ClickHouse

Движки семейства MergeTree

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Ставим “+”, если все хорошо
“-”, если есть проблемы

Тема вебинара

Движки семейства MergeTree.

Цыкунов Алексей

Co-founder & CTO at Hilbert Team

- Более 20 лет опыта в проектировании и реализации отказоустойчивых и высоконагруженных информационных систем в таких отраслях как телеком и FinTech
- Автор курсов по Linux в Otus.ru
- Более 8 лет опыта оптимизации работы продуктовых команд и R&D департаментов с помощью DevOps инструментов и методик (Kubernetes, CI/CD, etc.) и облачных технологий (AWS, GCP, Azure, Yandex.Cloud)

Маршрут вебинара

- Знакомство
- Движок MergeTree
- Куски, слияние, первичный ключ
- Семейство *MergeTree

Цели вебинара

К концу занятия вы сможете

1. сформулировать принципы работы движка MergeTree
2. оптимально настраивать работу с этим движком
3. выбирать наиболее подходящий движок из семейства MergeTree для ваших задач
4. понимать разницу между кусками и партициями
5. понимать разницу между первичным ключом и ключом сортировки

Смысл

Зачем вам это уметь

1. Оптимальная работа с Clickhouse требует понимания принципов и алгоритмов работы движков таблиц
2.
3.

Движок MergeTree

Основные особенности движка MergeTree

- данные записываются по частям, маленькими кусочками (parts)
- данные объединяются в фоновом режиме.
  - процесс называется Merge (слияние)
- хранит данные, отсортированные по первичному ключу.
  - или ключу сортировки
- используется разреженный индекс, который не обеспечивает уникальности
- поддерживает репликацию данных.
  - с помощью движка ReplicatedMergeTree
- Поддерживает сэмплирование данных.
- Используется сжатие данных

Синтаксис CREATE TABLE

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [TTL expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2] [TTL expr2],
    ...
    INDEX index_name1 expr1 TYPE type1(...) GRANULARITY value1,
    INDEX index_name2 expr2 TYPE type2(...) GRANULARITY value2
) ENGINE = MergeTree()
ORDER BY expr
[PARTITION BY expr]
[PRIMARY KEY expr]
[SAMPLE BY expr]
[TTL expr
    [DELETE|TO DISK 'xxx'|TO VOLUME 'xxx' [, ...] ]
    [WHERE conditions]
    [GROUP BY key_expr [SET v1 = aggr_func(v1) [, v2 = aggr_func(v2) ...]] ] ]
[SETTINGS name=value, ...]
```

Аттрибуты таблицы

- ORDER BY - ключ сортировки, обязательный параметр
- PARTITION BY - ключ партицирования, например функция toYYYYMM, toYYYYMMDD от поля содержащего дату
- PRIMARY KEY - первичный ключ, если он отличается от ключа сортировки
- SAMPLE BY – выражение для сэмплирования.
- TTL - список правил, определяющих длительности хранения записей, а также задающих правила перемещения частей на определённые тома или диски спустя некоторое время
    - *SYSTEM [STOP|START] MOVES [[db.]tbl_name]*
- SETTINGS - раздел в котором можно указать настройки таблицы (гранулярность, расположение таблицы на диске определяемое параметром storage_policy и т.д.)

Аттрибуты полей

- MATERIALIZED <expression>. Значение не передается при вставке, а вычисляется и сохраняется как физический столбец
- ALIAS <expr>. Вычисляемое выражение, не сохраняется.
- Также в описании поля может быть указан кодек сжатия данных.
- TTL колонки определяет, когда значения будут сброшены в дефолтное состояние. Если срок действия всех значений в куске истек, то ClickHouse удаляет столбец из куска данных на файловой системе.
    - TTL не может быть применен к полям входящим в ключ сортировки
    - Отрабатывается при слияниях
    - *SYSTEM [STOP|START] TTL MERGES [[db.]tbl_name]*

Пример создания простой таблицы

```
DROP TABLE IF EXISTS otus_tbl;
CREATE TABLE otus_tbl
(
     id UInt64,
     name String
)
ENGINE MergeTree()
ORDER BY id;

INSERT INTO otus_tbl(id, name)
select number * number, 'data ' || toString(number),
FROM numbers(10);

-- продублируем вставку
```

Анализируем информацию по созданной таблице

```
select name, uuid, engine, data_paths, metadata_path
from system.tables
where database = 'default' and name = 'otus_tbl'
FORMAT Vertical;

SELECT *
FROM system.parts
WHERE (database = 'default') AND (table = 'otus_tbl')
ORDER BY name ASC
FORMAT Vertical

-------

ls -la /var/lib/clickhouse/data/default/
```

Хранение данных на диске. Куски.

- Кусок (part) - это директория с файлами, которая создается при каждом INSERT
  - формат имени: **partitionId_minBlock_maxBlock_mergeLevel_<mutation>**
- Куски могут объединяться с помощью слияний (merges)
  - в фоновом режиме
  - форсированно с помощью OPTIMIZE TABLE
- файлы внутри директории могут хранится в 2х форматах
  - wide - каждый столбец в отдельном файле
  - compact - все столбцы в одном файле
  - регулируется параметрами **min_rows_for_wide_part** и **min_bytes_for_wide_part**

Анализируем информацию по созданной таблице

```
root@clickhouse:~# ls -la /var/lib/clickhouse/data/default/otus_tbl/
total 24
drwxr-x--- 5 clickhouse clickhouse 4096 Jan  4 22:19 .
drwxr-x--- 3 clickhouse clickhouse 4096 Jan  4 22:15 ..
drwxr-x--- 2 clickhouse clickhouse 4096 Jan  4 22:15 all_1_1_0
drwxr-x--- 2 clickhouse clickhouse 4096 Jan  4 22:19 all_2_2_0
drwxr-x--- 2 clickhouse clickhouse 4096 Jan  4 22:15 detached
-rw-r----- 1 clickhouse clickhouse    1 Jan  4 22:15 format_version.txt
```

Анализируем отдельный кусок

```
root@clickhouse:~# ls -la /var/lib/clickhouse/data/default/otus_tbl/all_1_1_0/
total 44
drwxr-x--- 2 clickhouse clickhouse 4096 Jan  4 22:15 .
drwxr-x--- 5 clickhouse clickhouse 4096 Jan  4 22:19 ..
-rw-r----- 1 clickhouse clickhouse  261 Jan  4 22:15 checksums.txt
-rw-r----- 1 clickhouse clickhouse   63 Jan  4 22:15 columns.txt
-rw-r----- 1 clickhouse clickhouse    2 Jan  4 22:15 count.txt
-rw-r----- 1 clickhouse clickhouse  142 Jan  4 22:15 data.bin
-rw-r----- 1 clickhouse clickhouse   58 Jan  4 22:15 data.cmrk3
-rw-r----- 1 clickhouse clickhouse   10 Jan  4 22:15 default_compression_codec.txt
-rw-r----- 1 clickhouse clickhouse    1 Jan  4 22:15 metadata_version.txt
-rw-r----- 1 clickhouse clickhouse   50 Jan  4 22:15 primary.cidx
-rw-r----- 1 clickhouse clickhouse  151 Jan  4 22:15 serialization.json
```

Файлы при формате compact

- checksums.txt - содержит контрольные суммы всех файлов
- columns.txt - информация о колонках, имена и типы колонок в текстовом формате
- count.txt - число строк в куске
- data.bin - данные всех колонок в одном файле
- data.cmrk3 – файл засечек
- default_compression_codec.txt – содержит описание кодека сжатия по умолчанию
- primary.cidx – содержит данные первичного ключа в сжатом виде

Рассмотрим более полный пример

```
DROP TABLE IF EXISTS otus_wide;
CREATE TABLE otus_wide
(
    col_default UInt64 DEFAULT 42,
    col_materialized UInt64 MATERIALIZED col_default * 10,
    col_alias UInt64 ALIAS col_default + 1,
    col_codec String CODEC(ZSTD(10)),
    col_dt DateTime COMMENT 'Some comment',
    col_ttl UInt64 TTL col_dt + INTERVAL 1 MONTH
)
ENGINE MergeTree()
ORDER BY (col_default, col_dt)
PARTITION BY toYYYYMM(col_dt)
PRIMARY KEY (col_default)
SETTINGS
     min_bytes_for_wide_part = 0;
```

Произведем вставку

```
INSERT INTO otus_wide (
     col_default,
     col_codec,
     col_dt,
     col_ttl
)
     select
          number,
          'text value ' || toString(number),
          toDateTime('2023-11-01 00:00:00') + number * 3153600,
          rand(1) % 100000
     FROM numbers(10);
```

Проведем анализ содержимого кусков

```
SELECT
    name, partition, active,
    part_type, marks, rows
FROM system.parts WHERE table = 'otus_wide'

┌─name──────────┬─partition─┬─active─┬─part_type─┬─marks─┬─rows─┐
│ 202310_1_1_0  │ 202310    │      0 │ Wide      │     2 │    1 │
│ 202310_1_1_1  │ 202310    │      1 │ Wide      │     2 │    1 │
│ 202311_2_2_0  │ 202311    │      0 │ Wide      │     2 │    1 │
│ 202311_2_2_1  │ 202311    │      1 │ Wide      │     2 │    1 │
│ 202312_3_3_0  │ 202312    │      1 │ Wide      │     2 │    1 │
│ 202401_4_4_0  │ 202401    │      1 │ Wide      │     2 │    1 │
│ 202402_5_5_0  │ 202402    │      1 │ Wide      │     2 │    1 │
│ 202403_6_6_0  │ 202403    │      1 │ Wide      │     2 │    1 │
│ 202405_7_7_0  │ 202405    │      1 │ Wide      │     2 │    1 │
│ 202406_8_8_0  │ 202406    │      1 │ Wide      │     2 │    1 │
│ 202407_9_9_0  │ 202407    │      1 │ Wide      │     2 │    1 │
```

Партиции

- Определяются произвольным ключом партиционирования
    - не рекомендуется делать слишком гранулированным
- Оптимизируют работу с данными
- Имеется большой набор операций для манипуляций с партициями
    - https://clickhouse.com/docs/ru/sql-reference/statements/alter/partition
- Директория detached содержит куски, отсоединенные от таблицы с помощью запроса DETACH. Поврежденные куски также попадают в эту директорию – они не удаляются с сервера.

Проанализируем влияние партиций на выборку

```
explain description=1, indexes=1 select * from otus_wide where col_dt='2024-07-19';

┌─explain─────────────────────────────────────────────────┐
│ Expression ((Projection + Before ORDER BY))             │
│   ReadFromMergeTree (default.otus_wide)                 │
│   Indexes:                                              │
│     MinMax                                              │
│       ...                                               │
│     Partition                                           │
│       Keys:                                             │
│         toYYYYMM(col_dt)                                │
│       Condition: (toYYYYMM(col_dt) in [202407, 202407]) │
│       Parts: 1/1                                        │
│       Granules: 1/1
```

Проведем анализ содержимого кусков

```
# ls -c1 /var/lib/clickhouse/store/e4e/e4e639e7-b3ff-4119-95a3-6086698fac26/202408_10_10_0/
checksums.txt
col_codec.bin
col_codec.cmrk2
col_default.bin
col_default.cmrk2
col_dt.bin
col_dt.cmrk2
col_materialized.bin
col_materialized.cmrk2
col_ttl.bin
col_ttl.cmrk2
columns.txt
count.txt
default_compression_codec.txt
metadata_version.txt
minmax_col_dt.idx
```

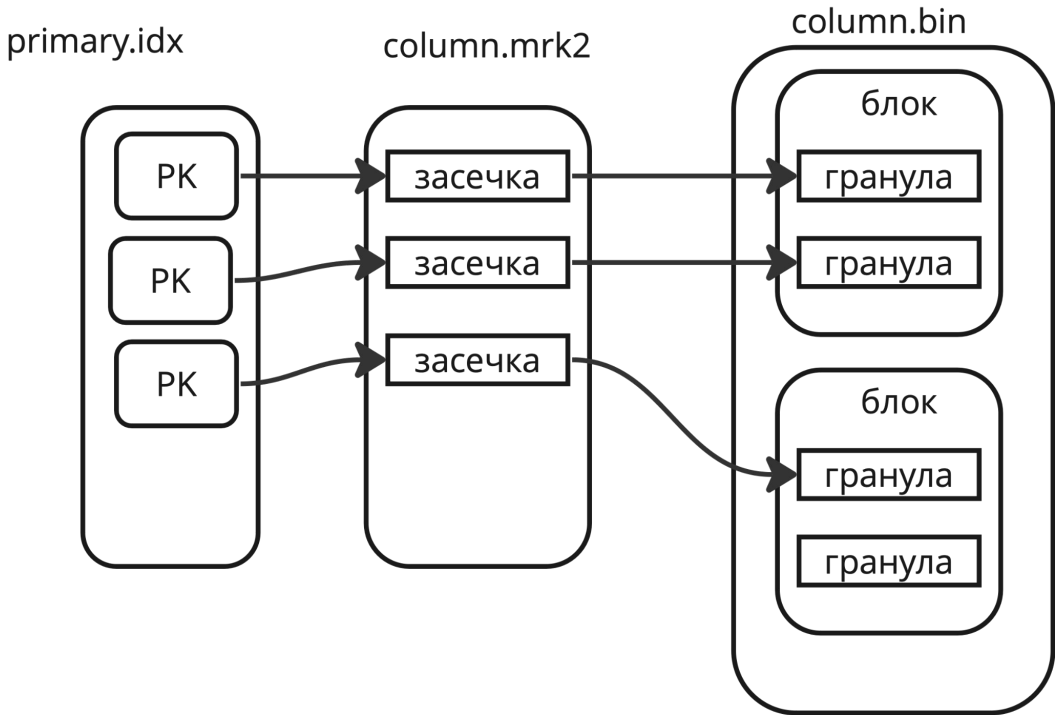
Формат wide

- отдельный файл .bin для каждого столбца
  - данные разбиты на блоки отсортированные по первичному ключу
  - блок содержит данные в сжатом виде
  - блок состоит из гранул
  - гранула - минимальный неделимый набор данных, считываемый при выборке
    - размер гранул ограничен настройками
    - первая строка гранулы помечается засечкой
- отдельный файл засечек .cmrk2 для каждого столбца хранит засечки в виде
  - смещение блока в bin-файле
  - смещение в разжатом блоке
  - количество строк в грануле.

Ключ сортировки, первичный ключ

- ключ сортировки должен быть обязательно указан
- первичный ключ может быть не указан
    - первичным ключом будет ключ сортировки
    - если указан то должен быть префиксом кортежа ключа сортировки
    - имеет смысл при использовании движков SummingMergeTree и AggregatingMergeTree.
- первичный ключ не обеспечивает уникальности
- позволяет поиск по любой части ключа
- для хранения используется разреженный индекс в файле primary.cidx
    - каждое хранимое значение адресует засечку
    - количество значений равно количеству гранул

Ключ, засечки, гранулы



Слияние

- Процесс слияния запускается в фоновом режиме для оптимизации хранения
- Один кусок участвует только в одном слиянии
- Куски из разных партиций не сливаются вместе
- Обработанные куски становятся неактивными
- Неактивные куски впоследствии удаляются из файловой системы и system.parts
- Чтение происходит только из активных кусков
    - поэтому слияние не мешает чтению
- Ключ сортировки является ключом для слияния и является первичным ключом по умолчанию.

OPTIMIZE TABLE

```
OPTIMIZE TABLE [db.]name [ON CLUSTER cluster]
     [PARTITION partition | PARTITION ID 'partition_id']
     [FINAL]
[DEDUPLICATE [BY expression]]
SETTINGS [
     alter_sync = [0, 1, 2],
     optimize_throw_if_noop = [0, 1]
]
```

Внеплановое слияние

- создает внеочередное задание на merge
- по умолчанию не оповещается если merge выполнить не удалось
  - регулируется опцией optimize_throw_if_noop.
- PARTITION - только для одной партиции
- DEDUPLICATE - удаляет полные дубли строк
  - либо по набору BY
- FINAL - принудительное слияние даже при наличии только одного куска
  - не рекомендуется
- слияние на репликах регулируется опцией alter_sync

Практика: слияния

Настройки MergeTree

- можно указать для каждой таблицы при создании
- select * from system.merge_tree_settings
- документация

Стоит обратить внимание

- гранулярность
- политики хранения

Вопросы для проверки

1. Что такое кусок?
2. Уникальны ли значения в первичном ключе?
3. Чем отличается первичный ключ от ключа сортировки?
4.

Семейство *MergeTree

- - "под капотом" движок MergeTree для хранения данных
- - SummingMergeTree
- - AggregatingMergeTree
- - ReplacingMergeTree
- - CollapsingMergeTree(sign)
- - VersionedCollapsingMergeTree
- - GraphiteMergeTree

- - Replicated*MergeTree

Движок SummingMergeTree

SummingMergeTree

- группировка по **первичному ключу**
- **суммируются** значения в полях, заданных как **параметры движка**
- все остальные поля агрегируются выбором произвольного значения
- группировка происходит во время слияний

Практический пример

```
CREATE TABLE summing_otus
(
id UInt32,
val UInt32,
val2 UInt32
)
ENGINE = SummingMergeTree(val)  -- сумма будет применяться к полю val
ORDER BY (id); -- записи по этому ключу будут группироваться
```

Практический пример

```
INSERT INTO summing_otus
SELECT 1, (number + 1) * 10, number from numbers(10);
INSERT INTO summing_otus SELECT 1, 100, 5;
SELECT * FROM summing_otus;
SELECT * FROM summing_otus FINAL;
OPTIMIZE TABLE summing_otus;
-- не забываем следить за тем, что происходит с кусками
select event_time, event_type, part_name, partition_id
from system.part_log
where table = 'summing_otus';
```

Движок AggregatingMergeTree

AggregatingMergeTree

- группировка по **первичному ключу**
  - который может быть задан отдельно и быть меньше ключа сортировки
- **агрегируются** значения в полях с типами данных
  - AggregateFunction - хранит промежуточное состояние
    - Для вставки данных используйте **INSERT SELECT** с агрегатными **-State**-функциями.
    - При выборке данных из таблицы **AggregatingMergeTree**, используйте **GROUP BY** и те же агрегатные функции, что и при вставке данных, но с суффиксом **-Merge**.
    - для чтения сырых данных используйте **finalizeAggregation**
  - SimpleAggregateFunction - хранит текущее состояние

Практический пример

```
CREATE TABLE aggr_otus ( id UInt64,
val_uniq AggregateFunction(uniq, UInt64),
val_max AggregateFunction(maxIf, String, UInt8),
val_avg AggregateFunction(avg, UInt64) )
ENGINE=AggregatingMergeTree
ORDER BY id;

-- произведем вставку
INSERT INTO aggr_otus
SELECT 1,
uniqState(toUInt64(rnd)),
maxIfState(toString(rnd),rnd%2=0),
avgState(toUInt64(rnd))
FROM (SELECT rand(1) % 100000 as rnd from numbers(10));
```

Практический пример

```
select * from aggr_otus;
select finalizeAggregation(val_uniq),
finalizeAggregation(val_avg),
finalizeAggregation(val_max)  from aggr_otus

select uniqMerge(val_uniq),
avgMerge(val_avg),
maxIfMerge(val_max)  from aggr_otus
-- повторим вставку и выборку
```

Движок ReplacingMergeTree

ReplacingMergeTree

- подходит в случае необходимости частых обновлений
- удаляет дублирующиеся записи с одинаковым значением **ключа сортировки**.
- остается самая последняя строка из самой последней вставки.
  - или с наибольшей версией
- `ENGINE = ReplacingMergeTree([ver [, is_deleted]])`
  - ver - столбец с номером версии (необязательный параметр).
  - is_deleted - для управления удалением (необязательный параметр).
- удаление дублирующихся записей происходит при слиянии
  - Некоторая часть данных может оставаться необработанной.
- подходит для хранения данных без дубликатов, но не гарантирует отсутствие дубликатов при чтении без модификатора FINAL.

Практический пример

```
CREATE TABLE replacing_otus
(
id UInt32,
val UInt32
)
ENGINE = ReplacingMergeTree
ORDER BY (id);

INSERT INTO replacing_otus SELECT 1, (number + 1) * 10 from numbers(3);
INSERT INTO replacing_otus SELECT 2, (number + 1) * 100 from numbers(3);

SELECT * FROM replacing_otus

INSERT INTO replacing_otus SELECT 1, 100;
INSERT INTO replacing_otus SELECT 2, 100;
```

Движки CollapsingMergeTree и VersionedCollapsingMergeTree

CollapsingMergeTree(sign)

- подходит в случае необходимости частых обновлений
  - удаляем предыдущее значение с помощью **sign = -1**
  - вставляем новое значение с помощью sign = 1
- `ENGINE = CollapsingMergeTree(sign)`
  - удаляет строки с одинаковыми значениями ключа сортировки, но разными `sign (1 и -1)`
  - за значениями `sign` должно следить приложение
  - строки без пары сохраняются
  - удаление происходит во время слияний

Практический пример

```
CREATE TABLE table_collapsing
(
id UInt32, pageViews UInt8,
duration UInt8, sign Int8
)
ENGINE = CollapsingMergeTree(sign)
ORDER BY id;

INSERT INTO table_collapsing VALUES (1, 7, 100, 1);
INSERT INTO table_collapsing VALUES (1, 7, 100, -1), (1, 8, 150, 1);

SELECT * FROM table collapsing;
SELECT * FROM table_collapsing FINAL;

OPTIMIZE TABLE table collapsing;
SELECT * FROM table_collapsing;
```

VersionedCollapsingMergeTree

- подходит в случае необходимости частых обновлений
  - удаление с помощью sign оптимизирует хранение
  - версионирование позволяет работать с многопоточной вставкой
- ENGINE = VersionedCollapsingMergeTree(sign, version)
  - удаляет каждую пару строк, которые имеют один и тот же первичный ключ + версию и разный sign

Практический пример

```
CREATE TABLE table_versioned_collapsing
(
id UInt32,
pageViews UInt8,
duration UInt8,
sign Int8,
version UInt8
)
ENGINE = VersionedCollapsingMergeTree(sign, version)
ORDER BY id;

INSERT INTO table versioned collapsing VALUES (1, 7, 100, 1, 1);
INSERT INTO table_versioned_collapsing VALUES (1, 7, 100, -1, 1), (1, 8, 150, 1, 2);

SELECT * FROM table_versioned_collapsing FINAL;
```

Движок GraphiteMergeTree

Graphite rollup

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    Path String,
    Time DateTime,
    Value <Numeric_type>,
    Version <Numeric_type>
    ...
) ENGINE = GraphiteMergeTree(config_section)
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[SETTINGS name=value, ...]
```

Graphite rollup

```
DROP TABLE IF EXISTS GraphiteTable;
CREATE TABLE GraphiteTable
(
Path String,
Time DateTime,
Value Float64,
Timestamp UInt64
)
ENGINE = GraphiteMergeTree('graphite_rollup_example')
ORDER BY (Path,Time);

INSERT INTO GraphiteTable SELECT
'my metric', toDateTime('2023-10-01 00:00:00') + number*10,
toFloat64(randUniform(5.5, 10)), 1 FROM numbers(10000);

SELECT * from GraphiteTable;
```

GraphiteMergeTree

- подходит для настройки прореживания и агрегирования/усреднения (rollup) данных Graphite
- В таблице должны быть столбцы для следующих данных:
  - Название метрики. Тип данных: String.
  - Время измерения метрики. Тип данных DateTime.
  - Значение метрики. Тип данных: любой числовой.
  - Версия метрики. Тип данных: любой числовой
- Названия столбцов указываются в конфиге для rollup, либо используются имена по умолчанию
  - *Path, Time, Value, Timestamp*
- конфиг для rollup указывается в config.xml

Слайд с домашним заданием

1. описание по ссылке
2.
3.
4.

Сроки выполнения:

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Рефлексия

С какими впечатлениями уходите с вебинара?

Как будете применять на практике то,
что узнали на вебинаре?

Заполните, пожалуйста, опрос о занятии по ссылке в чате