)
DISTRIBUTED BY HASH(id) BUCKETS 6;
```

2. Вставьте данные в Aggregate-таблицу. Вставьте данные из столбца `id` CSV-файлов в столбец `id`, а INTEGER-столбец `order_id_int` из таблицы словаря `dict` (обработанный функцией `to_bitmap()`) — в столбец `order_id_bitmap`.

```
curl --location-trusted -u root: \
    -H "format: CSV" -H "column_separator:," \
    -H "columns: id, order_uuid,
order_id_bitmap=to_bitmap(dict_mapping('dict', order_uuid))" \
    -T batch1.csv \
    -XPUT http://<fe_host>:
<fe_http_port>/api/example_db/dest_table_bitmap/_stream_load

curl --location-trusted -u root: \
    -H "format: CSV" -H "column_separator:," \
    -H "columns: id, order_uuid,
order_id_bitmap=to_bitmap(dict_mapping('dict', order_uuid))" \
    -T batch2.csv \
    -XPUT http:///<fe_host>:
<fe_http_port>/api/example_db/dest_table_bitmap/_stream_load
```

3. Используйте функцию `BITMAP_UNION_COUNT()` на BITMAP-столбце для точной дедупликации.

```
SELECT id, BITMAP_UNION_COUNT(order_id_bitmap) FROM dest_table_bitmap
GROUP BY id ORDER BY id;
```

6.13 Flat JSON

> *Beta* **BETA FEATURE**
>
> Рекомендации по использованию бета-функций

Эта статья описывает базовую концепцию Flat JSON и как использовать эту функцию.

Начиная с версии 2.2.0, Selena поддерживает тип данных JSON, чтобы обеспечить более гибкое хранение данных. Однако при запросе JSON большинство сценариев не включает непосредственное чтение всех данных JSON, а скорее доступ к данным по указанным путям. Например:

```
-- Сохранить необходимые поля в логах как фиксированные поля, а другие
поля, которые часто меняются с бизнесом, упаковать как JSON.
SELECT
    time,
    event,
    user,
    get_json_string(remain_json, "$.from_system"),
    get_json_string(remain_json, "$.tag")
FROM logs;
```

Из-за особой природы типа JSON его производительность в запросах не так хороша, как у стандартных типов (INT, STRING и т.д.). Причины включают:

- Накладные расходы хранения: JSON является полуструктурированным типом, который требует хранения информации о структуре каждой строки, что приводит к высокому использованию хранилища и низкой эффективности сжатия.
- Сложность запроса: Запросы должны определять структуры данных на основе данных времени выполнения, что затрудняет достижение оптимизации векторизованного выполнения.
- Избыточные данные: Запросы должны читать все данные JSON, которые включают много избыточных полей.

Selena вводит функцию Flat JSON для улучшения эффективности запросов данных JSON и снижения сложности использования JSON.

- Эта функция доступна начиная с версии 3.3.0. Она отключена по умолчанию до v2.0.0 и должна быть включена вручную. Начиная с v2.0.0, она включена по умолчанию.

6.13.1 Что такое Flat JSON

Основной принцип Flat JSON заключается в обнаружении данных JSON во время загрузки и извлечении общих полей из данных JSON для хранения в качестве данных стандартного типа. При запросе JSON эти общие поля оптимизируют скорость запроса JSON. Пример данных:

```
1, {"a": 1, "b": 21, "c": 3, "d": 4}
2, {"a": 2, "b": 22, "d": 4}
3, {"a": 3, "b": 23, "d": [1, 2, 3, 4]}
4, {"a": 4, "b": 24, "d": null}
5, {"a": 5, "b": 25, "d": null}
6, {"c": 6, "d": 1}
```

При загрузке вышеуказанных данных JSON поля `a` и `b` присутствуют в большинстве данных JSON и имеют схожие типы данных (оба INT). Поэтому данные полей `a` и `b` могут быть извлечены из JSON и сохранены отдельно как два столбца INT. Когда эти два столбца используются в запросах, их данные могут быть непосредственно прочитаны без необходимости обработки дополнительных полей JSON, что снижает вычислительные накладные расходы обработки структур JSON.

6.13.2 Включение Flat JSON

Flat JSON включен глобально по умолчанию начиная с v2.0.0. Для версий ранее v2.0.0 вы должны включить его вручную.

Начиная с v2.0.0, эта функция может быть настроена на уровне таблицы.

Включение для версий ранее v2.0.0

1. Измените конфигурацию BE: `enable_json_flat`, которая по умолчанию `false` до v2.0.0. Для методов изменения см. Configure BE parameters.

2. Включите функцию pruning FE:

   ```
   SET GLOBAL cbo_prune_json_subfield = true;
   ```

Включение функции Flat JSON на уровне таблицы

Установка свойств, связанных с Flat JSON, на уровне таблицы поддерживается начиная с v2.0.0.

1. При создании таблицы вы можете установить `flat_json.enable` и другие свойства, связанные с Flat JSON. Подробные инструкции см. в CREATE TABLE.

   Альтернативно, вы можете установить эти свойства с помощью ALTER TABLE.

Пример:

```
ALTER TABLE t1 SET ("flat_json.enable" = "true");
ALTER TABLE t1 SET ("flat_json.null.factor" = "0.1");
ALTER TABLE t1 SET ("flat_json.sparsity.factor" = "0.8");
ALTER TABLE t1 SET ("flat_json.column.max" = "90");
```

2. Включите функцию pruning FE:

```
SET GLOBAL cbo_prune_json_subfield = true;
```

6.13.3 Проверка эффективности Flat JSON

После загрузки данных вы можете запросить извлеченные подстолбцы соответствующего столбца:

```
SELECT flat_json_meta(<json_column>)
FROM <table_name>[_META_];
```

Вы можете проверить, получает ли выполняемый запрос преимущества от оптимизации Flat JSON, через Query Profile, наблюдая следующие метрики:

- `PushdownAccessPaths`: Количество путей подполей, проталкиваемых в хранилище.
- `AccessPathHits`: Количество попаданий подполей Flat JSON с подробной информацией о конкретном попадании JSON.
- `AccessPathUnhits`: Количество промахов подполей Flat JSON с подробной информацией о конкретном промахе JSON.
- `JsonFlattern`: Время, затраченное на извлечение подстолбцов на месте, когда Flat JSON не попадает.

6.13.4 Пример использования

- Метод 1: Настройка свойств Flat JSON при создании таблицы со столбцами JSON. Поддерживается начиная с v2.0.0.

```
CREATE TABLE `t1` (
    `k1` int,
    `k2` JSON,
    `k3` VARCHAR(20),
    `k4` JSON
)
DUPLICATE KEY(`k1`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`k1`) BUCKETS 2
PROPERTIES (
    "replication_num" = "3",
    "flat_json.enable" = "true",
    "flat_json.null.factor" = "0.5",
    "flat_json.sparsity.factor" = "0.5",
    "flat_json.column.max" = "50");
 INSERT INTO t1 (k1,k2) VALUES

(11,parse_json('{"str":"test_flat_json","Integer":123456,"Double":3.14158,
{"c":"d"},"arr":[10,20,30],"Bool":false,"null":null}')),
    (15,parse_json('{"str":"test_str0","Integer":11,"Double":3.14,"Object
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}')),
    (15,parse_json('{"str":"test_str1","Integer":111,"Double":3.14,"Objec
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}')),
    (15,parse_json('{"str":"test_str2","Integer":222,"Double":3.14,"Objec
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}')),
    (15,parse_json('{"str":"test_str2","Integer":222,"Double":3.14,"Objec
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}')),
    (16,parse_json('{"str":"test_str3","Integer":333,"Double":3.14,"Objec
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}')),
    (17,parse_json('{"str":"test_str3","Integer":333,"Double":3.14,"Objec
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}')),
    (18,parse_json('{"str":"test_str5","Integer":444,"Double":3.14,"Objec
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}')),
    (19,parse_json('{"str":"test_str6","Integer":444,"Double":3.14,"Objec
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}')),
    (20,parse_json('{"str":"test_str6","Integer":444,"Double":3.14,"Objec
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}'));
```

- Метод 2: Включение Flat JSON перед созданием таблицы.

1. Включите функцию (см. другие разделы)

2. Создайте таблицу со столбцами JSON. В этом примере используйте INSERT INTO для загрузки данных JSON в таблицу.

```
CREATE TABLE `t1` (
    `k1` int,
    `k2` JSON,
    `k3` VARCHAR(20),
    `k4` JSON
)
DUPLICATE KEY(`k1`)
COMMENT "OLAP"
DISTRIBUTED BY HASH(`k1`) BUCKETS 2
PROPERTIES ("replication_num" = "3");


INSERT INTO t1 (k1,k2) VALUES

(11,parse_json('{"str":"test_flat_json","Integer":123456,"Double":3.14158,
{"c":"d"},"arr":[10,20,30],"Bool":false,"null":null}')),
   (15,parse_json('{"str":"test_str0","Integer":11,"Double":3.14,"Object"
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}')),
   (15,parse_json('{"str":"test_str1","Integer":111,"Double":3.14,"Object
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}')),
   (15,parse_json('{"str":"test_str2","Integer":222,"Double":3.14,"Object
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}')),
   (15,parse_json('{"str":"test_str2","Integer":222,"Double":3.14,"Object
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}')),
   (16,parse_json('{"str":"test_str3","Integer":333,"Double":3.14,"Object
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}')),
   (17,parse_json('{"str":"test_str3","Integer":333,"Double":3.14,"Object
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}')),
   (18,parse_json('{"str":"test_str5","Integer":444,"Double":3.14,"Object
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}')),
   (19,parse_json('{"str":"test_str6","Integer":444,"Double":3.14,"Object
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}')),
   (20,parse_json('{"str":"test_str6","Integer":444,"Double":3.14,"Object
{"a":"b"},"arr":[1,2,3],"Bool":true,"null":null}'));
```

3. Просмотрите извлеченные подстолбцы для столбца `k2`.

```
SELECT flat_json_meta(k2) FROM t1[_META_];
+-----------------------------------------------------------------------
---+
| flat_json_meta(k2)
|
+-----------------------------------------------------------------------
---+
|
["nulls(TINYINT)","Integer(BIGINT)","Double(DOUBLE)","str(VARCHAR)","Bool
|
+-----------------------------------------------------------------------
---+
```

4. Выполните запросы данных.

```
SELECT * FROM t1;
SELECT get_json_string(k2,'\$.Integer') FROM t1 WHERE k2->'str' =
'test_flat_json';
SELECT get_json_string(k2,'\$.Double') FROM t1 WHERE k2->'Integer' =
123456;
SELECT get_json_string(k2,'\$.Object') FROM t1 WHERE k2->'Double' =
3.14158;
SELECT get_json_string(k2,'\$.arr') FROM t1 WHERE k2->'Object' =
to_json(map{'c':'d'});
SELECT get_json_string(k2,'\$.Bool') FROM t1 WHERE k2->'arr' =
'[10,20,30]';
```

5. Просмотрите метрики, связанные с Flat JSON, в Query Profile

```
   PushdownAccessPaths: 2
   - Table: t1
   - AccessPathHits: 2
   - __MAX_OF_AccessPathHits: 1
   - __MIN_OF_AccessPathHits: 1
   - /k2: 2
      - __MAX_OF_/k2: 1
      - __MIN_OF_/k2: 1
   - AccessPathUnhits: 0
   - JsonFlattern: 0ns
```

6.13.5 Связанные переменные и конфигурации

Переменные сессии

- `cbo_json_v2_rewrite` (по умолчанию: true): Включает перезапись путей JSON v2, чтобы выражения типа `get_json_*` могли быть переписаны в прямой доступ к подстолбцам Flat JSON, открывая проталкивание предикатов и обрезку столбцов.
- `cbo_json_v2_dict_opt` (по умолчанию: true): Включает оптимизацию словаря низкой мощности для расширенных строковых подстолбцов Flat JSON, произведенных перезаписью, что может ускорить строковые выражения, GROUP BY и JOIN.

Пример:

```
SET cbo_json_v2_rewrite = true;
SET cbo_json_v2_dict_opt = true;
```

Конфигурации BE

- json_flat_null_factor
- json_flat_column_max
- json_flat_sparsity_factor
- enable_compaction_flat_json
- enable_lazy_dynamic_flat_json

6.13.6 Ограничения функции

- Все типы таблиц в Selena поддерживают Flat JSON.
- Совместима с историческими данными без необходимости повторного импорта. Исторические данные будут сосуществовать с данными, сглаженными с помощью Flat JSON.
- Исторические данные не будут автоматически применять оптимизацию Flat JSON, если не будут загружены новые данные или не произойдет Compaction.
- Включение Flat JSON увеличит время загрузки JSON. Чем больше JSON извлекается, тем дольше это занимает.
- Flat JSON может поддерживать только материализацию общих ключей в JSON Objects, а не ключей в JSON Arrays.
- Flat JSON не изменяет метод сортировки данных, поэтому производительность запросов и степень сжатия данных все еще будут зависеть от сортировки данных. Для

достижения оптимальной производительности могут потребоваться дополнительные настройки сортировки данных.

6.13.7 Примечания к версиям

Кластеры Selena shared-nothing поддерживают Flat JSON начиная с v1.5.2, а кластеры shared-data поддерживают его начиная с v1.5.2.

В версиях v1.5.2:

- При загрузке данных поддерживается извлечение общих полей и их раздельное хранение как типов JSON без определения типа.
- Хранятся как извлеченные столбцы, так и исходные данные JSON. Извлеченные данные будут удалены вместе с исходными данными.

Начиная с версии v1.5.2:

- Результаты, извлеченные Flat JSON, делятся на общие столбцы и столбцы зарезервированных полей. Когда все схемы JSON согласованы, столбцы зарезервированных полей не генерируются.
- Flat JSON хранит только столбцы общих полей и столбцы зарезервированных полей, без дополнительного хранения исходных данных JSON.
- При загрузке данных общие поля будут автоматически определять типы как BIGINT/LARGEINT/DOUBLE/STRING. Нераспознанные типы будут определены как типы JSON, а столбцы зарезервированных полей будут храниться как типы JSON.

6.14 Ускорение соединений, сохраняющих кардинальность

Эта тема описывает, как ускорить соединения, сохраняющие кардинальность, с помощью отсечения таблиц. Эта функция поддерживается начиная с версии v1.5.2.

6.14.1 Обзор

Соединение, сохраняющее кардинальность, гарантирует, что кардинальность и фактор дублирования выходных строк остаются такими же, как у одной из входных таблиц в соединении. Рассмотрим следующие примеры:

- **Inner Join:**

```
SELECT A.* FROM A INNER JOIN B ON A.fk = B.pk;
```

В этом случае `A.fk` (Foreign Key) является **NOT NULL** и ссылается на `B.pk` (Primary Key). Каждая строка в `A` соответствует ровно одной строке в `B`, поэтому кардинальность и фактор дублирования вывода соответствуют `A`.

- **Left Join:**

```
SELECT A.* FROM A LEFT JOIN B ON A.fk = B.pk;
```

Здесь `A.fk` ссылается на `B.pk`, но `A.fk` может содержать NULL значения. Каждая строка в `A` соответствует не более чем одной строке в `B`. В результате кардинальность и фактор дублирования вывода остаются согласованными с `A`.

В этих типах соединений, если конечные выходные столбцы зависят только от столбцов таблицы `A`, и столбцы таблицы `B` не используются, таблица `B` может быть отсечена из соединения. Начиная с версии v1.5.2, Selena поддерживает **отсечение таблиц** в соединениях, сохраняющих кардинальность, которые могут встречаться в общих табличных выражениях (CTE), логических представлениях и подзапросах.

Вариант использования: Выбор признаков в реальном времени в контроле рисков

Функция отсечения таблиц для соединений, сохраняющих кардинальность, особенно полезна в сценариях, таких как **выбор признаков в реальном времени** для контроля рисков. В этом контексте пользователям необходимо выбирать данные из большого количества таблиц, часто имея дело с комбинаторным взрывом столбцов и таблиц. Следующие характеристики распространены в области контроля рисков:

- Многочисленные признаки распределены по многим независимо обновляемым таблицам.
- Свежие данные должны быть видимыми и запрашиваемыми в реальном времени.
- **Плоское логическое представление** используется для упрощения модели данных, делая SQL для извлечения столбцов более лаконичным и продуктивным.

Использование плоского логического представления, а не других ускоренных слоев данных, помогает пользователям эффективно получать доступ к данным в реальном времени. В каждом запросе извлечения столбца требуется соединить только несколько таблиц (не все таблицы в логическом представлении). Отсекая неиспользуемые таблицы из этих запросов, вы можете уменьшить количество соединений и улучшить производительность.

Поддержка функций

Функция отсечения таблиц поддерживает многотабличные соединения как в **схеме звезды**, так и в **схеме снежинки**. Многотабличные соединения могут появляться в CTE, логических представлениях и подзапросах, обеспечивая более эффективное выполнение запросов.

В настоящее время функция отсечения таблиц поддерживается только для таблиц OLAP и Cloud-native таблиц. Внешние таблицы в многотабличных соединениях не могут быть отсечены.

6.14.2 Использование

Следующие примеры используют набор данных TPC-H.

Предварительные условия

Для использования функции отсечения таблиц должны быть выполнены следующие условия:

1. Включить отсечение таблиц
2. Установить ключевые ограничения

Включение отсечения таблиц

По умолчанию отсечение таблиц отключено. Вам необходимо включить функцию, настроив следующие переменные сессии:

```
-- Включить отсечение таблиц на этапе RBO.
SET enable_rbo_table_prune=true;
-- Включить отсечение таблиц на этапе CBO.
SET enable_cbo_table_prune=true;
-- Включить отсечение таблиц на этапе RBO для оператора UPDATE на таблицах
Primary Key.
SET enable_table_prune_on_update = true;
```

Установка ключевых ограничений

Таблицы, которые должны быть отсечены, должны иметь ограничения Unique Key или Primary Key, по крайней мере, в LEFT или RIGHT Joins. Для отсечения таблиц в INNER JOIN вы должны определить ограничения Foreign Key в дополнение к ограничениям Unique Key или Primary Key.

Таблицы Primary Key и Unique Key имеют свои неявные ограничения Primary Key или Unique Key, естественно встроенные в них. Однако для таблиц Duplicate Key вы должны вручную определить ограничения Unique Key и убедиться, что дублирующиеся строки не существуют. Обратите внимание, что Selena не применяет ограничения Unique Key на таблицах Duplicate Key. Вместо этого она рассматривает их как подсказки для оптимизации для более агрессивного планирования запросов.

Пример:

```
-- Определение ограничения Unique Key во время создания таблицы.
CREATE TABLE `lineitem` (
  `l_orderkey` int(11) NOT NULL COMMENT "",
  `l_partkey` int(11) NOT NULL COMMENT "",
  `l_suppkey` int(11) NOT NULL COMMENT "",
  `l_linenumber` int(11) NOT NULL COMMENT "",
  `l_quantity` decimal64(15, 2) NOT NULL COMMENT "",
  `l_extendedprice` decimal64(15, 2) NOT NULL COMMENT "",
  `l_discount` decimal64(15, 2) NOT NULL COMMENT "",
  `l_tax` decimal64(15, 2) NOT NULL COMMENT "",
  `l_returnflag` varchar(1) NOT NULL COMMENT "",
  `l_linestatus` varchar(1) NOT NULL COMMENT "",
  `l_shipdate` date NOT NULL COMMENT "",
  `l_commitdate` date NOT NULL COMMENT "",
  `l_receiptdate` date NOT NULL COMMENT "",
  `l_shipinstruct` varchar(25) NOT NULL COMMENT "",
  `l_shipmode` varchar(10) NOT NULL COMMENT "",
  `l_comment` varchar(44) NOT NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`l_orderkey`,`l_partkey`, `l_suppkey`)
DISTRIBUTED BY HASH(`l_orderkey`) BUCKETS 96
PROPERTIES (
"unique_constraints" = "l_orderkey,l_linenumber"
);

-- Или вы можете определить ограничение Unique Key после создания таблицы.
ALTER TABLE lineitem SET ("unique_constraints" =
"l_orderkey,l_linenumber");
```

Ограничения Foreign Key, с другой стороны, должны быть определены явно. Подобно ограничениям Unique Key на таблицах Duplicate Key, ограничения Foreign Key действуют как подсказки для оптимизатора. Selena не обеспечивает согласованность ограничений Foreign Key. Вы должны обеспечить целостность данных при производстве и загрузке данных в Selena.

Пример:

```
-- Создание таблицы, на которую будет ссылаться ограничение Foreign Key.
-- Обратите внимание, что столбец, на который ссылаются, должен иметь
ограничения Unique Key или Primary Key.
-- В этом примере `p_partkey` является Primary Key таблицы `part`.
CREATE TABLE part (
    p_partkey     int(11) NOT NULL,
    p_name        VARCHAR(55) NOT NULL,
    p_mfgr        CHAR(25) NOT NULL,
    p_brand       CHAR(10) NOT NULL,
    p_type        VARCHAR(25) NOT NULL,
    p_size        INT NOT NULL,
    p_container   CHAR(10) NOT NULL,
    p_retailprice DOUBLE NOT NULL,
    p_comment     VARCHAR(23) NOT NULL
) ENGINE=OLAP
PRIMARY KEY(`p_partkey`)
DISTRIBUTED BY HASH(`p_partkey`) BUCKETS 12;

-- Определение ограничения Foreign Key во время создания таблицы.
CREATE TABLE `lineitem` (
  `l_orderkey` int(11) NOT NULL COMMENT "",
  `l_partkey` int(11) NOT NULL COMMENT "",
  `l_suppkey` int(11) NOT NULL COMMENT "",
  `l_linenumber` int(11) NOT NULL COMMENT "",
  `l_quantity` decimal64(15, 2) NOT NULL COMMENT "",
  `l_extendedprice` decimal64(15, 2) NOT NULL COMMENT "",
  `l_discount` decimal64(15, 2) NOT NULL COMMENT "",
  `l_tax` decimal64(15, 2) NOT NULL COMMENT "",
  `l_returnflag` varchar(1) NOT NULL COMMENT "",
  `l_linestatus` varchar(1) NOT NULL COMMENT "",
  `l_shipdate` date NOT NULL COMMENT "",
  `l_commitdate` date NOT NULL COMMENT "",
  `l_receiptdate` date NOT NULL COMMENT "",
  `l_shipinstruct` varchar(25) NOT NULL COMMENT "",
  `l_shipmode` varchar(10) NOT NULL COMMENT "",
  `l_comment` varchar(44) NOT NULL COMMENT ""
) ENGINE=OLAP
DUPLICATEK KEY(`l_orderkey`,`l_partkey`, `l_suppkey`)
DISTRIBUTED BY HASH(`l_orderkey`) BUCKETS 96
PROPERTIES (
"foreign_key_constraints" =  "(l_partkey) REFERENCES part(p_partkey)"
);

-- Или вы можете определить ограничение Foreign Key после создания таблицы.
```

```
ALTER TABLE lineitem SET ("foreign_key_constraints" =  "(l_partkey)
REFERENCES part(p_partkey)");
```

Отсечение таблиц в LEFT/RIGHT JOIN на основе Unique или Primary Keys

Отсечение таблиц в LEFT или RIGHT JOIN не требует, чтобы сохраняемая сторона соединения имела Foreign Key, ссылающийся на отсекаемую сторону. Это делает отсечение более гибким и надежным, даже если целостность ссылок не может быть гарантирована.

Отсечение в LEFT/RIGHT JOIN на основе Unique или Primary keys имеет **менее строгие требования** по сравнению с отсечением INNER JOIN на основе Foreign Keys.

Условия для отсечения:

- **Отсекаемая сторона**

  Отсекаемая таблица должна быть **правой стороной** в LEFT JOIN или **левой стороной** в RIGHT JOIN.

- **Условия соединения**

  Соединение должно использовать только условия равенства (`=`), и соединяемые столбцы отсекаемой стороны должны быть **надмножеством** Unique или Primary keys.

- **Выходные столбцы**

  Только столбцы из сохраняемой стороны должны быть в выводе, и результат должен поддерживать **такую же кардинальность и фактор дублирования**, как у сохраняемой стороны.

- **NULL/значения по умолчанию**

  Соединяемые столбцы в сохраняемой стороне **могут** содержать NULL или другие значения по умолчанию, которые не соответствуют отсекаемой стороне.

Пример:

1. Создайте таблицы и вставьте данные.

```
-- Таблица `depts` имеет ограничение Primary Key на столбце `deptno`.
CREATE TABLE `depts` (
`deptno` int(11) NOT NULL COMMENT "",
`name` varchar(25) NOT NULL COMMENT ""
) ENGINE=OLAP
PRIMARY KEY(`deptno`)
DISTRIBUTED BY HASH(`deptno`) BUCKETS 10;

CREATE TABLE `emps` (
`empid` int(11) NOT NULL COMMENT "",
`deptno` int(11) NOT NULL COMMENT "",
`name` varchar(25) NOT NULL COMMENT "",
`salary` double NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`empid`)
DISTRIBUTED BY HASH(`empid`) BUCKETS 10;

INSERT INTO depts VALUES
(1, "R&D"),
(2, "Marketing"),
(3, "Community"),
(4, "DBA"),(5, "POC");

INSERT INTO emps VALUES
(1, 1, "Alice", "6000"),
(2, 1, "Bob", "6100"),
(3, 2, "Candy", "10000"),
(4, 2, "Dave", "20000"),
(5, 3, "Evan","18000"),
(6, 3, "Freman","1000"),
(7, 4, "George","1800"),
(8, 4, "Harry","2000"),
(9, 5, "Ivan", "15000"),
(10, 5, "Jim","20000"),
(11, -1, "Kevin","1500"),
(12, -1, "Lily","2500");
```

2. Просмотрите логические планы выполнения запросов.

```
-- Q1: Запрос всех столбцов `emps`. Не все столбцы `depts`
запрашиваются.
EXPLAIN LOGICAL SELECT emps.* FROM emps LEFT JOIN depts ON emps.deptno
= depts.deptno;
+-----------------------------------------------------------------------
-------+
| Explain String
|
+-----------------------------------------------------------------------
-------+
| - Output => [1:empid, 2:deptno, 3:name, 4:salary]
|
|     - SCAN [emps] => [1:empid, 2:deptno, 3:name, 4:salary]
|
|             Estimates: {row: 10, cpu: ?, memory: ?, network: ?, cost:
20.0} |
|             partitionRatio: 1/1, tabletRatio: 10/10
|
+-----------------------------------------------------------------------
-------+

-- Q2: Запрос только `deptno` и `salary` в `emps`. Ни один столбец в
`depts` не запрашивается.
EXPLAIN LOGICAL SELECT emps.deptno, avg(salary) AS mean_salary
FROM emps LEFT JOIN depts ON emps.deptno = depts.deptno
GROUP BY emps.deptno
ORDER BY mean_salary DESC
LIMIT 5;
+-----------------------------------------------------------------------
---------------------------+
| Explain String
|
+-----------------------------------------------------------------------
---------------------------+
| - Output => [2:deptno, 7:avg]
|
|     - TOP-5(FINAL)[7: avg DESC NULLS LAST]
|
|             Estimates: {row: 3, cpu: ?, memory: ?, network: ?, cost:
138.0}                      |
|         - TOP-5(PARTIAL)[7: avg DESC NULLS LAST]
|
|                 Estimates: {row: 3, cpu: ?, memory: ?, network: ?,
cost: 114.0}                  |
|             - AGGREGATE(GLOBAL) [2:deptno]
```

```
|
|                     Estimates: {row: 3, cpu: ?, memory: ?, network:
?, cost: 90.0}              |
|                     7:avg := avg(7:avg)
|
|                 - EXCHANGE(SHUFFLE) [2]
|
|                         Estimates: {row: 6, cpu: ?, memory: ?,
network: ?, cost: 72.0}          |
|                     - AGGREGATE(LOCAL) [2:deptno]
|
|                             Estimates: {row: 6, cpu: ?, memory: ?,
network: ?, cost: 48.0}      |
|                             7:avg := avg(4:salary)
|
|                         - SCAN [emps] => [2:deptno, 4:salary]
|
|                                 Estimates: {row: 12, cpu: ?, memory:
?, network: ?, cost: 12.0} |
|                                 partitionRatio: 1/1, tabletRatio:
10/10                         |
+-----------------------------------------------------------------------
---------------------------+

-- Q3: Запрашиваются только столбцы в `emps`. Хотя предикат `name =
"R&D"` только
-- выбирает определенные строки `depts`, конечные результаты зависят
только от `emps`.
EXPLAIN LOGICAL SELECT emps.deptno, avg(salary) AS mean_salary
FROM emps LEFT JOIN
(SELECT deptno FROM depts WHERE name="R&D") t ON emps.deptno = t.deptno
GROUP BY emps.deptno
ORDER BY mean_salary DESC
LIMIT 5;
+-----------------------------------------------------------------------
---------------------------+
| Explain String
|
+-----------------------------------------------------------------------
---------------------------+
| - Output => [2:deptno, 7:avg]
|
|     - TOP-5(FINAL)[7: avg DESC NULLS LAST]
|
|             Estimates: {row: 3, cpu: ?, memory: ?, network: ?, cost:
138.0}                    |
```

```
|         - TOP-5(PARTIAL)[7: avg DESC NULLS LAST]
|
|                 Estimates: {row: 3, cpu: ?, memory: ?, network: ?,
cost: 114.0}                  |
|             - AGGREGATE(GLOBAL) [2:deptno]
|
|                     Estimates: {row: 3, cpu: ?, memory: ?, network:
?, cost: 90.0}               |
|                     7:avg := avg(7:avg)
|
|                 - EXCHANGE(SHUFFLE) [2]
|
|                         Estimates: {row: 6, cpu: ?, memory: ?,
network: ?, cost: 72.0}          |
|                     - AGGREGATE(LOCAL) [2:deptno]
|
|                             Estimates: {row: 6, cpu: ?, memory: ?,
network: ?, cost: 48.0}      |
|                             7:avg := avg(4:salary)
|
|                         - SCAN [emps] => [2:deptno, 4:salary]
|
|                                 Estimates: {row: 12, cpu: ?, memory:
?, network: ?, cost: 12.0} |
|                                 partitionRatio: 1/1, tabletRatio:
10/10                         |
+-------------------------------------------------------------------------
---------------------------+

-- Q4: Предикат `depts.name="R&D"` в условии WHERE нарушает
-- условия сохранения кардинальности, поэтому `depts` не может быть
отсечена.
EXPLAIN LOGICAL SELECT emps.deptno, avg(salary) AS mean_salary
FROM emps LEFT JOIN depts ON emps.deptno = depts.deptno
WHERE depts.name="R&D";

+-------------------------------------------------------------------------
-------------------------+
| Explain String
|
+-------------------------------------------------------------------------
-------------------------+
| - Output => [8:any_value, 7:avg]
|
|     - AGGREGATE(GLOBAL) []
|
```

```
|             Estimates: {row: 1, cpu: ?, memory: ?, network: ?, cost:
110.5}                    |
|             7:avg := avg(7:avg)
|
|             8:any_value := any_value(8:any_value)
|
|         - EXCHANGE(GATHER)
|
|                 Estimates: {row: 1, cpu: ?, memory: ?, network: ?,
cost: 105.5}               |
|             - AGGREGATE(LOCAL) []
|
|                     Estimates: {row: 1, cpu: ?, memory: ?, network:
?, cost: 101.5}           |
|                     7:avg := avg(4:salary)
|
|                     8:any_value := any_value(2:deptno)
|
|                 - HASH/INNER JOIN [2:deptno = 5:deptno] => [2:deptno,
4:salary]               |
|                       Estimates: {row: 12, cpu: ?, memory: ?,
network: ?, cost: 79.5}       |
|                     - SCAN [emps] => [2:deptno, 4:salary]
|
|                             Estimates: {row: 12, cpu: ?, memory: ?,
network: ?, cost: 12.0}   |
|                             partitionRatio: 1/1, tabletRatio: 10/10
|
|                     - EXCHANGE(BROADCAST)
|
|                             Estimates: {row: 5, cpu: ?, memory: ?,
network: ?, cost: 25.0}    |
|                       - SCAN [depts] => [5:deptno]
|
|                                 Estimates: {row: 5, cpu: ?, memory:
?, network: ?, cost: 5.0} |
|                                 partitionRatio: 1/1, tabletRatio:
10/10                       |
|                                 predicate: 6:name = 'R&D'
|
+-------------------------------------------------------------------------
-------------------------+
```

В приведенном выше примере отсечение таблиц разрешено в Q1, Q2 и Q3, как показано в плане выполнения, в то время как Q4, который нарушает условия сохранения кардинальности, не выполняет отсечение таблиц.

Отсечение таблиц в INNER/LEFT/RIGHT JOIN на основе Foreign Keys

Отсечение таблиц в **INNER JOIN** более ограничено, поскольку сохраняемая сторона должна иметь **Foreign Key**, который ссылается на отсекаемую сторону, и целостность ссылок должна быть обеспечена. В настоящее время Selena не обеспечивает проверки согласованности ограничений Foreign Key.

Хотя отсечение таблиц на основе Foreign Keys более строгое, оно также более мощное. Оно позволяет оптимизатору использовать вывод эквивалентности столбцов в INNER JOIN, делая отсечение возможным в более сложных сценариях.

Условия для отсечения следующие:

- **Отсекаемая сторона**
  - В LEFT JOIN отсекаемая таблица должна быть на правой стороне. В RIGHT JOIN она должна быть на левой стороне.
  - В INNER JOIN отсекаемая таблица должна иметь ограничение Unique Key на соединяемых столбцах, и каждая строка в сохраняемой стороне должна соответствовать ровно одной строке в отсекаемой стороне. Если в сохраняемой стороне существуют несоответствующие строки, отсечение не может произойти.
- **Условия соединения**

  Соединение должно использовать только условия равенства (`=`). Соединяемые столбцы в сохраняемой стороне должны быть **Foreign Keys**, в то время как соединяемые столбцы в отсекаемой стороне должны быть **Primary Keys** или **Unique Keys**. Эти столбцы должны быть согласованы в соответствии с ограничениями Foreign Key.

- **Выходные столбцы**
  - Только столбцы из сохраняемой стороны должны быть в выводе, и результат должен поддерживать **такую же кардинальность и фактор дублирования**, как у сохраняемой стороны.

- Для INNER JOIN соединяемые столбцы из отсекаемой стороны могут быть заменены их эквивалентными столбцами в сохраняемой стороне. Если все выходные столбцы из отсекаемой стороны являются соединяемыми столбцами, отсечение также может произойти.

- **NULL/значения по умолчанию:**

  Соединяемые столбцы в сохраняемой стороне **не могут** содержать NULL или другие значения по умолчанию, которые не соответствуют отсекаемой стороне.

Пример:

1. Создайте таблицы, определите Foreign Key и вставьте данные.

```
CREATE TABLE `depts` (
`deptno` int(11) NOT NULL COMMENT "",
`name` varchar(25) NOT NULL COMMENT ""
) ENGINE=OLAP
PRIMARY KEY(`deptno`)
DISTRIBUTED BY HASH(`deptno`) BUCKETS 10;

CREATE TABLE `emps` (
`empid` int(11) NOT NULL COMMENT "",
`deptno` int(11) NOT NULL COMMENT "",
`name` varchar(25) NOT NULL COMMENT "",
`salary` double NULL COMMENT ""
) ENGINE=OLAP
DUPLICATE KEY(`empid`)
DISTRIBUTED BY HASH(`empid`) BUCKETS 10;

ALTER TABLE emps SET ("foreign_key_constraints" = "(deptno) REFERENCES
depts(deptno)");

INSERT INTO depts VALUES
(1, "R&D"),
(2, "Marketing"),
(3, "Community"),
(4, "DBA"),(5, "POC");

INSERT INTO emps VALUES
(1, 1, "Alice", "6000"),
(2, 1, "Bob", "6100"),
(3, 2, "Candy", "10000"),
(4, 2, "Dave", "20000"),
(5, 3, "Evan","18000"),
(6, 3, "Freman","1000"),
(7, 4, "George","1800"),
(8, 4, "Harry","2000"),
(9, 5, "Ivan", "15000"),
(10, 5, "Jim","20000");
```

2. Просмотрите логические планы выполнения запросов.

```
-- Q1: Запрос `empid` и `name` в `emps`, и `deptno` в `depts`.
-- Однако, как показано в условии Join, `emps.deptno` эквивалентен
`depts.deptno`,
-- поэтому `emps.deptno` может заменить `depts.deptno`.
EXPLAIN LOGICAL WITH t0 AS (
SELECT empid, depts.deptno, emps.name, emps.salary, depts.name AS
dept_name
FROM emps INNER JOIN depts ON emps.deptno = depts.deptno
)
SELECT empid, deptno, name FROM t0;
+-----------------------------------------------------------------------
-------+
| Explain String
|
+-----------------------------------------------------------------------
-------+
| - Output => [7:empid, 8:deptno, 9:name]
|
|     - SCAN [emps] => [7:empid, 8:deptno, 9:name]
|
|             Estimates: {row: 12, cpu: ?, memory: ?, network: ?, cost:
18.0} |
|             partitionRatio: 1/1, tabletRatio: 10/10
|
+-----------------------------------------------------------------------
-------+

-- Q2: Запрашивается только `salary` в `emps`.
EXPLAIN LOGICAL SELECT avg(salary)
FROM emps INNER JOIN depts ON emps.deptno = depts.deptno;
+-----------------------------------------------------------------------
------------------+
| Explain String
|
+-----------------------------------------------------------------------
------------------+
| - Output => [7:avg]
|
|     - AGGREGATE(GLOBAL) []
|
|             Estimates: {row: 1, cpu: ?, memory: ?, network: ?, cost:
16.5}             |
|             7:avg := avg(7:avg)
|
|         - EXCHANGE(GATHER)
```

```
|
|                 Estimates: {row: 1, cpu: ?, memory: ?, network: ?,
cost: 14.0}          |
|             - AGGREGATE(LOCAL) []
|
|                     Estimates: {row: 1, cpu: ?, memory: ?, network:
?, cost: 12.0}      |
|                     7:avg := avg(4:salary)
|
|                 - SCAN [emps] => [4:salary]
|
|                         Estimates: {row: 10, cpu: ?, memory: ?,
network: ?, cost: 5.0} |
|                         partitionRatio: 1/1, tabletRatio: 10/10
|
+-----------------------------------------------------------------------
------------------+

-- Q3: Предикат `name="R&D"` влияет на кардинальность конечных
результатов.
EXPLAIN LOGICAL SELECT emps.deptno, avg(salary) AS mean_salary
FROM emps INNER JOIN
(SELECT deptno FROM depts WHERE name="R&D") t ON emps.deptno = t.deptno
GROUP BY emps.deptno
ORDER BY mean_salary DESC
LIMIT 5;
+-----------------------------------------------------------------------
---------------------------------+
| Explain String
|
+-----------------------------------------------------------------------
---------------------------------+
| - Output => [2:deptno, 7:avg]
|
|     - TOP-5(FINAL)[7: avg DESC NULLS LAST]
|
|             Estimates: {row: 2, cpu: ?, memory: ?, network: ?, cost:
165.0480769230769}               |
|         - TOP-5(PARTIAL)[7: avg DESC NULLS LAST]
|
|                 Estimates: {row: 2, cpu: ?, memory: ?, network: ?,
cost: 145.0480769230769}           |
|             - AGGREGATE(GLOBAL) [2:deptno]
|
|                     Estimates: {row: 2, cpu: ?, memory: ?, network:
?, cost: 125.04807692307692}      |
```

```
|                     7:avg := avg(4:salary)
|
|                 - HASH/INNER JOIN [2:deptno = 5:deptno] => [2:deptno,
4:salary]                        |
|                        Estimates: {row: 10, cpu: ?, memory: ?,
network: ?, cost: 100.04807692307692} |
|                     - EXCHANGE(SHUFFLE) [2]
|
|                            Estimates: {row: 10, cpu: ?, memory: ?,
network: ?, cost: 50.0}           |
|                        - SCAN [emps] => [2:deptno, 4:salary]
|
|                                Estimates: {row: 10, cpu: ?, memory:
?, network: ?, cost: 10.0}       |
|                                partitionRatio: 1/1, tabletRatio:
10/10                              |
|                     - EXCHANGE(SHUFFLE) [5]
|
|                            Estimates: {row: 5, cpu: ?, memory: ?,
network: ?, cost: 15.0}            |
|                        - SCAN [depts] => [5:deptno]
|
|                                Estimates: {row: 5, cpu: ?, memory:
?, network: ?, cost: 5.0}         |
|                                partitionRatio: 1/1, tabletRatio:
10/10                              |
|                                predicate: 6:name = 'R&D'
|
+-----------------------------------------------------------------------
---------------------------------+
```

В приведенном выше примере отсечение таблиц разрешено в Q1 и Q2, как показано в плане выполнения, в то время как Q3, чей предикат влияет на кардинальность конечных результатов, не выполняет отсечение таблиц.

6.15 JIT-компиляция для выражений

В этой теме описывается, как включить и настроить JIT-компиляцию для выражений в Selena.

6.15.1 Обзор

Just-in-time компиляция (JIT) — это генерация машинного кода и его выполнение во время работы программы. По сравнению с интерпретатором, JIT-компилятор может значительно улучшить эффективность выполнения наиболее часто используемых частей кода. Selena поддерживает JIT-компиляцию для определённых сложных выражений, что позволяет многократно улучшить производительность.

6.15.2 Использование

Начиная с версии v1.5.2, Selena включает JIT-компиляцию по умолчанию для узлов BE с лимитом памяти (настраивается параметром конфигурации BE `mem_limit`) 16 ГБ или более. Поскольку JIT-компиляция потребляет определённый объём ресурсов памяти, она отключена по умолчанию для узлов BE с памятью менее 16 ГБ.

Вы можете включить и настроить JIT-компиляцию для выражений, используя следующие параметры.

jit_lru_cache_size (конфигурация BE)

- По умолчанию: 0
- Тип: Int
- Единица измерения: Байты
- Изменяемый: Да
- Описание: Размер LRU-кэша для JIT-компиляции. Представляет фактический размер кэша, если установлено значение больше 0. Если установлено значение меньше или равное 0, система автоматически устанавливает кэш по формуле `jit_lru_cache_size = min(mem_limit*0.01, 1GB)` (при этом `mem_limit` узла должен быть не менее 16 ГБ).
- Добавлено в: -

jit_level (системная переменная)

- Описание: Уровень, на котором включена JIT-компиляция для выражений. Допустимые значения:
  - `1`: Система адаптивно включает JIT-компиляцию для компилируемых выражений.
  - `-1`: JIT-компиляция включена для всех компилируемых неконстантных выражений.

  - `0`: JIT-компиляция отключена. Вы можете отключить её вручную, если при использовании этой функции возникает ошибка.
- По умолчанию: 1
- Тип данных: Int
- Добавлено в: -

6.15.3 Поддержка функций

Поддерживаемые выражения

- `+`, `-`, `*`, `/`, `%`, `&`, `|`, `^`, `>>`, `<<`
- Преобразование типов данных с помощью CAST
- CASE WHEN
- `=`, `!=`, `>`, `>=`, `<`, `<=`, `<=>`
- `AND`, `OR`, `NOT`

Поддерживаемые операторы

- OLAP Scan Operator для фильтрации
- Projection Operator
- Aggregate Operator для выражений
- HAVING
- Sort Operator для выражений

Поддерживаемые типы данных

- BOOLEAN
- TINYINT
- SMALLINT
- INT
- BIGINT
- LARGEINT
- FLOAT
- DOUBLE

6.16 Skew Join V2

Skew Join V2 — это продвинутая функция оптимизации в Selena, которая решает проблемы перекоса данных (data skew) в операциях JOIN путём broadcast-передачи

«перекошенных» значений. Эта функция значительно улучшает производительность запросов при работе с сильно неравномерным распределением данных.

6.16.1 Обзор

Перекос данных возникает, когда определённые значения в столбцах объединения встречаются гораздо чаще других, что приводит к неравномерному распределению данных между узлами и вызывает узкие места производительности. Skew Join V2 решает эту проблему следующим образом:

1. **Определение перекошенных значений**: Ручное указание значений, вызывающих перекос данных
2. **Broadcast-передача перекошенных значений**: Передача этих конкретных значений на все узлы для обеспечения равномерного распределения данных
3. **Гибридное выполнение**: Использование комбинации shuffle- и broadcast-соединений для оптимальной производительности

Skew Join V2 создаёт гибридный план выполнения, который объединяет:

1. **Shuffle Join**: Для обычных данных используется стандартное соединение на основе shuffle
2. **Broadcast Join**: Для перекошенных значений данные правой таблицы передаются на все узлы

6.16.2 Включение Skew Join V2

По умолчанию оптимизация Skew Join V2 отключена, а её предшественник, Skew Join V1, включён. Если вы обнаружите, что производительность Skew Join V1 неудовлетворительна, необходимо отключить Skew Join V1 перед включением Skew Join V2.

```
-- Отключение Skew Join V1
SET enable_optimize_skew_join_v1 = false;

-- Включение Skew Join V2
SET enable_optimize_skew_join_v2 = true;
```

После включения Skew Join V2 вы можете обновить свои запросы на основе синтаксиса с явно указанными перекошенными значениями.

> **ПРИМЕЧАНИЕ**
>
> В настоящее время Skew Join V2 не поддерживает автоматическое переписывание плана на основе статистики. Поддерживается только ручное переписывание SQL на основе подсказок (hint).

6.16.3 Использование

По сравнению с методом переписывания запросов Skew Join V1, Skew Join V2 предлагает новый синтаксис, позволяющий явно указывать перекошенные значения с использованием Broadcast.

Синтаксис:

```
SELECT select_list FROM
table1 JOIN [skew|table1.column(skew_value1, skew_value2, ...)] table2
ON join_condition
[WHERE where_clause]
```

Параметры:

`[skew|table1.column(skew_value1, skew_value2, ...)]`: Подсказка skew. Она включает:

- `table1.column`: Столбец из левой таблицы, содержащий перекошенные значения.
- `skew_value1, skew_value2, ...`: Список значений через запятую, вызывающих перекос данных.

> **ПРИМЕЧАНИЕ**
>
> Не опускайте скобки в подсказке skew.

6.16.4 Примеры

Базовое использование

1. Создайте тестовые таблицы.

```
CREATE TABLE orders (
    order_id INT,
    customer_id INT,
    order_date DATE,
    amount DECIMAL(10,2)
) DUPLICATE KEY(order_id)
DISTRIBUTED BY HASH(order_id) BUCKETS 8;

CREATE TABLE customers (
    customer_id INT,
    customer_name VARCHAR(100),
    city VARCHAR(50)
) DUPLICATE KEY(customer_id)
DISTRIBUTED BY HASH(customer_id) BUCKETS 8;
```

2. Вставьте образцы данных с перекосом.

```
INSERT INTO orders VALUES
(1, 1001, '2024-01-01', 100.00),
(2, 1001, '2024-01-02', 200.00),
(3, 1001, '2024-01-03', 150.00),
(4, 1002, '2024-01-01', 300.00),
(5, 1003, '2024-01-01', 250.00);

INSERT INTO customers VALUES
(1001, 'John Doe', 'New York'),
(1002, 'Jane Smith', 'Los Angeles'),
(1003, 'Bob Johnson', 'Chicago');
```

3. Запросите данные с использованием Skew Join V2.

```
SELECT o.order_id, c.customer_name, o.amount
FROM orders o
JOIN [skew|o.customer_id(1001)] c
ON o.customer_id = c.customer_id;
```

Несколько перекошенных значений

Skew Join V2 поддерживает несколько перекошенных значений.

```
SELECT o.order_id, c.customer_name, o.amount
FROM orders o
JOIN [skew|o.customer_id(1001, 1002, 1003)] c
ON o.customer_id = c.customer_id;
```

Различные типы данных

Skew Join V2 поддерживает различные типы данных для перекошенных значений.

```
-- Строковые значения
SELECT t1.id, t2.name
FROM table1 t1
JOIN [skew|t1.category('electronics', 'clothing', 'books')] t2
ON t1.category = t2.category;

-- Значения даты
SELECT t1.id, t2.event_name
FROM events t1
JOIN [skew|t1.event_date('2024-01-01', '2024-01-02')] t2
ON t1.event_date = t2.event_date;

-- Числовые значения
SELECT t1.id, t2.region
FROM sales t1
JOIN [skew|t1.region_id(1, 2, 3)] t2
ON t1.region_id = t2.region_id;
```

Сложные условия соединения

Skew Join V2 поддерживает сложные условия соединения.

```
-- Условие соединения со сложными выражениями
SELECT t1.id, t2.value
FROM table1 t1
JOIN [skew|t1.key(abs(t1.key))] t2
ON abs(t1.key) = abs(t2.key);

-- Множественные условия соединения
SELECT t1.id, t2.name
FROM table1 t1
JOIN [skew|t1.category('electronics')] t2
ON t1.category = t2.category AND t1.region = t2.region;
```

Различные типы соединений

Skew Join V2 поддерживает различные типы соединений.

```
-- Left Join
SELECT t1.id, t2.name
FROM table1 t1
LEFT JOIN [skew|t1.category('electronics')] t2
ON t1.category = t2.category;

-- Left Semi Join
SELECT t1.id
FROM table1 t1
LEFT SEMI JOIN [skew|t1.category('electronics')] t2
ON t1.category = t2.category;

-- Left Anti Join
SELECT t1.id
FROM table1 t1
LEFT ANTI JOIN [skew|t1.category('electronics')] t2
ON t1.category = t2.category;
```

6.16.5 Лучшие практики

Шаг 1. Определите перекошенные значения

Перед использованием Skew Join V2 определите значения, вызывающие перекос данных:

```sql
-- Анализ распределения данных
SELECT customer_id, COUNT(*) as cnt
FROM orders
GROUP BY customer_id
ORDER BY cnt DESC
LIMIT 10;
```

Шаг 2. Выберите подходящие перекошенные значения

Следуйте этим правилам при выборе подходящих перекошенных значений:

- Включайте только наиболее частые значения, вызывающие значительный перекос.
- Избегайте включения слишком большого количества значений, так как это может увеличить накладные расходы на broadcast.
- Учитывайте компромисс между оптимизацией перекоса и стоимостью broadcast.

6.16.6 Ограничения

- **Поддерживаемые типы соединений** В настоящее время поддерживаются только INNER JOIN, LEFT JOIN, LEFT SEMI JOIN и LEFT ANTI JOIN.
- **Типы данных** В настоящее время поддерживаются только базовые типы данных (INT, BIGINT, STRING, DATE, DATETIME и строковые типы).
- **Сложные выражения** Поддержка сложных выражений в условиях соединения ограничена.
- **Таблица с перекосом** Skew Join V2 может обрабатывать только сценарии, когда большая таблица в операции JOIN имеет перекос, и эта большая таблица должна использоваться как левая таблица.
- **Переупорядочивание соединений** Использование подсказки `skew` предотвращает переупорядочивание соединений оптимизатором. Соединение будет выполняться в порядке, указанном в SQL, и оптимизатор не будет пытаться изменить порядок соединений или поменять местами левую и правую таблицы узла соединения, содержащего подсказку.

6.16.7 Устранение неполадок

Отладочная информация

Выполните EXPLAIN VERBOSE для запроса, чтобы получить его план выполнения:

```
EXPLAIN VERBOSE
SELECT ... FROM ... JOIN [skew|...] ...;
```

Проверьте следующие поля в плане:

- `SplitCastDataSink`: Разделение данных.
- `BROADCAST` и `SHUFFLE`: Тип распределения.
- `UNION`: Объединение результатов обоих типов соединений.

6.16.8 Связанные темы

- Планирование запросов
- Настройка профиля запросов
- Системные переменные
- Операции JOIN