Базы данных
Лекция 9

ClickHouse

Мгер Аршакян

План лекции

1. Основы ClickHouse
2. Структура и проектирование БД

Основы ClickHouse

Принципы и задачи ClickHouse

ClickHouse — **колоночная** СУБД, она не имеет накладных расходов на работу с широкими таблицами с большим количеством столбцов.

ClickHouse подходит

1. для быстрых запросов по неагрегированной статистике
2. для хранения и чтения логов
3. для хранения телеметрии, метрик по клиентам и устройствам

Когда не нужна ClickHouse

1. Сложные запросы с использованием JOIN:
    - JOIN между двумя большими таблицами может потребовать перемещения большого количества данных между узлами кластера
    - JOIN-операции могут потребовать большого объёма памяти
2. Выборки отдельных строк таблиц с ожидаемой низкой задержкой ответа.
3. Профиль нагрузки, в котором данные часто изменяются и обновляются в режиме, приближенном к реальному времени.

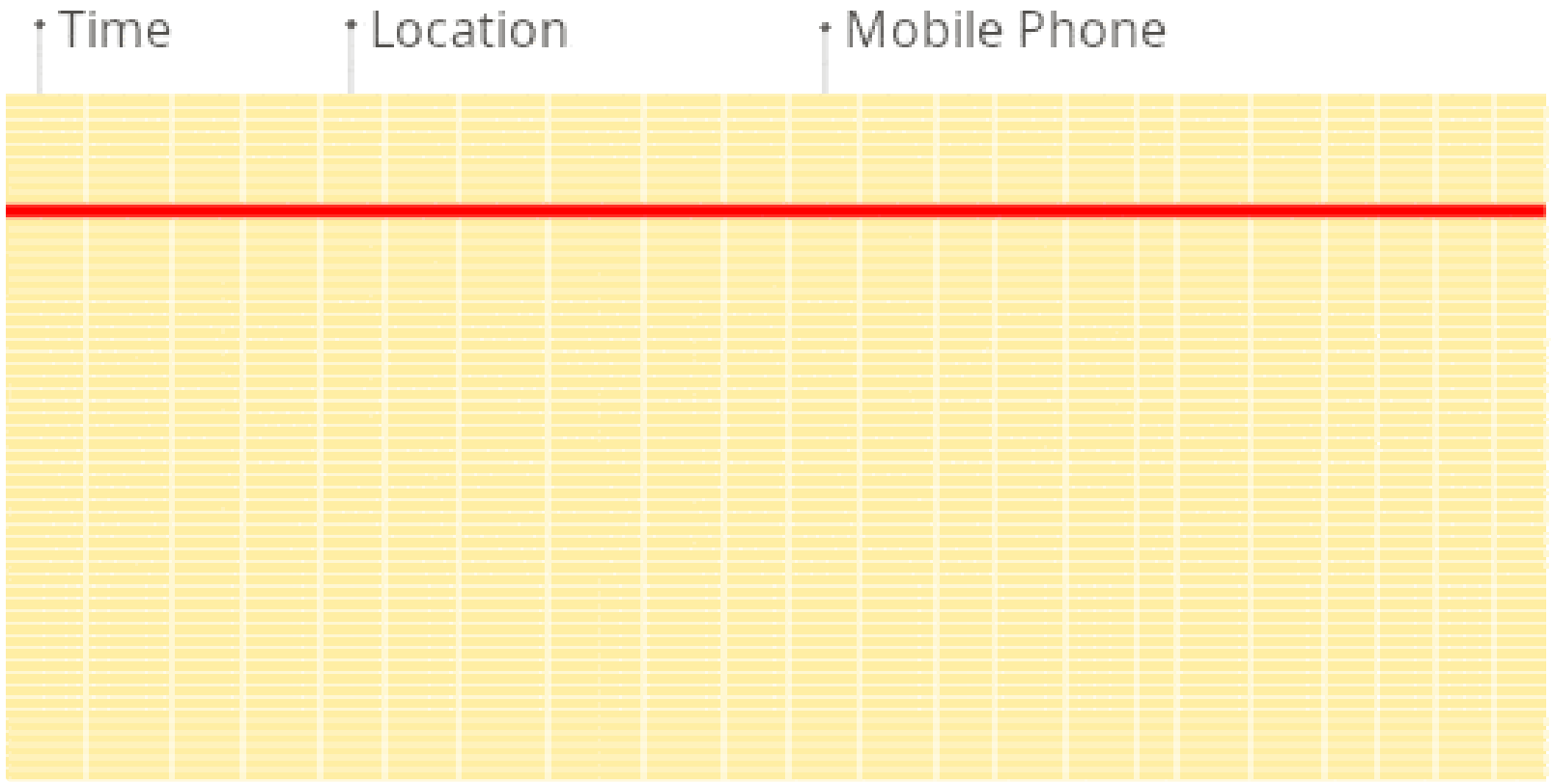
Ключевые особенности ClickHouse

1. **ClickHouse** является **колоночной СУБД**
2. Большие запросы **распараллеливаются**
3. **ClickHouse** может работать как в **однонодовой**, так и в **кластерной** конфигурации
4. Отсутствию единой точки отказа **ClickHouse** (**Асинхронная multi-master репликация**)
5. **ClickHouse** совершает аналитическую обработку данных в **реальном времени**
6. **ClickHouse** поддерживает несколько движков таблиц, предназначенных для разных сценариев обработки данных

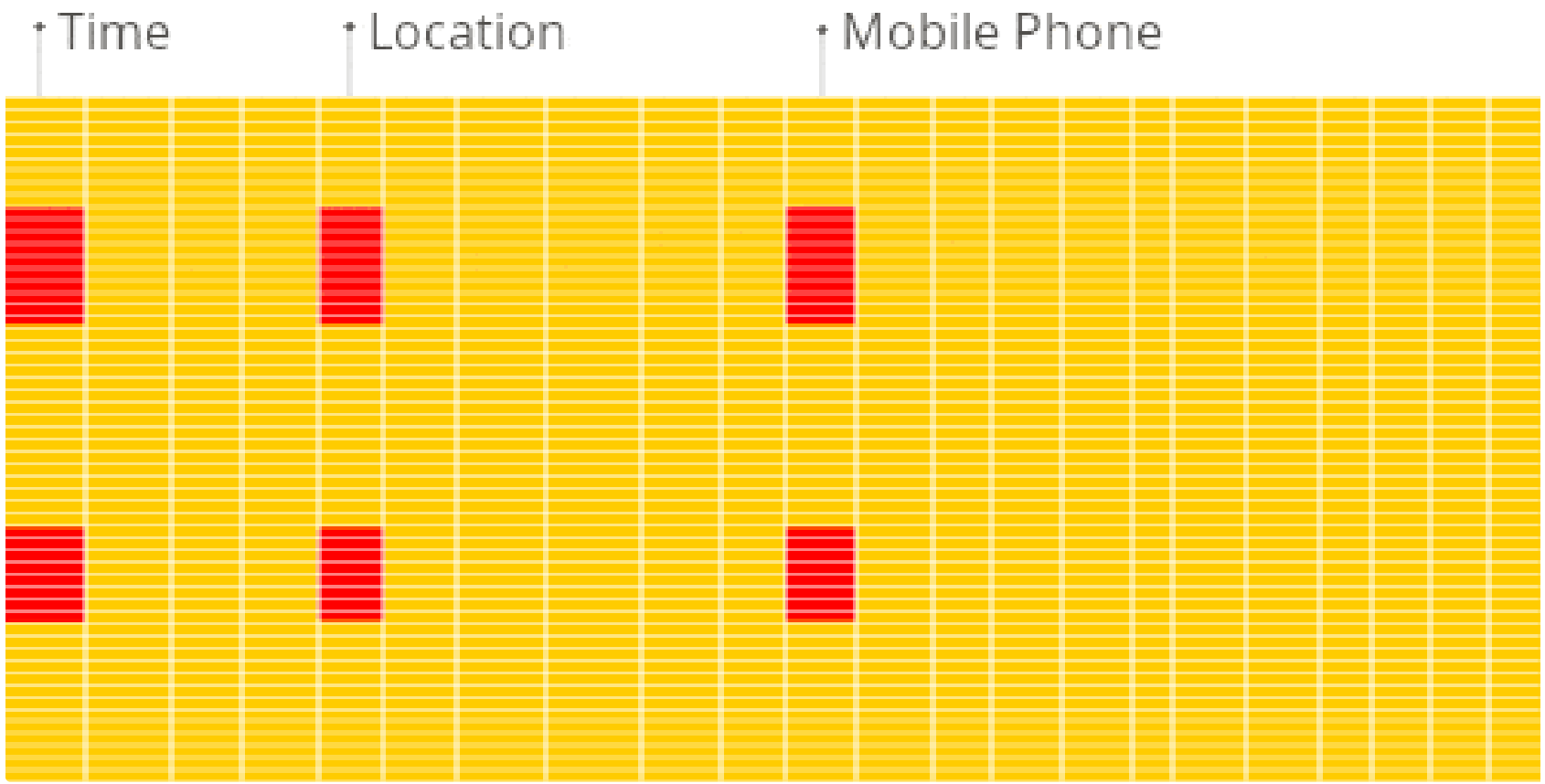
Что такое столбцовая база данных?

1. Столбцовая база данных хранит данные каждой колонки независимо
2. С диска читаем только те данные которые используются в запросе

Традиционная система, ориентированная на строки



Столбцовая



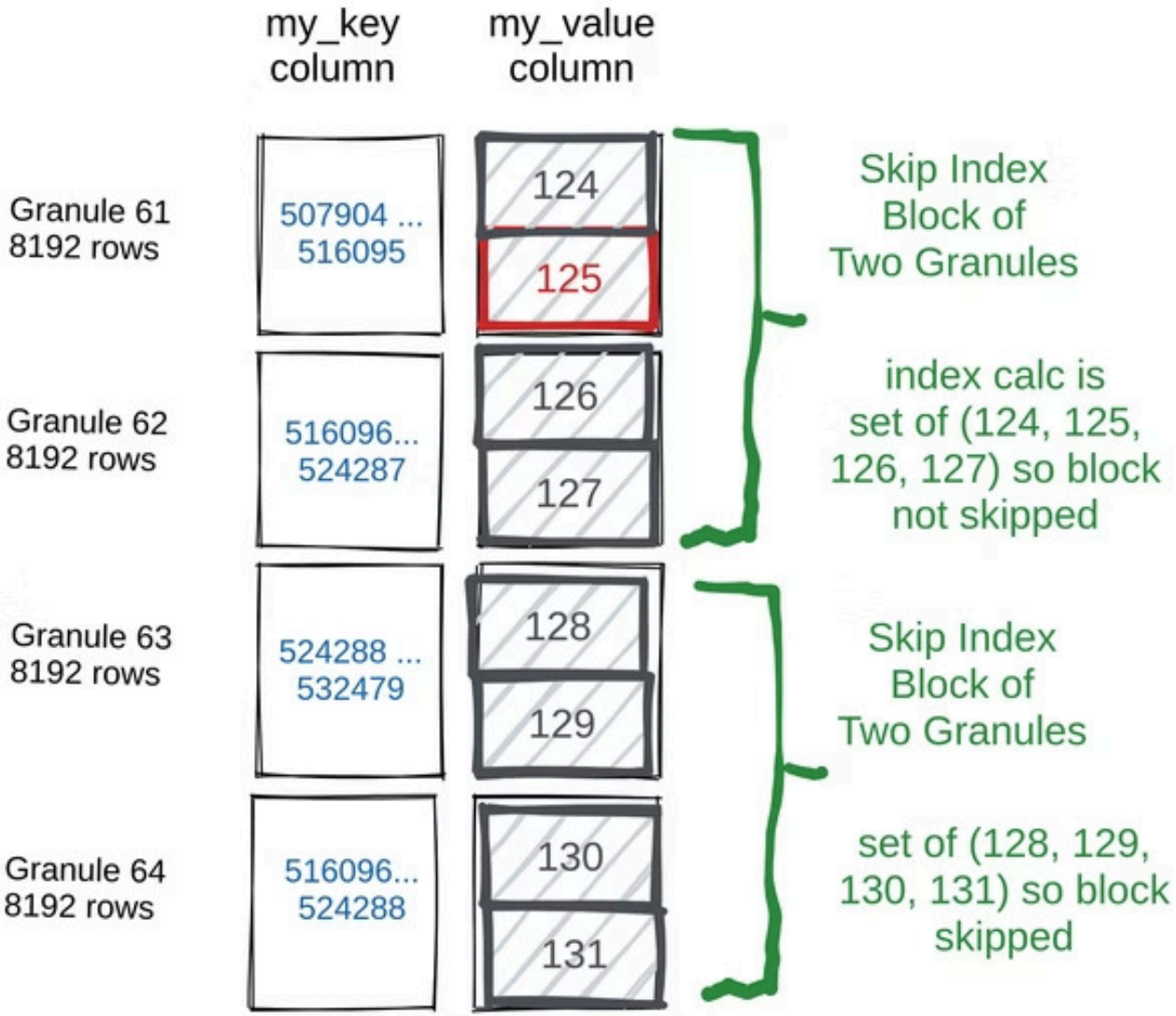
Ключевыми преимуществами столбцовой базы данных являются

1. Запросы, которые используют только несколько колонок из многих.
2. Агрегирующие запросы к большим объемам данных.
3. Сжатие данных по колонкам.

Структура базы данных

Разреженные индексы

В таком индексе хранятся не все значения, он дробит диапазон на блоки по N записей и сохраняет только значение индекса в начале каждого диапазона


my_key column
my_value column
Granule 61
8192 rows
507904 ...
516095
124
125
Granule 62
8192 rows
516096...
524287
126
127
Granule 63
8192 rows
524288 ...
532479
128
129
Granule 64
8192 rows
516096...
524288
130
131
Skip Index
Block of
Two Granules
index calc is
set of (124, 125,
126, 127) so block
not skipped
Skip Index
Block of
Two Granules
set of (128, 129,
130, 131) so block
skipped

Преимущества

1. Почти всегда помещается в оперативную память и позволяет работать с большим количеством строк в таблицах.
2. Всегда работает лучше, чем full scan.

Недостатки

1. На одну таблицу приходится только один разреженный индекс
2. Плохо работает для точечных поисковых запросов для нахождения одной записи
3. Разреженный индекс допускает чтение лишних строк
4. Индекс не обеспечивает уникальность

Движки таблиц ClickHouse

Движок в ClickHouse — это механизм, который определяет

1. **Где хранятся данные** (на диске, в памяти, удаленно)
2. **Как данные организованы** (сортировка, партиции, индексы)
3. **Какие операции поддерживаются** (INSERT, SELECT, UPDATE, DELETE)
4. **Как происходит репликация** и распределение
5. **Производительность** и оптимизации

ENGINE	SELECT	INSERT	DELETE	UPDATE	persistent	indexes
*MergeTree	+	+	+	+	+	+
*Log	+	+	-	-	+	-
EmbeddedRocksDB	+	+	-	-	+	-
URL	+	-/+	-	-	external	-
Buffer	+	+	-	-	+	-
Memory	+	+	+	+	-	-
Set	(only IN)	+	-	-	+	-
Join	+	+	+	-	+	-
PostgreSQL	+	+	-	-	external	-
Kafka	+	+	-	-	external	-

Семейство MergeTree

Движки этого семейства в ClickHouse используются чаще всего.

Они хранят данные на диске в сжатом формате и поддерживают операторы DML — `SELECT`, `INSERT`, `DELETE`, `UPDATE`. Поля этих таблиц можно проиндексировать. Это единственное семейство движков, поддерживающее индексы пропуска данных.

Семейство Log

С помощью таблиц семейства **Log** можно записывать небольшое количество данных — до одного миллиона строк — в большое количество таблиц с последующим массовым чтением записей.

Модификация `UPDATE` и удаление `DELETE` записанных данных и индексы в таблицах этого семейства не поддерживаются.

Движок EmbeddedRocksDB

ClickHouse имеет встроенный движок **RocksDB** для хранения и обработки данных key-value.

RocksDB является форком от **Google LevelDB** и поддерживает транзакционность, **снапшоты**, **фильтры Блума**, TTL для ключей и многое другое. На данный момент в RocksDB поддерживается чтение `SELECT` и вставка данных `INSERT`.

Движок URL

Движок URL позволяет управлять данными на удалённом сервере.

Запросы `INSERT` и `SELECT` транслируются в `POST` и `GET`. Модификация данных не поддерживается. Движок не хранит данные локально.

```
CREATE TABLE exchange_rates (date Date, currency String, rate Float64)
ENGINE = URL('https://example.com/rates.csv', 'CSV');
--------------
SELECT * FROM exchange_rates WHERE currency = 'USD'
```

Движок Buffer

Движок Buffer накапливает записываемые данные в ОЗУ и при срабатывании некоторых пороговых значений сбрасывает в таблицу-приёмник.

Если инстанс базы данных перезапускается, данные в Buffer не исчезают.

Данные накапливаются, но при этом хранятся на диске, а не в памяти.

Движок Memory

Движок Memory позволяет читать `SELECT`, записывать `INSERT`, модифицировать `UPDATE` и удалять `DELETE` данные.

Если инстанс базы данных перезапускается, данные очищаются. Индексы не поддерживаются.

Движок Set

Движок Set предназначен для использования в правой части оператора `IN`.

Обычный `SELECT` из такой таблицы недоступен. В **Set**-таблицу записываются данные для последующей подстановки в условия отбора с использованием оператора `IN`.

```
CREATE TABLE banned_users (user_id UInt64) ENGINE = Set();

SELECT * FROM users WHERE user_id IN banned_users;

-- Работает быстрее чем IN с указанными значениями

SELECT * FROM users
WHERE user_id IN (123, 456, 789, 999, 1001, 1002, ...тысячи значений...);
```

Движок Join

Движок Join предназначен, чтобы предварительно готовить данные для использования в операциях `JOIN`.

Данные целиком прогреты и готовы к соединению и фильтрации. В Join-таблице можно читать `READ`, записывать `INSERT` данные и удалять `DELETE`, обновление данных в таблице недоступно.

Движок PostgreSQL

Движок PostgreSQL позволяет читать `SELECT` данные из PostgreSQL, а также вставлять `INSERT` записи. Этот движок является интерфейсом для доступа к внешней таблице

Движок Kafka

Движок Kafka не хранит данные самостоятельно и предназначен для подписки на потоки данных и публикации данных.

Семейство MergeTree

1. **MergeTree**
2. **ReplacingMergeTree**
3. **SummingMergeTree**
4. **AggregatingMergeTree**
5. **CollapsingMergeTree**
6. **VersionedCollapsingMergeTree**

Прочитать и посмотреть примеры: **https://clickhouse.com/docs/ru/engines/table-engines/mergetree-family**

Простые и материализованные представления ClickHouse

Принцип работы

- **Обычное VIEW**: запрос выполняется каждый раз
- **Materialized VIEW**: результат сохраняется и обновляется автоматически

Антипаттерны написания SQL-запросов

Использование `JOIN`

Работа с `JOIN` в ClickHouse отличается от других СУБД.

Большое количество `JOIN` при работе с ClickHouse — это всегда сложный сценарий, поскольку требует большого объёма ресурсов. И он может усложняться соединением распределённых таблиц.

В таких случаях производительность запроса может снизиться на порядки.

Частые запросы `SELECT` по одной записи из одной таблицы

Индекс в ClickHouse разреженный, поэтому данные будут считываться блоками.

Запросы большого размера

Запрос размером в сотни мегабайт к распределённой таблице сильно увеличит сетевой трафик и снизит производительность, поскольку будет передаваться между узлами в несжатом виде.

Например, при запросе размером 100 МБ в распределённую таблицу, состоящую из 50 узлов-исполнителей, передастся около 5 ГБ данных, а время выполнения запроса будет определяться пропускной способностью сети.

Запросы без использования первичного ключа

Когда данных в таблице мало, то запросы без первичного ключа не будут создавать проблем. Когда данных станет на порядки больше, то вырастет время выполнения и повысится утилизация дисковой подсистемы.

Краткий чек-лист

✅ Не соединяйте много таблиц с помощью `JOIN`, тем более когда эти таблицы распределённые.

✅ Не делайте частые запросы `SELECT`, извлекающие по одной записи.

✅ Не делайте запросы в сотни мегабайт к распределённым таблицам, чтобы не создавать нагрузку по трафику.

✅ Используйте первичный ключ в условиях запросов к таблицам MergeTree.

✅ Не используйте поля строкового типа для группировок, чтобы не создавать избыточную нагрузку ОЗУ.

✅ Не используйте агрегатные функции для набора данных, который можно предварительно отфильтровать.

Антипаттерны проектирования

Маленькие батчи для вставки

Не следует ожидать высокой скорости вставки, если батч содержит менее 1000 записей.

Использование атрибута LowCardinality

Бывает так, что поля таблицы, содержащие низкокардинальные данные, объявлены без атрибута **LowCardinality.** Это приводит к ненужной нагрузке при хранении и обработке данных.

Поля без кодеков сжатия

Доступ к сжатым данным всегда быстрее, чем к несжатым. Поля без кодека сжатия — плохая практика. Все данные должны быть сжаты.

Спасибо за внимание!

Беседа курса в Telegram