Yandex Cloud

Разгоним запросы: как быстро готовить ClickHouse

Кузьма Лешаков
Yandex Cloud

Кузьма Лешаков

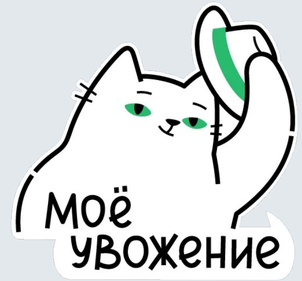
≈ 8 лет в IT

2023 — Приблизительно год в Yandex Cloud
Команда Data Platform

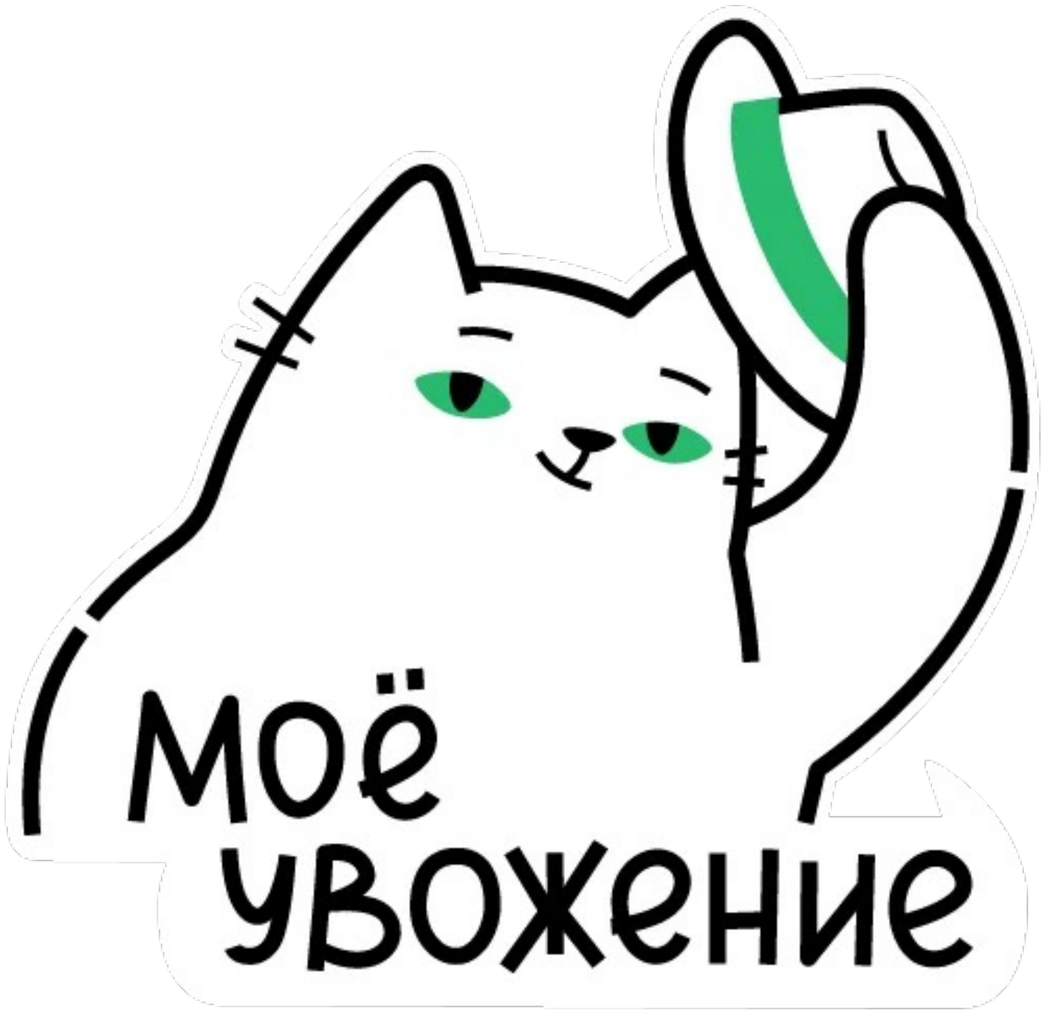
2021 — Uma.Tech.
Ведущий инженер данных

2019 — Clover Group
Инженер данных

2017 — The Linux Foundation
Стажёр-разработчик



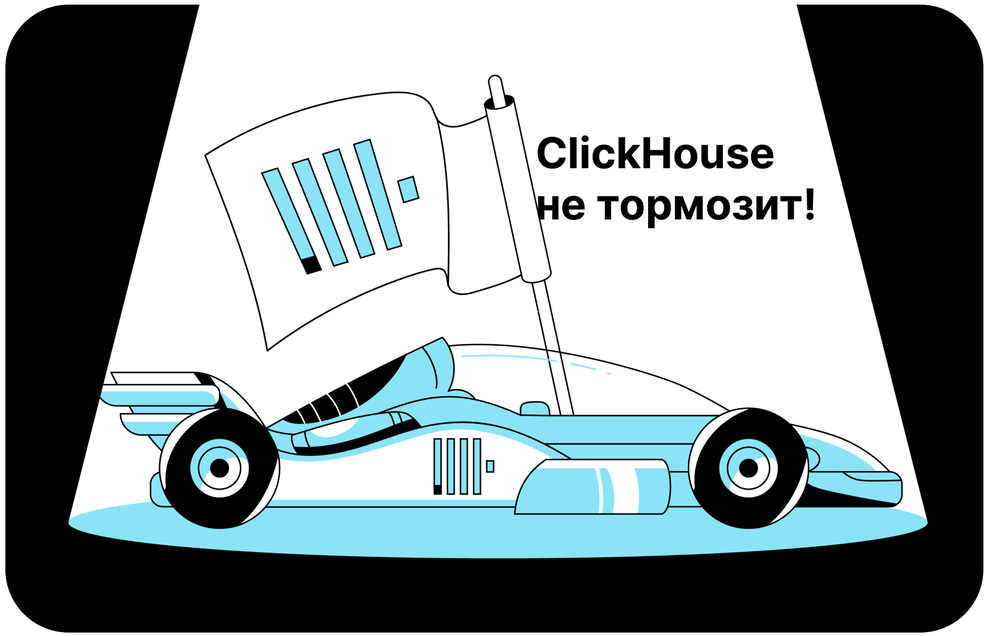
Котик-помощник



О чём поговорим

1. Индексации
2. Проекции
3. Шардирование
4. Итоги

Ключевые особенности ClickHouse



Железо

2 шарда и в каждом по 1 хосту:

```
s3-c8-m32 (8 vCPU, 100% vCPU rate, 32 ГБ RAM),
1.03 ТБ network-ssd
```

Датасет

100 000 000 записей

≈ 52 GB — до сжатия

≈ 13 GB — после сжатия

1. Индексы
2. Проекции
3. Шардирование

Индексы

- Первичный индекс
- Индексы пропуска данных
- Инвертированные*

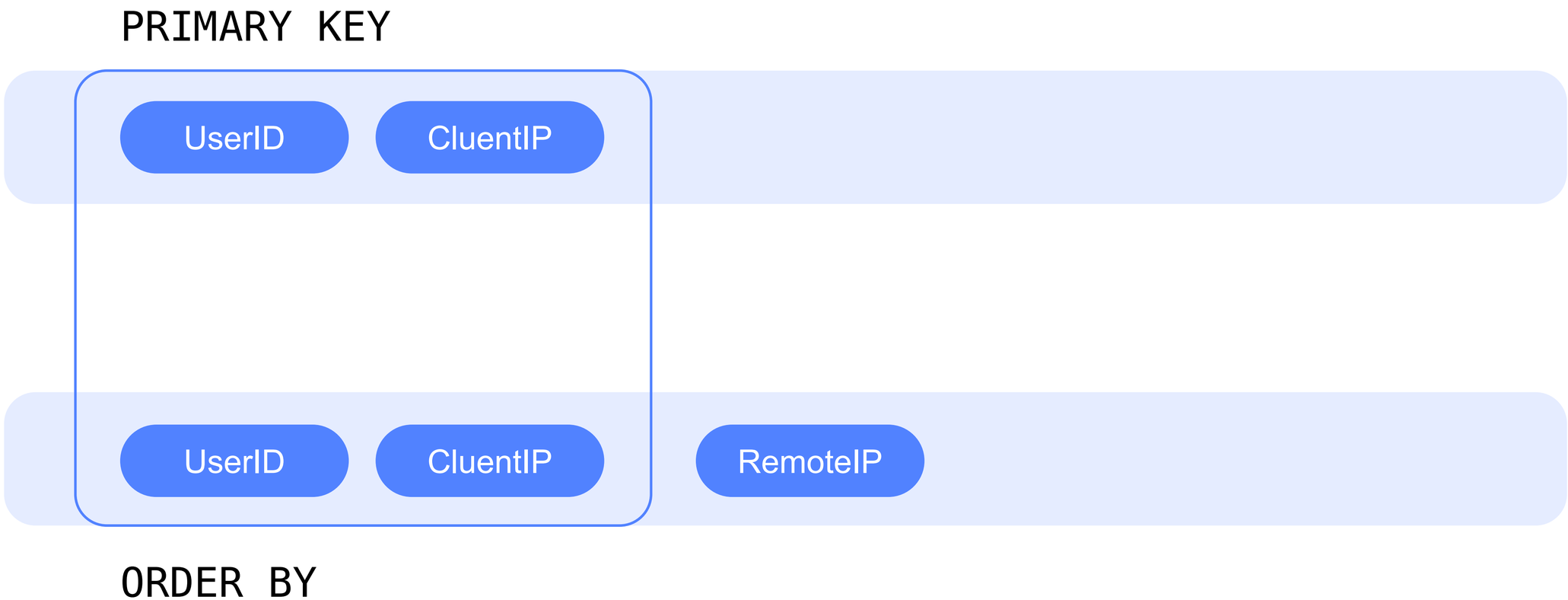
Какой индекс?

```
CREATE TABLE hits_100m
(
    `ClientIP`            UInt32,
    `UserID`              UInt64,
    `WindowClientWidth`   UInt16,
    `WindowClientHeight`  UInt16,
    …
)
ENGINE = MergeTree
ORDER BY (UserID, ClientIP);
```

Ответ

- Нет индекса
- ClientIP
- UserID, ClientIP
- Случайный выбор из доступных колонок

PRIMARY KEY VS ORDER BY



Где посмотреть?

system.tables

```
SELECT *
FROM system.tables
WHERE table = 'hits_100m';
```

system.data_skipping_indices

```
SELECT *
FROM system.data_skipping_indices
WHERE table = 'hits_100m';
```

Где посмотреть?

```
EXPLAIN indexes = 1
SELECT *
FROM hits_100m
WHERE UserID = 2428136496412786849
```

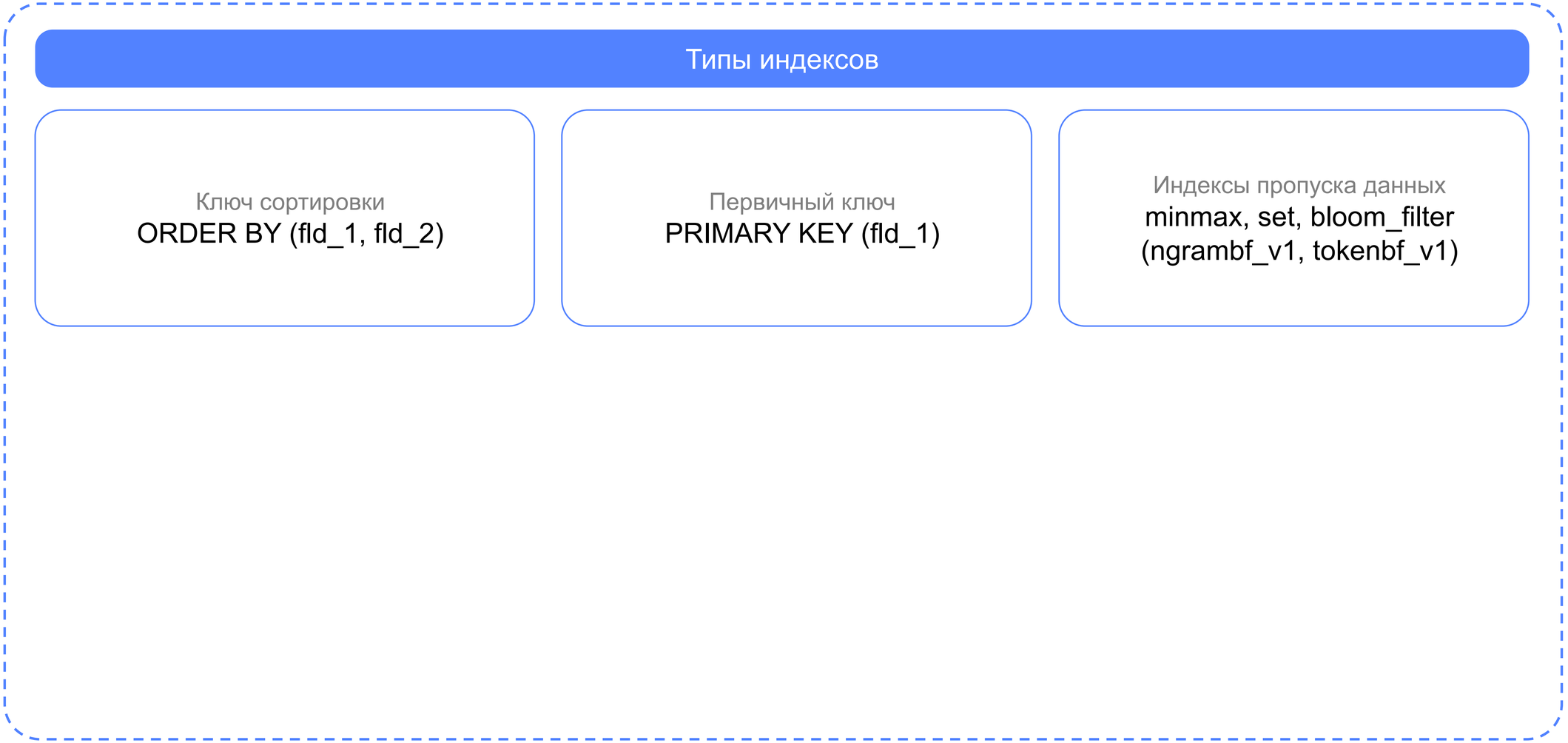
explain

```
Expression ((Projection + Before ORDER BY))
    ReadFromMergeTree (datasets.hits_100m)
    Indexes:
      PrimaryKey
        Keys:
          UserID
        Condition: (UserID in [242…9, 242…9]
        Parts: 1/1
        Granules: 1/12341
```

Где посмотреть?

```
SET send_logs_level = 'trace';
```

Типы индексов



Первичный индекс

Движок БД

ENGINE	SELECT	INSERT	DELETE	UPDATE	persistent	indexes
*MergeTree	⊕	⊕	⊕	⊕	⊕	⊕
*Log	⊕	⊕	⊖	⊖	⊕	⊖
EmbeddedRocksDB	⊕	⊕	⊖	⊖	⊕	⊖
URL	⊕	⊕⊖	⊖	⊖	external	⊖
Buffer	⊕	⊕	⊖	⊖	⊕	⊖
Memory	⊕	⊕	⊕	⊕	⊖	⊖
Set	(only in)	⊕	⊖	⊖	⊕	⊖
Join	⊕	⊕	⊕	⊖	⊕	⊖
PostgreSQL	⊕	⊕	⊖	⊖	external	⊖
Kafka	⊕	⊕	⊖	⊖	external	⊖

MergeTree

```
CREATE TABLE hits_100m
(
    `ClientIP`           UInt32,
    `UserID`             UInt64,
    `WindowClientWidth`  UInt16,
    `WindowClientHeight` UInt16,
    …
)
ENGINE = MergeTree
ORDER BY (UserID, ClientIP)
SETTINGS index_granularity = 8192;
```

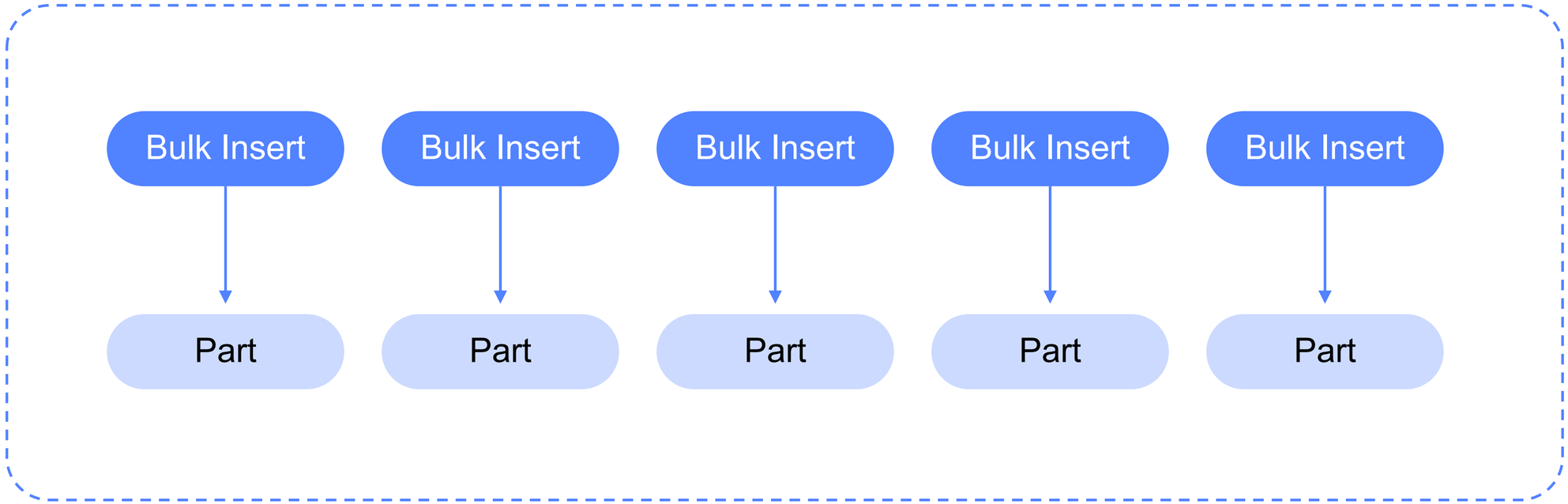
Первичный индекс

Запомним терминологию

- **Гранула (granule)** – логическая разбивка строк внутри блока до компрессии; по умолчанию – 8192 строки
- **ORDER BY** – порядок сортировки таблицы
- **Первичный ключ (Primary Key)** –значения первой строки указанных колонок в порядке сортировки
- **Первичный индекс (Primary Index)** – индекс, хранящий в памяти значения первичных ключей каждой гранулы из файла PRIMARY.idx
- **Кусок данных (data part)** – директория файлов, состоящая из файлов колонок и файла индекса подмножества данных таблицы

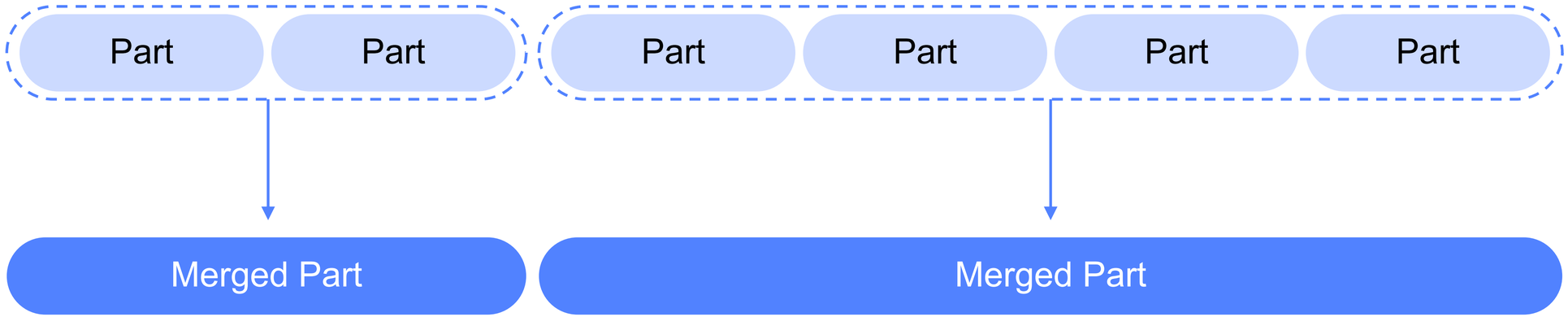
Первичный индекс

Каждая вставка создаёт кусок



Первичный индекс

- Со временем куски могут быть объединены в фоновом режиме
- Поэтому движок и называется **MergeTree**



Первичный индекс

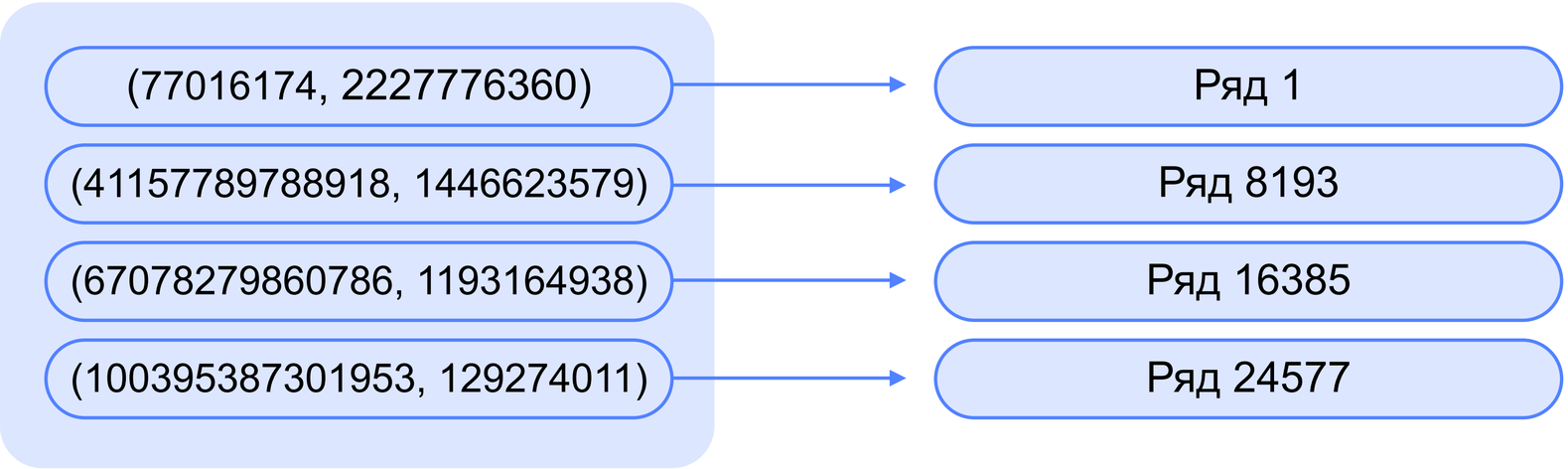
- Файлы колонок отсортированы по ORDER BY
- Каждая колонка в куске имеет неизменяемый (immutable) файл с данными колонки

Директория куска

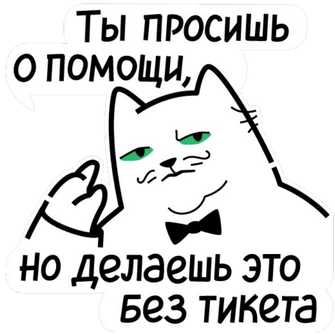
Primary.idx	UserID.bin	ClientIP.bin	URL.bin
(77016174, 2227776360)	77016174	2227776360	http://kinopoisk.ru
	77016174	2227776360	http://kinopoisk.ru/express-226330839&s_yers=20010116
	217463585	3520361063	http://kinopoisk.ru/search?film-Findex.ru/real
	5322209069	758512731	http://tiensmed.ru/real-estate/out-of-type=flatTypeId
	12947021928	3760850764	http://smeshariki.ru/region
	14665400046	3063077662	http://topic98603531343e0a3c852a7fc7365016411246/?from]=&int[15412899.Calips tv велики прохождение

Первичный индекс

1. Первичный индекс разрежен
2. Файл primary.idx содержит одну запись на каждую гранулу
   - По умолчанию каждая запись соответствует 8192 ряду или 10 мб данных
   - Это разреженное поведение позволяет ключам храниться в памяти



Первичный индекс




```
SELECT *
FROM hits_100m
WHERE UserID = 2428136496412786849;
```

```
2 rows in set. Elapsed: 0.823 sec. Processed 100.00 million rows,
801.86 MB (121.54 million rows/s., 974.60 MB/s.)
```

Ключ UserID, ClientIP

```
SELECT *
FROM hits_100m
WHERE UserID = 2428136496412786849;
```

```
2 rows in set. Elapsed: 0.079 sec. Processed 8.16 thousand rows, 6.26
MB (103.33 thousand rows/s., 79.29 MB/s.)
```

Размер первичного индекса 145 KiB

Преимущества

- Почти всегда помещается в оперативную память и позволяет работать с большим количеством строк в таблицах (является и ограничением)
- Поддерживает сложные логические выражения
- Всегда работает лучше, чем full scan
- Работает даже на части составного индекса, невзирая на его позицию в индексе

Недостатки

- На одну таблицу приходится только один разреженный индекс
- Не лучшие результаты работы в точечных lookup-запросах — поисковых запросах для нахождения одной записи
- Не обеспечивает уникальность

Индексы пропуска данных

1. bloom filter
2. minmax
3. set

Индексы пропуска данных

1. bloom filter
2. minmax
3. set

bloom filter

Hash_func1()
Hash_func2()
Hash_func3()

Input_value	Hash_func1	Hash_func2	Hash_func3
‘carrot’	5	7	3
‘pickup’	2	0	5

array_bloom(10)

1		1	1		1		1		

bloom filter

```
ALTER TABLE hits_100m
    ADD INDEX blm_filter ClientIP_2 TYPE bloom_filter GRANULARITY 4;
```

```
ALTER TABLE hits_100m MATERIALIZE INDEX blm_filter;
```

bloom filter

Колонка – ClientIP. Для быстрого определения наличия IP клиента в спам-списке

```
SELECT *
FROM hits_100m
WHERE ClientIP = 2087437801;
```

```
4 rows in set. Elapsed: 0.093 sec. Processed 100.00 million rows, 405.94 MB
(1.08 billion rows/s., 4.38 GB/s.)
```

```
SELECT *
FROM hits_100m
WHERE ClientIP_2 = 2087437801;
```

```
4 rows in set. Elapsed: 0.067 sec. Processed 32.77 thousand rows, 6.07 MB
(492.35 thousand rows/s., 91.24 MB/s.)
```

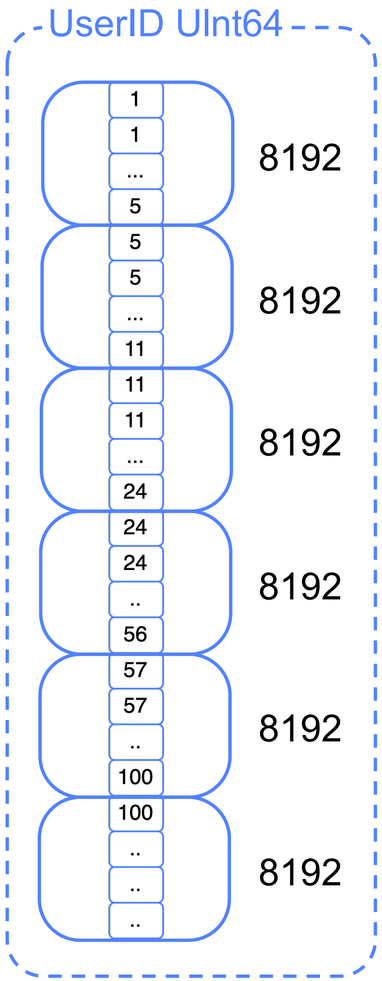
Размер индекса:

```
74 MB (compressed), 100 MB (compressed)
```

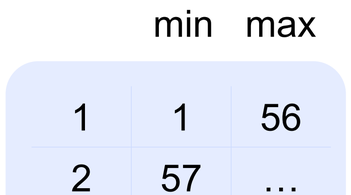
Индексы пропуска данных

1. bloom filter
2. minmax
3. set

minmax



Type minmax
GLANULARITY 4



	min	max
1	1	56
2	57	…

minmax

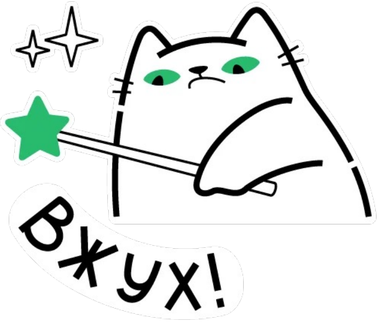
```
ALTER TABLE hits_100m
    ADD INDEX minmax_idx ResponseEndTiming_2 TYPE minmax GRANULARITY 10;

ALTER TABLE hits_100m MATERIALIZE INDEX minmax_idx;
```

minmax

Primary Key – ResponseStartTiming. **minmax index** – ResponseEndTiming_2.
Чем выше первое значение, тем выше второе

```
SELECT count(*)
FROM hits_100m_minmax
WHERE ResponseEndTiming = 1911
```

```
> 1 row in set. Elapsed: 0.070 sec. Processed 100.00 million rows, 400.00 MB (1.43 billion rows/s., 5.71 GB/s.)
```




```
SELECT count(*)
FROM hits_100m_minmax
WHERE ResponseEndTiming_2 = 1911
```

```
> 1 row in set. Elapsed: 0.057 sec. Processed 65.56 million rows, 262.24 MB (1.15 billion rows/s., 4.59 GB/s.)
```

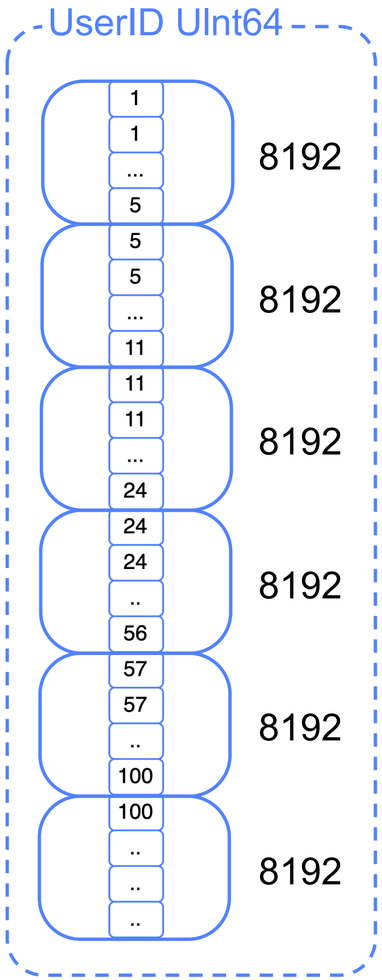
Размер индекса:

uncompressed 0,01 MB, compressed 0.005 MB

Индексы пропуска данных

1. bloom filter
2. minmax
3. set

set



Type set (0)
GLANULARITY 4

1	1, 2, 3, 5, 6, 7, 11, 24, 26, 28, 56
2	57, 58, 59, 100…

set

```
ALTER TABLE hits_100m
      ADD INDEX set_idx BrowserCountry_2 TYPE set(100) GRANULARITY 2;

ALTER TABLE hits_100m MATERIALIZE INDEX set_idx;
```

set

Primary Key – ClientIP. **SET index** – BrowserCountry_2. Небольшое количество стран, скорее всего, имеет корреляцию с IP-адресами клиентов

```
SELECT count(*)
FROM hits_100m_sets
WHERE BrowserCountry IN ('qg', 'YD')
```

```
> 1 row in 1 row in set. Elapsed: 0.188 sec. Processed 100.00 million rows,
200.01 MB (532.62 million rows/s., 1.07 GB/s.)
```

```
SELECT count(*)
FROM hits_100m_sets
WHERE BrowserCountry_2 IN ('qg', 'YD');
```

```
> 1 row 1 row in set. Elapsed: 0.026 sec. Processed 41.91 thousand rows,
167.68 KB (1.64 million rows/s., 6.54 MB/s.)
```

Размер индекса:

uncompressed 0,24 MB, compressed 0.14 MB

Преимущества

- ⊕ Ускоряют запросы
- ⊕ Нет ограничений по количеству

Недостатки

- ⊖ Замедляют процесс вставки новых данных
- ⊖ Занимают дополнительное место
- ⊖ В большинстве случаев требуют корреляции с первичным ключом и вдумчивого подхода при использовании

1. Индексы
2. Проекции
3. Шардирование

Проекции

- Выполнение запросов по колонке, не являющейся частью первичного ключа
- Преагрегационные колонки, которые снизят потребление как вычислительных, так и IO-ресурсов

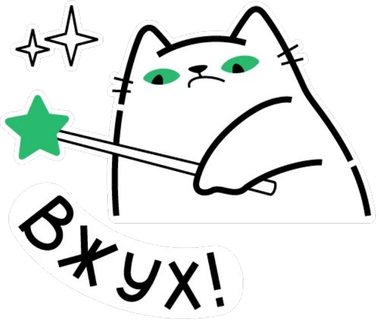
Проекции

Primary Key – (UserID, ClientIP). Поиск по **RemoteIP IN (3596690355, 2983581987);**

```
SELECT *
FROM hits_100m
WHERE RemoteIP IN (3596690355, 2983581987);
```

```
> 2 rows in set. Elapsed: 0.176 sec. Processed 100.00 million rows, 406.11 MB (569.78 million rows/s., 2.31 GB/s.)
```

```
ALTER TABLE hits_100m
    ADD PROJECTION remote_ip_projection (SELECT * ORDER BY RemoteIP);
ALTER TABLE hits_100m
    MATERIALIZE PROJECTION remote_ip_projection;
```




```
SELECT *
FROM hits_100m
WHERE RemoteIP IN (3596690355, 2983581987);
```

```
> 2 rows in set. Elapsed: 0.048 sec. Processed 16.38 thousand rows, 12.64 MB (342.23 thousand rows/s., 263.98 MB/s.)
```

Размер проекции практически идентичен оригинальной таблице

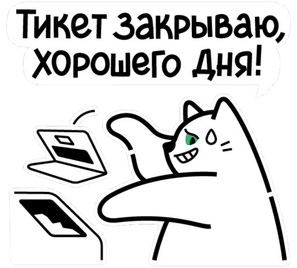
Проекции

Primary Key – (UserID, ClientIP). Посчитаем среднее по количеству хитов на пользователя

```
SELECT UserID, count(*)
FROM hits_100m
GROUP BY UserID;
```

```
> 17630976 rows in set. Elapsed: 14.501 sec. Processed 100.00 million rows, 800.00 MB
(6.90 million rows/s., 55.17 MB/s.)
```

```
ALTER TABLE hits_100m
    ADD PROJECTION hits_avg_user_prj (SELECT UserID, count(*) GROUP BY UserID);
ALTER TABLE hits_100m
    MATERIALIZE PROJECTION hits_avg_user_prj;
```




```
SELECT UserID, count(*)
FROM hits_100m
GROUP BY UserID;
```

```
> 17630976 rows in set. Elapsed: 8.909 sec. Processed 17.63 million rows, 537.69 MB
(1.98 million rows/s., 60.35 MB/s.)
```

Проекции

1. Автоматически
2. force_optimize_projection=1

Преимущества

- Помогают ускорить запрос при использовании агрегаций или изменении ключа сортировки

Недостатки

- Занимают схожее количество места, как и оригинальные таблицы

Проекции

Другие схожие опции с индексированием

- Две таблицы
- Материализованное представление

1. Индексы
2. Проекции
3. Шардирование

Шардирование

- Distributed

Distributed

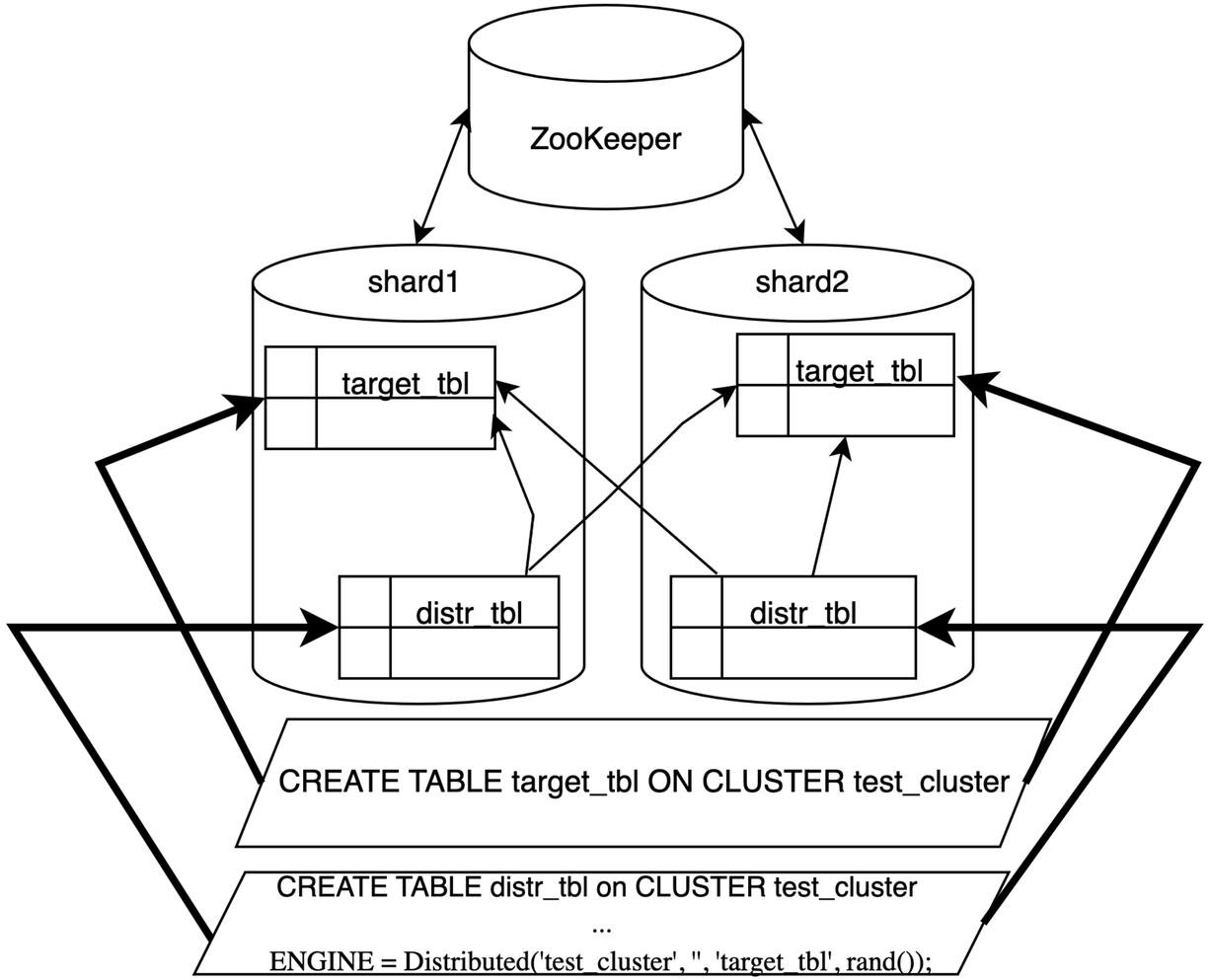
```
CREATE TABLE hits_100m
(
    `ClientIP`           UInt32,
    `UserID`             UInt64,
    `WindowClientWidth`  UInt16,
    `WindowClientHeight` UInt16,
    …
)
ENGINE = MergeTree
ORDER BY (UserID, ClientIP);
```

```
CREATE TABLE hits_100m_local
(
    `ClientIP`           UInt32,
    `UserID`             UInt64,
    `WindowClientWidth`  UInt16,
    `WindowClientHeight` UInt16,
    …
)
ENGINE = MergeTree
ORDER BY (UserID, ClientIP);
```

Distributed

```
CREATE TABLE hits_100m_distr
(
    `ClientIP`           UInt32,
    `UserID`             UInt64,
    `WindowClientWidth`  UInt16,
    `WindowClientHeight` UInt16,
    ...
)
ENGINE = Distributed('cluster', 'database', 'hits_100m_local',
murmurHash2_64(ClientIP));
```

Distributed



Ключ шардирования

Количество рядов

50 061 036

Шард 1

49 938 964

Шард 2

Обычный запрос

```
SELECT ParamPrice, UserID
FROM datasets.hits_100m
WHERE ClientIP BETWEEN 2425757232 AND 3393676766
ORDER BY RegionID
LIMIT 10
SETTINGS max_threads = 8;
```

Средний результат

```
10 rows in set.
Elapsed: 0.269 sec.
```

Распределённый запрос


```
SELECT ParamPrice, UserID
FROM datasets.hits_100m_distr
WHERE ClientIP BETWEEN 2425757232 AND 3393676766
ORDER BY RegionID
LIMIT 10
SETTINGS max_threads = 16;
```

Средний результат

```
10 rows in set.
Elapsed: 0.168 sec.
```

Преимущества

- Преодоление технических ограничений
- Повышение отказоустойчивости
- Ускорение запросов

Недостатки

- Требуют дополнительных вычислительных мощностей
- Нуждаются в соблюдении дополнительных условий (ключ шардирования, global join vs join, и т. п.)
- Необходима ручная ребалансировка

Первичный ключ

- Самый эффективный вариант
- Можно только один

Индексы пропуска данных

- Нет ограничений по количеству
- Могут нагрузить систему

Проекции

- Вариативны
- Занимают много места

Шардирование

- Позволяет распределить нагрузку
- Требует дополнительных вычислительных мощностей

Буду рад продолжить общение

Кузьма Лешаков
Yandex Cloud
TG: @notes_techkuz