Построение ETL и stage-таблицы

- Введение
- Что такое stage-таблицы
- Как точно не стоит делать
    - Неоптимальные пайплайны
    - Неоптимальные шаги в пайплайнах
        - Использовать ALTER DELETE/UPDATE
        - Использование pandas где это не нужно
        - Сотни или тысячи вставок в секунду
- Оптимальные пайплайны
    - Подробнее про каждый шаг
    - 2.1. Пример с полной перезаписью таблицы
    - 2.2. Пример с перезаписью двух партиций (Таблица партиционирована по месяцу)
    - 2.3. Пример с перезаписью части данных в таблице без партиционирования
    - 2.4. Пример с перезаписью части данных в партициях большой таблицы
    - 3. Использование нескольких stage-таблиц
- Запись в (Replicated)ReplacingMergeTree таблицу

Введение

При построении ETL процессов важно сохранять идемпотентность записи.

Под идемпотентностью понимается:

- Каждый перезапуск ETL процесса не должен создавать дубли в таблице или как-то ее ломать.
- Каждый перезапуск шага во время работы ETL процесса должен приводить к одному и тому же результату, даже если шаг был перезапущен после ошибки.

> Отдельный шаг можно воспринимать за таску в Airflow DAG, а ETL процесс за набор тасок в DAG.

Что такое stage-таблицы

Помочь в достижении идемпотентности ETL процессов могут таблицы в схеме **stage**.

Stage-таблица – промежуточная таблица, не предназначенная для длительного хранения данных. Должна хранить данные только во время ETL процесса.

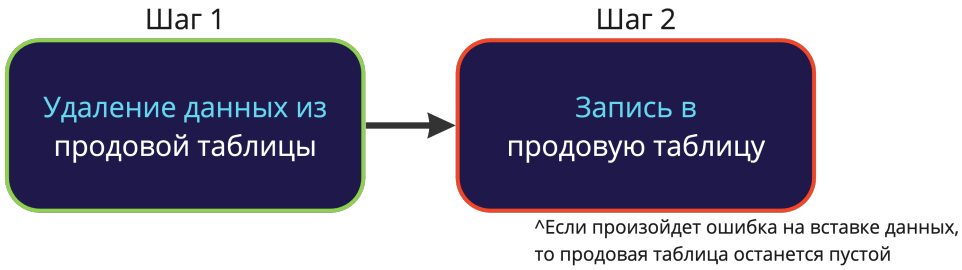
Stage-таблицы должны быть созданы с нереплицированным **MergeTree*** движком, но на всех нодах в шард-группе, т.е. **`ON CLUSTER`** `shard_group_old`.
Создавать stage-таблицу с **MergeTree*** движком нужно, чтобы, в случае отказа одной ноды, можно было без проблем переключить ETL процесс на другую ноду.
После завершения работы процесса stage-таблицу обязательно нужно очистить.

Про правила именования stage-таблиц читайте по ссылке. Краткая выдержка:

> **Формат названия stage-таблицы**
>
> `stage.<  >__<  >`
> 2 нижних подчеркивания
>
> Для сборки одной таблицы может быть создано несколько stage-таблиц.
> Формат нейминга в этом случае – stage.<схема>__<название>__<дополнительное осмысленное название промежуточного стейджа>
> 2 нижних подчеркивания 2 нижних подчеркивания

Как точно не стоит делать

Неоптимальные пайплайны



Проблема такого решения в том, что если произойдет ошибка на вставке данных (неправильный синтаксис, отвалилась сеть, и т. д.), то продовая таблица останется пустой.
Так как удаление данных уже произошло, а последующей вставки в продовую таблицу не произошло.

Неоптимальные шаги в пайплайнах

Использовать ALTER DELETE/UPDATE

`ALTER TABLE ... DELETE/UPDATE ...` это тяжелая операция для ClickHouse, не предназначенная для регулярного использования. Для того, чтобы выполнить эту операцию ClickHouse **полностью пересобирает файлы**, в которых находились удаленные /измененные строки. Подробнее.
Использование **`ALTER TABLE ... DELETE/UPDATE ...`** в пайплайнах запрещено.

Использование pandas где это не нужно

Пример

```
df = pd.read_sql_query(...) #    ClickHouse   Airflow.
df.to_sql(...) #  -  (  ,      SQL)     ClickHouse.
```

Здесь есть несколько проблем:

1. **Это медленно**. Данные нужно будет скачать из базы, после чего загрузить обратно, что ощутимо увеличит время выполнения этого шага, а также увеличит нагрузку на сеть.
2. Если данных в таблице будет много, то у контейнера, который выполняет код на питоне может не хватить оперативной памяти чтобы поместить датафрейм, что приведет к экстренному завершению процесса.

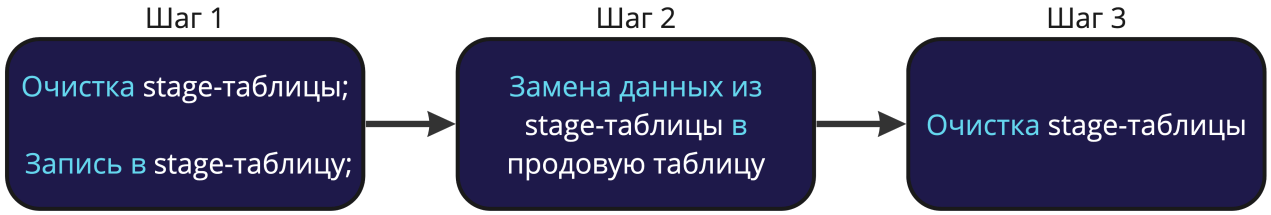
Сотни или тысячи вставок в секунду

Запрещено делать построчную вставку (1 INSERT – 1 строка) или множественную вставку маленьких кусков за короткий промежуток времени, например по 100 строк.

Движок *MergeTree очень плохо работает с сотнями/тысячами вставок в секунду, поскольку на каждую операцию вставки ClickHouse создает отдельный кусок данных, который ему нужно будет объединить с остальными кусками таблицы.

В пайплайнах стоит производить 1 раз большим батчем.

Оптимальные пайплайны



Для надежного пайплайна требуется stage-таблица, в случае ошибки на вставке данных в stage-таблицу, с продовой таблицей ничего не произойдет.
В общем случае, лучше иметь данные, которые актуальны на вчера, чем вообще никакие.

Подробнее про каждый шаг

На данном этапе необходимо осуществить очистку stage-таблицы перед наполнением. В случае сбоя в процессе вставки, при последующем перезапуске этого шага, строки, которые уже добавлены(упавшим инсертом) в таблицу, будут стёрты, и это предотвратит появление дублей stage-таблице.

Шаг 1. Очистка stage-таблицы и запись в stage-таблицу

```
--      /,     ,    .
-- , ,      ( ), ,            .
ALTER TABLE stage.prod_db__table_name DROP PARTITION '202401'

--      (  ).
TRUNCATE TABLE stage.prod_db__table_name

--      stage-.
INSERT INTO stage.prod_db__table_name (...)
SELECT ...

--
INSERT INTO stage.prod_db__table_name (...)
SELECT ...
WHERE <,     ,  >
```

Шаг 2. Замена данных из stage-таблицы в продовую таблицу

```
--    .
ALTER TABLE prod_db.table_name REPLACE PARTITION '202401' FROM stage.prod_db__table_name;
ALTER TABLE prod_db.table_name REPLACE PARTITION '202402' FROM stage.prod_db__table_name;

--     (  ).
ALTER TABLE prod_db.table_name REPLACE PARTITION tuple() FROM stage.prod_db__table_name
```

3. Очистка stage-таблицы

```
--      /,     ,    .
ALTER TABLE stage.prod_db__table_name DROP PARTITION '202401';
ALTER TABLE stage.prod_db__table_name DROP PARTITION '202402';

--      (  ).
TRUNCATE TABLE stage.prod_db__table_name
```

Поскольку данные в stage-таблицах больше не нужны, stage-таблицы обязательно очистить!

2.1. Пример с полной перезаписью таблицы

Шаг 1. Очистка stage-таблицы и запись в stage-таблицу

```
TRUNCATE stage.prod_db__table_name;

--
INSERT INTO stage.prod_db__table_name (...)
SELECT ...
```

Шаг 2. Замена данных из stage-таблицы в продовую таблицу

```
ALTER TABLE analytics.store_shipping_method
    REPLACE PARTITION tuple() FROM stage.analytics__store_shipping_method
```

Шаг 3. Очистка stage-таблицы

```
TRUNCATE TABLE stage.analytics__store_shipping_method
```

2.2. Пример с перезаписью двух партиций (Таблица партиционирована по месяцу)

Шаг 1. Очистка stage-таблицы и запись в stage-таблицу

```
ALTER TABLE stage.prod_db__table_name DROP PARTITION '202401';
ALTER TABLE stage.prod_db__table_name DROP PARTITION '202402';

--    2
INSERT INTO stage.prod_db__table_name (...)
SELECT ...
FROM ...
WHERE ...
```

Шаг 2. Замена данных в продовую таблицу

```
ALTER TABLE prod_db.table_name REPLACE PARTITION '202401' FROM stage.prod_db__table_name;
ALTER TABLE prod_db.table_name REPLACE PARTITION '202402' FROM stage.prod_db__table_name;
```

Шаг 3. Очистка stage-таблицы

```
ALTER TABLE stage.prod_db__table_name DROP PARTITION '202401';
ALTER TABLE stage.prod_db__table_name DROP PARTITION '202402';
```

2.3. Пример с перезаписью части данных в таблице без партиционирования

Бывает, что нужно каждый день перезаписывать данные за несколько предыдущих дней.
Например, если сегодня 2024-01-17, и нужно перезаписать данные за дни: 2024-01-14,2024-01-15, 2024-01-16, но таблица не партиционирована.

Шаг 1. Очистка stage-таблицы и запись в stage-таблицу

```
TRUNCATE stage.prod_db__table_name;


INSERT INTO stage.prod_db__table_name (...)
--    stage-.   ,      .
SELECT ...
FROM <- >
WHERE date BETWEEN toDate('2024-01-14') and toDate('2024-01-16')
UNION ALL
--       stage-.   ,      .
SELECT ...
FROM prod_db.table_name
WHERE date NOT BETWEEN toDate('2024-01-14') and toDate('2024-01-16')
```

Шаг 2. Замена данных в продовую таблицу

```
ALTER TABLE prod_db.table_name REPLACE PARTITION tuple() FROM stage.prod_db__table_name
```

Шаг 3. Очистка stage-таблицы

```
TRUNCATE stage.prod_db__table_name
```

2.4. Пример с перезаписью части данных в партициях большой таблицы

Если у вас таблица больше 5-10 GB, то нужно немного модифицировать пайплайн из пункта 2.3, чтобы в стейдже формировать только те партиции, которые будут изменены в проде.
Так **запись в стейдж будет в разы быстрее** и будет использовать меньше ресурсов.

Например, нам нужно перезаписать данные за 2024-06-31 (партиция 202406), 2024-07-01 (партиция 202407), 2024-07-02 (партиция 202407).

Шаг 1. Очистка stage-таблицы и запись в stage-таблицу

```
TRUNCATE stage.prod_db__table_name;


INSERT INTO stage.prod_db__table_name (...)
--    stage-.   ,      .
SELECT ...
FROM <- >
WHERE date BETWEEN toDate('2024-06-31') and toDate('2024-07-02')
UNION ALL
--       stage-.       ,      .
SELECT ...
FROM prod_db.table_name
WHERE True
    and toYYYYMM(date) BETWEEN toYYYYMM(toDate('2024-06-31')) AND toYYYYMM(toDate('2024-07-02')) --       ,    .
    and date NOT BETWEEN toDate('2024-06-31') and toDate('2024-07-02') --       ,     .
```

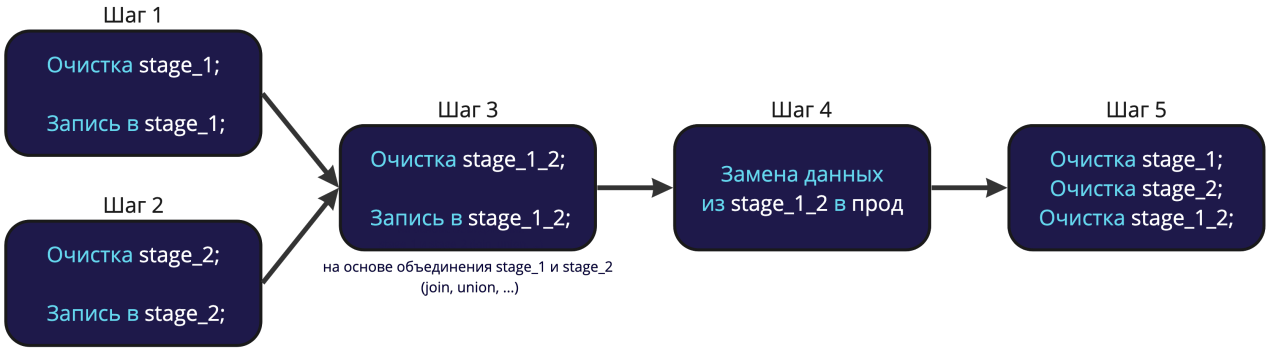
Шаг 2. Замена данных в продовую таблицу

```
ALTER TABLE prod_db.table_name REPLACE PARTITION tuple() FROM stage.prod_db__table_name
```

Шаг 3. Очистка stage-таблицы

```
TRUNCATE stage.prod_db__table_name
```

3. Использование нескольких stage-таблиц



Принципы этого подхода аналогичны пункту 2, можно перезаписывать всю таблицу, так и конкретные партиции.
Разбивать процесс на несколько стейджей оправдано, когда сборка целевой таблицы слишком объемна для одного запроса.
Промежуточные stage-таблицы используются для объединения их в единую конечную stage-таблицу, из которой происходит замена данных в продовую таблицу.

Запись в (Replicated)ReplacingMergeTree таблицу

Если получается выделить уникальный ключ, и строки в таблице не перезаписываются, а только добавляются – можно загружать данные в Replacing таблицу.
Но схлопывание дублей происходит в фоне в неизвестный момент времени (а может и не произойти), отсутствие дублей не гарантировано. Подробнее про Replacing
Это нужно учесть в архитектуре процесса. Например, делать **SELECT**-запросы с исключением дублей.