НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Факультет компьютерных наук
ООО "Яндекс"
Программный проект
“Дополнительные индексные структуры для пропуска блоков данных в таблицах”

Выполнил студент группы БПМИ-171
Васильев Н. С.
Научный руководитель:
Руководитель
Миловидов А. Н.

ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

- Задача относится к методам индексации и поиска в базах данных.
- Требовалось реализовать data skipping index --- способ поиска, при котором на каждый блок данных (подряд идущие строки) сохраняется некоторая “выжимка” данных, которая позволяет при SELECT запросе определять выполнимость WHERE выражения на целом наборе строк и, если выражение на блоке выполнено быть не может, то пропустить этот блок и не проверять выполнимость на каждой строке.

ОСНОВНЫЕ ПОНЯТИЯ, ОПРЕДЕЛЕНИЯ, ТЕРМИНЫ

Clickhouse --- столбцовая система управления базами данных (СУБД) для онлайн обработки аналитических запросов (OLAP). (из https://clickhouse.yandex/docs/ru/)
***MergeTree** --- вид таблиц в СУБД Clickhouse.
Кусок (Part) --- часть данных в таблице, отсортированная по ключу.
Гранула --- множество записей в таблице, идущих подряд. В таблицах *MergeTree размер гранулы задается при создании таблицы.

Терм --- переменная или функция от термов. (a, f(a,b), ...)
Предикат --- отображение из множества наборов аргументов в {0, 1}. (=, <, IN, LIKE)
Атомарная формула --- предикат от термов. (a + 10 > b)
Формула --- атомарная формула, или логическое выражение от формул.
Ограничение на аргументы --- подмножество множества всех наборов аргументов.
Выполнимая формула --- формула, которая может быть выполнена при данных ограничениях на аргументы.

АКТУАЛЬНОСТЬ РАБОТЫ

- Большой объем данных не позволяет эффективно использовать обычные индексы.
- Из-за этого СУБД Clickhouse в таблицах *MergeTree сортирует данные по первичному ключу и использует разреженный индекс.
- Но, несмотря на то, что выбор правильного первичного ключа позволяет СУБД Clickhouse выполнять запросы достаточно быстро, объем читаемых данных можно уменьшить еще сильнее.
- Также в некоторых запросах может вообще не быть условия на столбцы из первичного ключа, например,
  SELECT uniq(URL) FROM hits_v1 WHERE SearchPhrase LIKE '%яндекс%'
  просканирует всю таблицу.
- Производительность поиска можно попытаться улучшить с помощью индексных структур для пропуска блоков данных.

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Цель работы

Добавить возможность создания легковесных индексных структур, которые при записи будут сохранять некоторую выжимку о данных, а потом использовать её для уменьшения объема читаемых с диска данных при SELECT запросах, тем самым увеличивая скорость выполнения запроса.

Формально: определять выполнимость формулы на блоке данных, используя некоторую заранее рассчитанную и сохраненную информацию о данных в блоке (ограничения на аргументы), а также известные заранее характеристики данных, например, типы.

Задачи работы

1. Разработка общего механизма создания, изменения, расчета и хранения индексных структур.
2. Добавление конструкций в язык запросов (специальный SQL) для операций с индексными структурами.
3. Реализация различных индексных структур. (min/max, bloom filter, set)
4. Проверка улучшения производительности.

АНАЛИЗ СУЩЕСТВУЮЩИХ РЕШЕНИЙ

- Аналитическая платформа DataBricks Delta разрабатываемая компанией Databricks Inc. использует min-max индекс для оптимизации запросов. (запись в блоге от 31 июля, 2018)
  https://databricks.com/blog/2018/07/31/processing-petabytes-of-data-in-seconds-with-databricks-delta.html

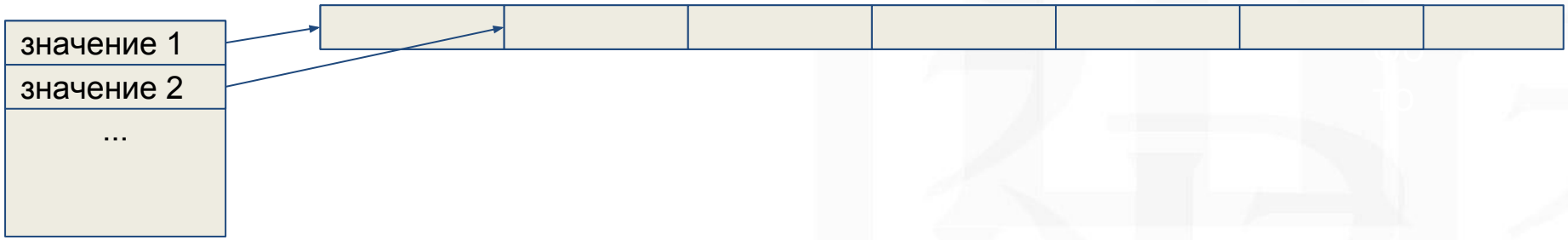
- IBM Cloud SQL Query разрабатываемая International Business Machines Corp. (запись в блоге от 4 марта, 2019).
  Использует индексы Min/max, Value list, Geospatial.
  https://www.ibm.com/blogs/bluemix/2019/03/data-skipping-for-ibm-cloud-sql-query/

- Статья “Column Imprints: A Secondary Index Structure”, авторы: Lefteris Sidirourgos и Martin Kersten, 22 - 27 июня, 2013.
  В статье рассматривается использование Column Imprints для оптимизации запроса.
  https://homepages.cwi.nl/~manegold/COMMIT/imprints.pdf

- “Processing a Trillion Cells per Mouse Click”, авторы: Alex Hall, Olaf Bachmann, Robert Buessow, Silviu-Ionut Ganceanu, Marc Nunkesser, 2012.
  Сохранение словаря уникальных значений + множества уникальных значений в чанке.
  https://ai.google/research/pubs/pub40465

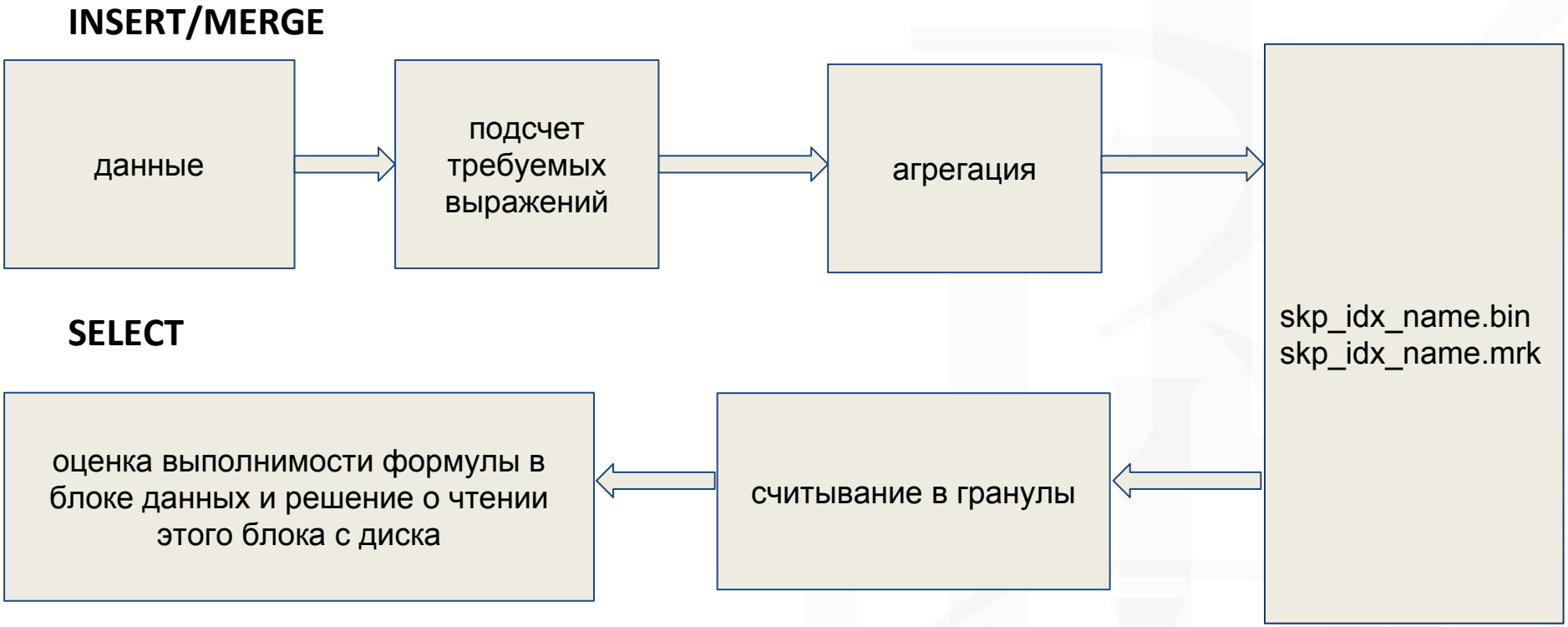
Как устроены таблицы *MergeTree

- Таблица разбита на отсортированные куски.
- В фоновом режиме куски сливаются в большие куски.
- Каждый кусок хранится в своей директории.
- Каждый кусок разбит на гранулы.
- Первичный ключ --- список значений ключа на краях гранул (первая строка каждой гранулы).
- На каждый столбец хранится два файла:
  - Бинарный файл со сжатыми данными
  - Файл с засечками, т. е. парами {индекс в разжатом файле, индекс в сжатом файле}.

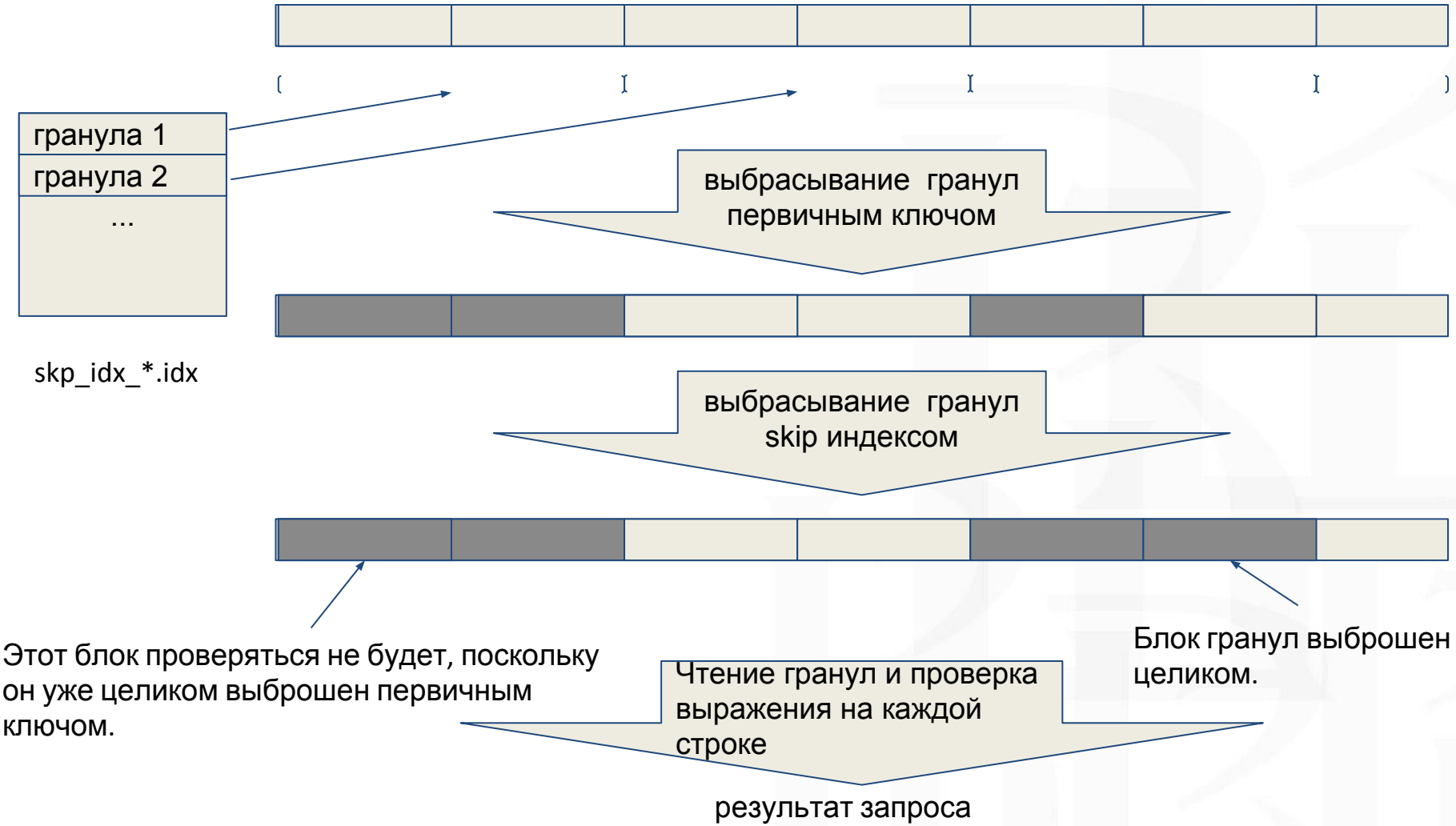


первичный ключ

Общая схема работы индексных структур



Общая схема работы индексных структур



Выбор алгоритмов для индексных структур

Краткий обзор реализованных индексов

min/max индекс
Хранит минимум и максимум в блоке по каждому из заданных термов. Тем самым для каждого блока хранится параллелограмм, выполнимость запроса в котором надо проверять.

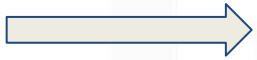
set индекс
Для заданного кортежа термов сохраняет все его уникальные значения в блоке. Потом использует их, подставляя в формулу, выполнимость которой нужно вычислить, сохраненные значения.

bloom filter индексы
Работает только со строками. Для заданного кортежа из имеющих строковой тип термов сохраняет все токены в блоке определенного типом индекса вида в фильтр Блума. Потом использует его проверяя вложенность одного множества в другое.

Min/max индекс

- Просто сохраняет минимум и максимум по каждому указанному терму для которого доступна операция сравнения, тем самым получая параллелограмм, содержащий все данные блока.
- При выполнении SELECT проверяется выполнимость формулы на параллелограмме с помощью уже существующих в кодовой базе методов, которые используются для первичного ключа (KeyCondition).
  - KeyCondition строит по формуле обратную польскую нотацию, заменяя атомарные формулы, для которых невозможно определить выполнимость на специальное значение, обозначающее неизвестность результата.
  - Затем выполняет полученную обратную польскую нотацию на параллелограмме, храня в процессе флаг возможности выполнимости выражения и флаг возможности невыполнимости выражения.
  - KeyCondition также использует оптимизации, например, для частично монотонных функций.

1	5
10	-100
4	3



point 1	1	-100
point 2	10	5

Set индекс

- Сохраняет множество уникальных значений заданного кортежа термов в блоке. Для агрегации используется хеш-таблица.
- При выполнении SELECT:
  - По полученной формуле строится атомарная формула, которая зависит только от термов из индекса.
    - Логические операции заменяются на битовые (а предикаты становятся термами, которые возвращают UInt8), которые интерпретируют последний бит как флаг возможности выполнимости выражения, а предпоследний как флаг возможности невыполнимости выражения.
    - Бывшие атомарные формулы, которые зависят от термов, которые невозможно вычислить, используя только термы из индекса, заменяются на терм-константу 11_2, которая значит, что о терме ничего не известно.
    - Полученный терм оборачивается в предикат равный последнему биту.
    - Все это выполняется в dfs по абстрактному синтаксическому дереву запроса.
  - Затем начинают поступать гранулы индекса, которые содержат сохраненные кортежи. На каждом таком кортеже происходит вычисление выражения, и, если на всех кортежах выражение ложно, то формула невыполнима в соответствующем грануле блоке данных.

Set индекс

a	b
1	1
20	1
10	4
1	1
10	5
20	1
10	4
20	1

1	1
20	1
10	4
10	5

WHERE a < b AND (a = 5 OR c = d)

bitAnd(less(a,b), bitOr(equal(a, 5), 3))

Bloom filter индекс

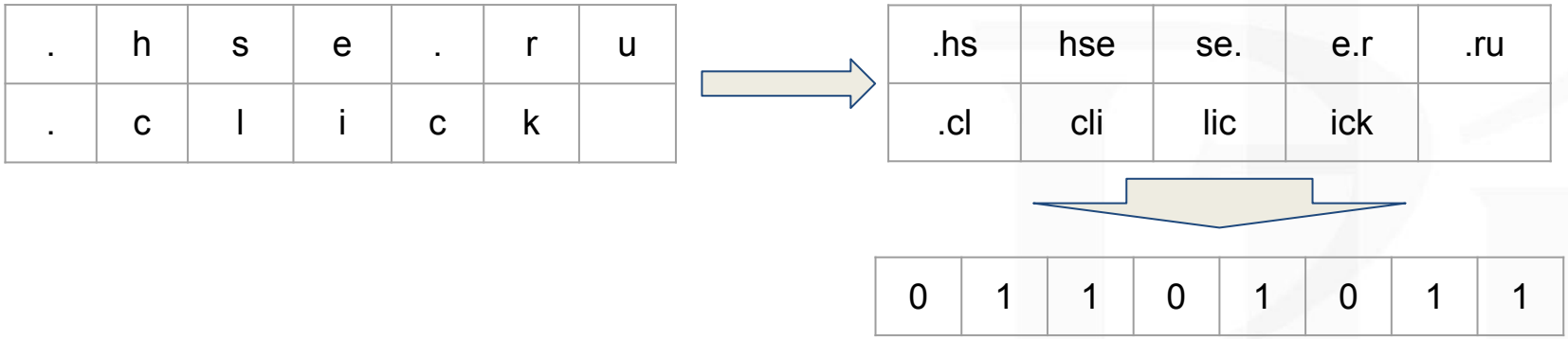
- Работает только для строк.
- Имеется функция-токенизатор, которая для данной строки возвращает множество токенов. В нашем случае это функция, возвращающая н-грамы или токены (здесь токен --- последовательность цифр, латинских букв и символов не из ascii). (Все в utf-8.)
- Хранит фильтр Блума, содержащий все токены, которые встретились в строках в блоке.
- При агрегации добавляет все полученные токенизатором токены в фильтр Блума.

Bloom filter индекс: SELECT запрос

- Строится обратная польская нотация для формулы подобно KeyCondition, но теперь могут оцениваться только атомные формулы, которые содержат предикаты equal (=), LIKE или IN, левый (а для equal любой) аргумент которых --- терм в индексе, а второй --- константа.
- Каждой атомарной формуле одного из оцениваемых типов в обратной польской нотации сопоставляется фильтр Блума.
- Для оцениваемой атомарной формулы для аргумента-константы собираются все содержащиеся в нем токены и вставляются в сопоставленный фильтр Блума. (Для LIKE используется отдельный токенизатор, который правильно обрабатывает экранирование и специальные символы, для IN хранится несколько фильтров Блума.)
- Затем выполняется выражение, записанное в обратной польской нотации. Для оценки выполнимости атомарной формулы проверяется включение фильтра Блума для константы в фильтр Блума гранулы.

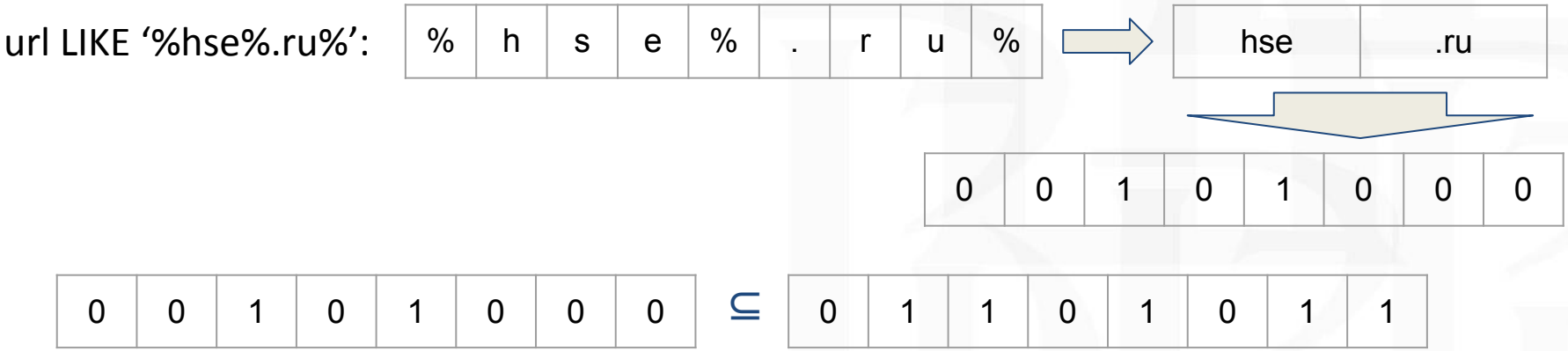
Bloom filter индекс

Агрегация

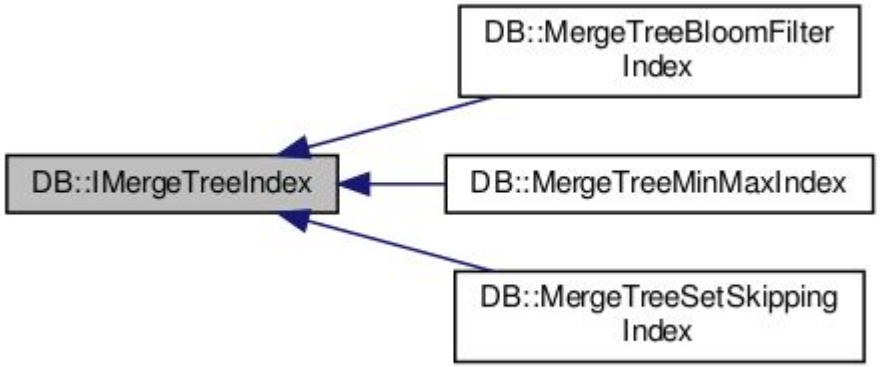


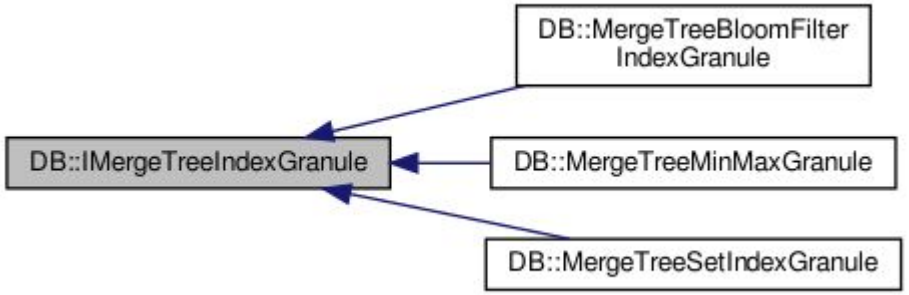
Проверка

url LIKE ‘%hse%.ru%’:



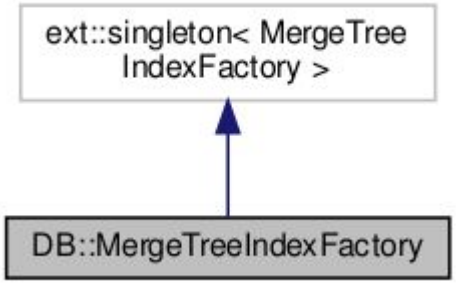
Интерфейсы вторичных индексов





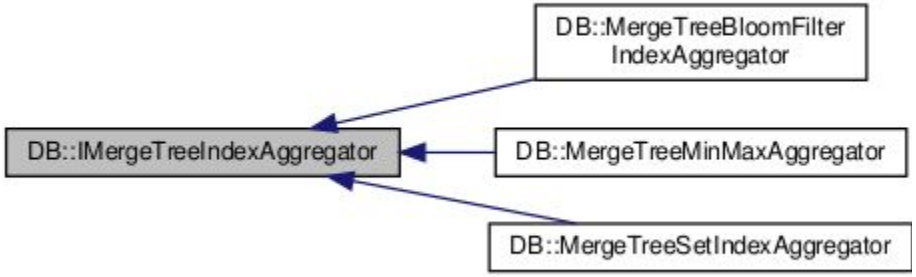
- Определяет тип индекса.
- Хранит метаинформацию.
- Отвечает за создание гранул, агрегаторов и условий.

- Хранение информации о блоке данных.
- Бинарная сериализация/десериализация.

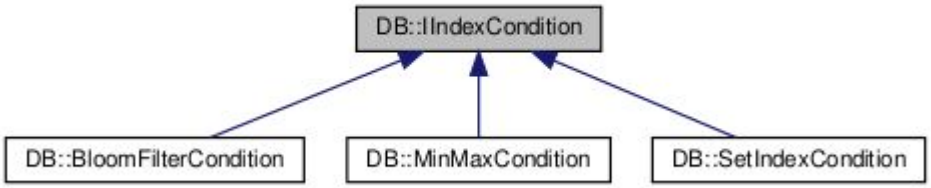


- Создание индексов (объектов IMergeTreeIndex) из описания (абстрактного синтаксического дерева).

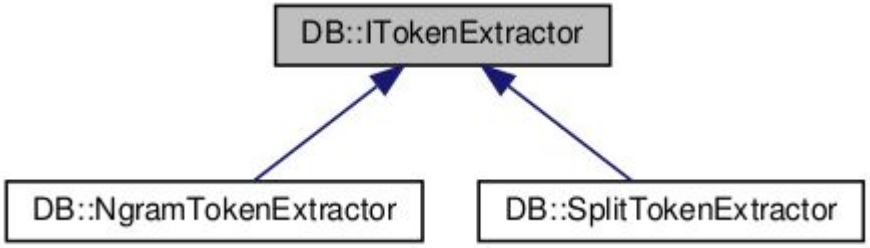
Интерфейсы вторичных индексов



- Агрегирует данные из блока данных в гранулу.
- Управляется извне.

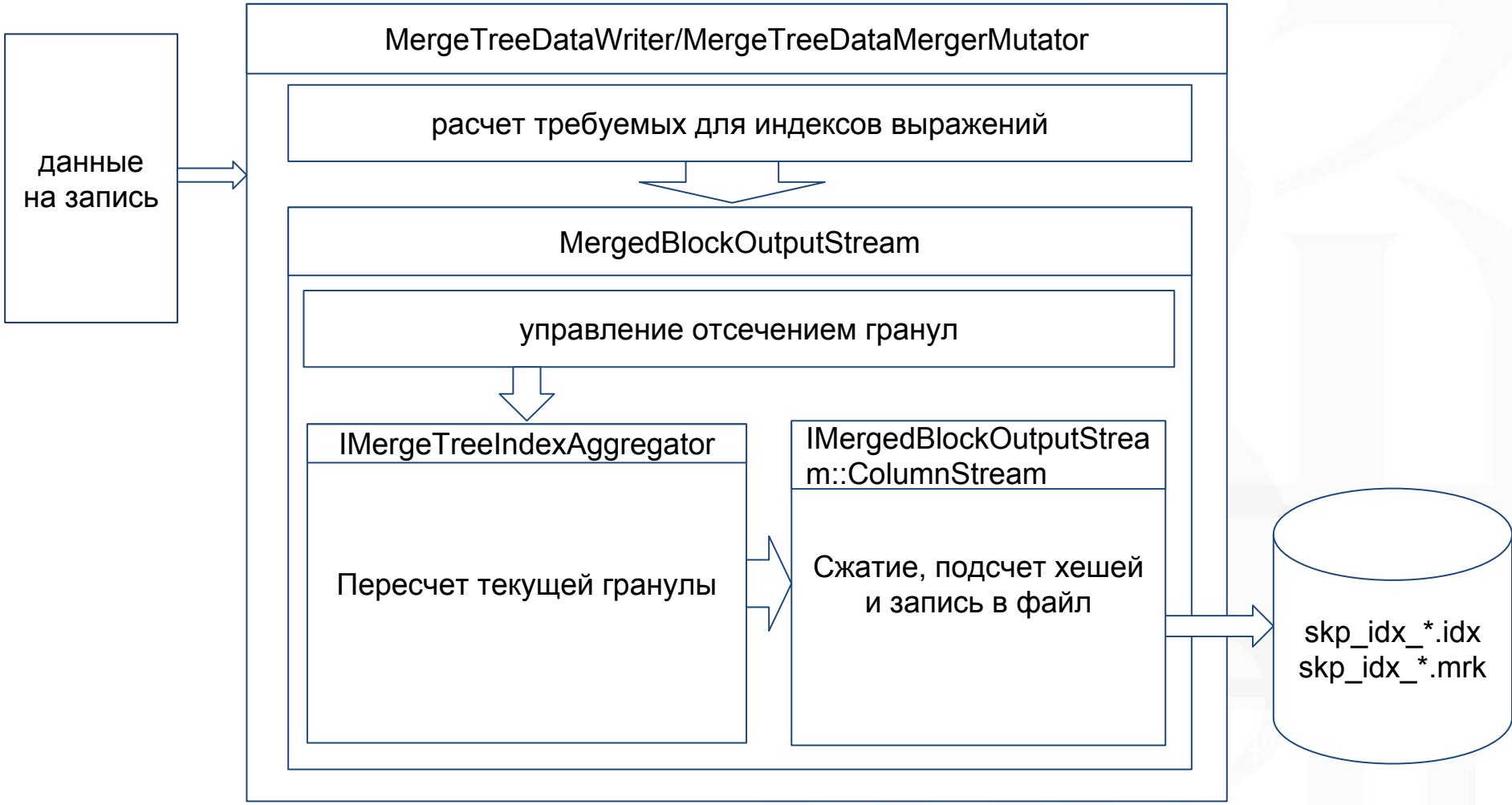


- Строится из WHERE выражения.
- Определяет полезность индекса на данном запросе.
- Определяет выполнимость запроса на блоке данных по грануле индекса.

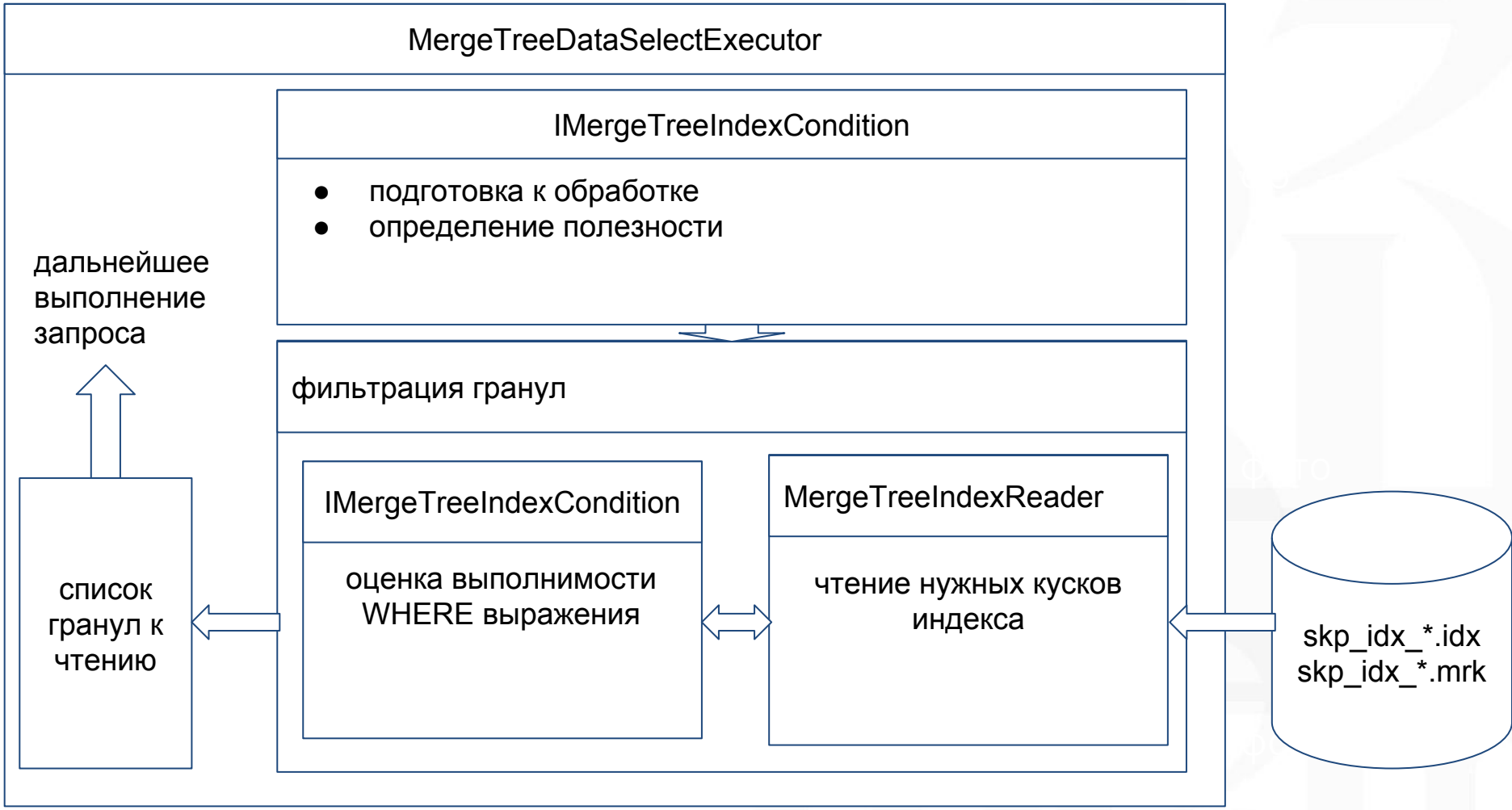


- Специфичен для bloom filter индекса.
- Выделяет токены из строки.
  - Метод для быстрого inplace выделения для обычного использования.
  - Метод для токенизации LIKE запроса, который обрабатывает экранирование и спец. символы.

Архитектура вторичных индексов: INSERT и слияния



Архитектура вторичных индексов: SELECT



Архитектура вторичных индексов: хранение

Хранение

- Хранится в метаданных таблицы. (SQL файл с запросом ATTACH TABLE)
- Хранятся сами индексы в формате: сжатый бинарный файл + файл с засечками, как и столбцы.

Добавление/удаление индексов

- Запрос ALTER TABLE .. ADD INDEX … просто добавляет сведения об индексе в метаданные, индексы считаются только при последующих вставках и слияниях, т.е. создание ленивое.
- Запрос ALTER TABLE … DROP INDEX … удаляет индекс из метаданных и стирает его файлы с диск.

Репликация

- Для реплицируемых таблиц информация об индексах хранится в ZooKeeper вместе с остальной метаинформацией.
- Индексы реплицируются вместе со столбцами.

Тестирование

- Проверялась работоспособность с помощью функциональных тестов СУБД Clickhouse.

- Также осуществлялось ручное тестирование с просмотром логов для проверки правильности функционирования частей программы, которые напрямую не влияют на правильность результата, но могут приводить к неприятным эффектам, например, к снижению производительности.

- Тестирование скорости работы проводилось с помощью запуска много раз запроса с получением замера времени из Clickhouse. Запросы выполнялись на небольшом куске данных (~10 гб) сервиса “Яндекс.Метрика”, который можно скачать с официального сайта СУБД Clickhouse.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ РАБОТЫ

Создание таблицы с индексами

```
ThinkPad-E570 :) CREATE TABLE table_name (u64 UInt64, i32 Int32, s String, INDEX a (u64 * i32, s) TYPE minmax GRANULARITY 3, IN
DEX b (u64 * length(s)) TYPE set(1000) GRANULARITY 4 ) ENGINE = MergeTree() ORDER BY tuple()

CREATE TABLE table_name
(
    u64 UInt64,
    i32 Int32,
    s String,
    INDEX a (u64 * i32, s) TYPE minmax GRANULARITY 3,
    INDEX b u64 * length(s) TYPE set(1000) GRANULARITY 4
)
ENGINE = MergeTree()
ORDER BY tuple()

Ok.

0 rows in set. Elapsed: 0.060 sec.
```

Изменение индексов

```
ThinkPad-E570 :) alter table table_name drop index b, add index b (s) TYPE ngrambf_v1(5, 16, 2, 0) GRANULARITY 15

ALTER TABLE table_name
    DROP INDEX b,
    ADD INDEX
    b s TYPE ngrambf_v1(5, 16, 2, 0) GRANULARITY 15


Ok.

0 rows in set. Elapsed: 0.041 sec.
```

ОСНОВНЫЕ РЕЗУЛЬТАТЫ РАБОТЫ

Выбрасывание блоков данных

INDEX tk (URLDomain, Title, SearchPhrase) TYPE tokenbf_v1(512, 1, 0) GRANULARITY 1

```
ThinkPad-E570 :) select uniq(URL) from hits where SearchPhrase like '% яндекс %'

SELECT uniq(URL)
FROM hits
WHERE SearchPhrase LIKE '% яндекс %'

┌─uniq(URL)─┐
│        68 │
└───────────┘

1 rows in set. Elapsed: 0.066 sec. Processed 4.79 million rows, 70.44 MB (73.03 million rows/s., 1.07 GB/s.)
```

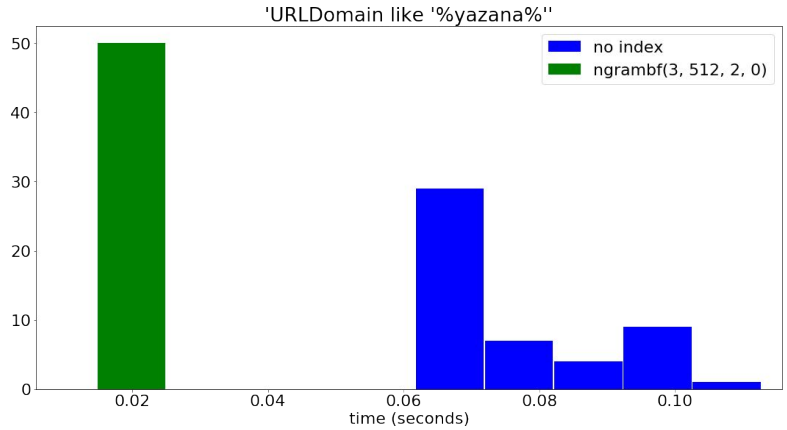
нет индексов

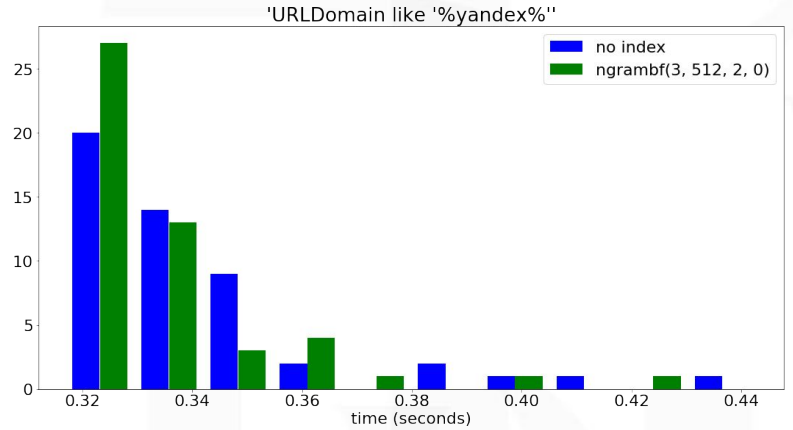
```
ThinkPad-E570 :) select uniq(URL) from datasets.hits_v1 where SearchPhrase like '% яндекс %'

SELECT uniq(URL)
FROM datasets.hits_v1
WHERE SearchPhrase LIKE '% яндекс %'

┌─uniq(URL)─┐
│        68 │
└───────────┘

1 rows in set. Elapsed: 0.083 sec. Processed 8.87 million rows, 112.70 MB (106.50 million rows/s., 1.35 GB/s.)
```

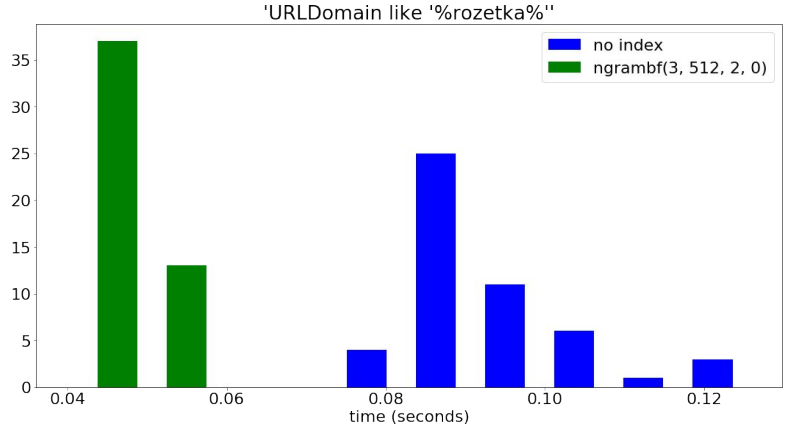
Измерения скорости запросов

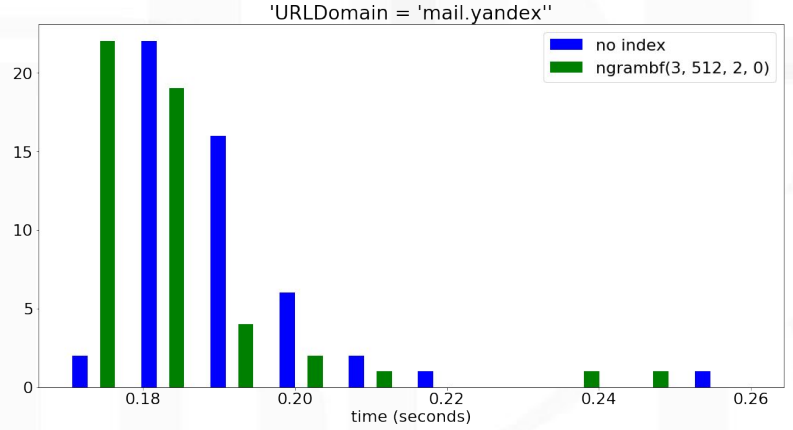
ngrambf_v1(3, 512, 2, 0)

select uniq(URL) from {table} where {where}





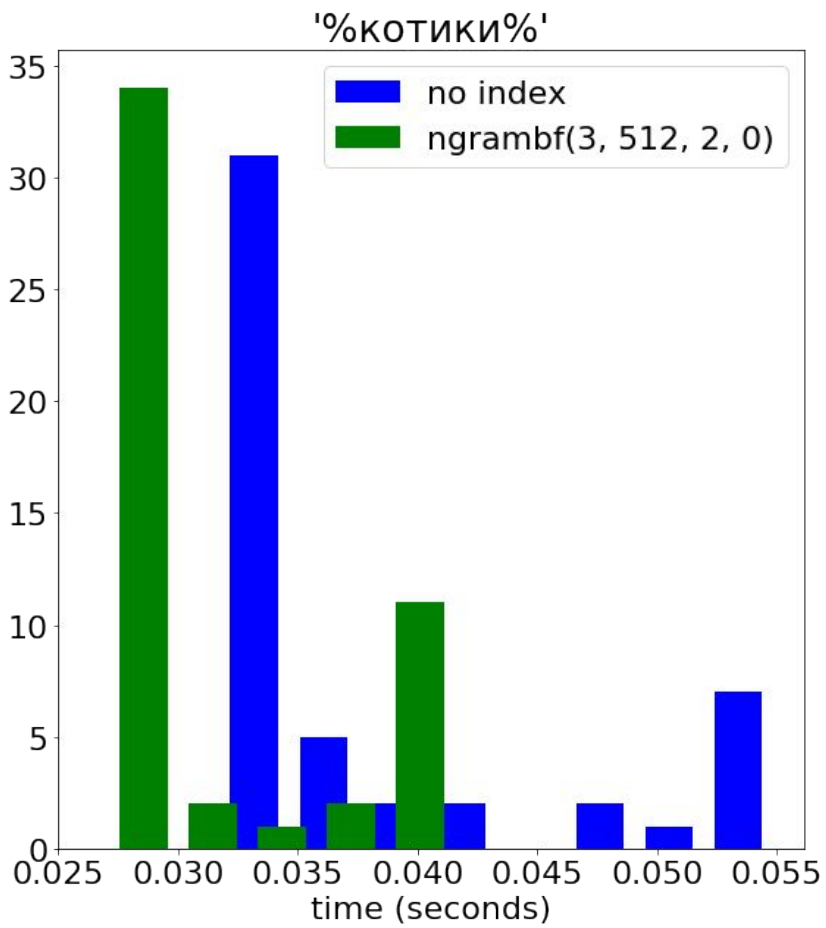


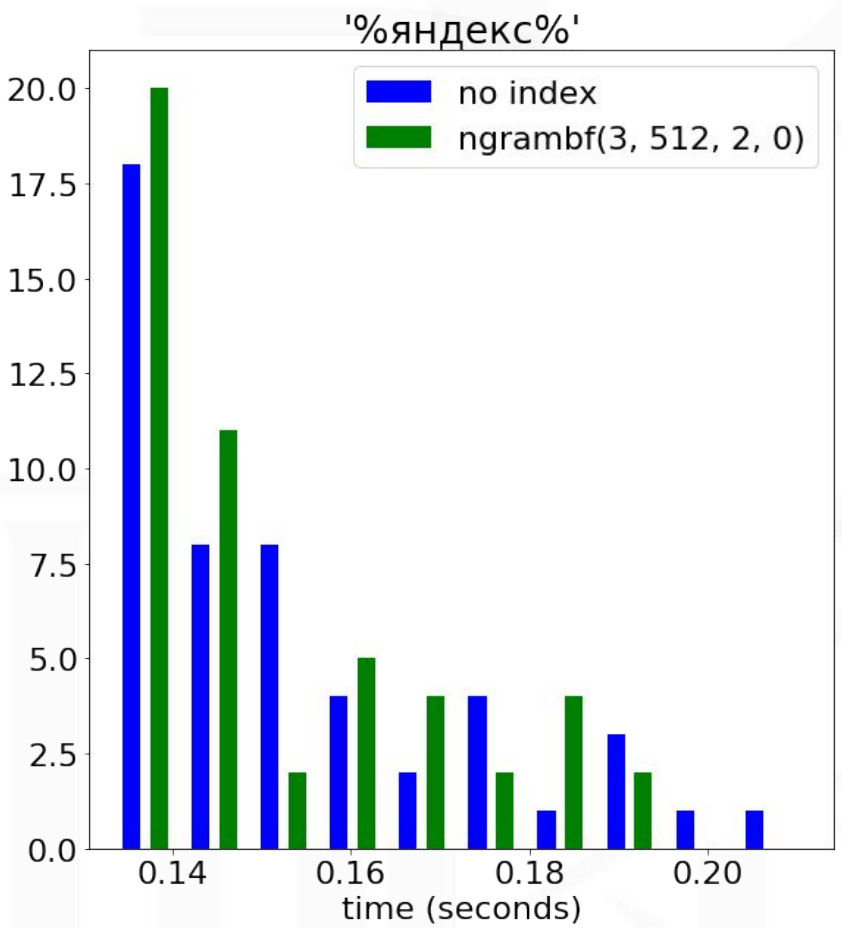


Измерения скорости запросов

ngrambf_v1(3, 512, 2, 0)

select uniq(URL) from {table} where SearchPhrase like '{phrase}'

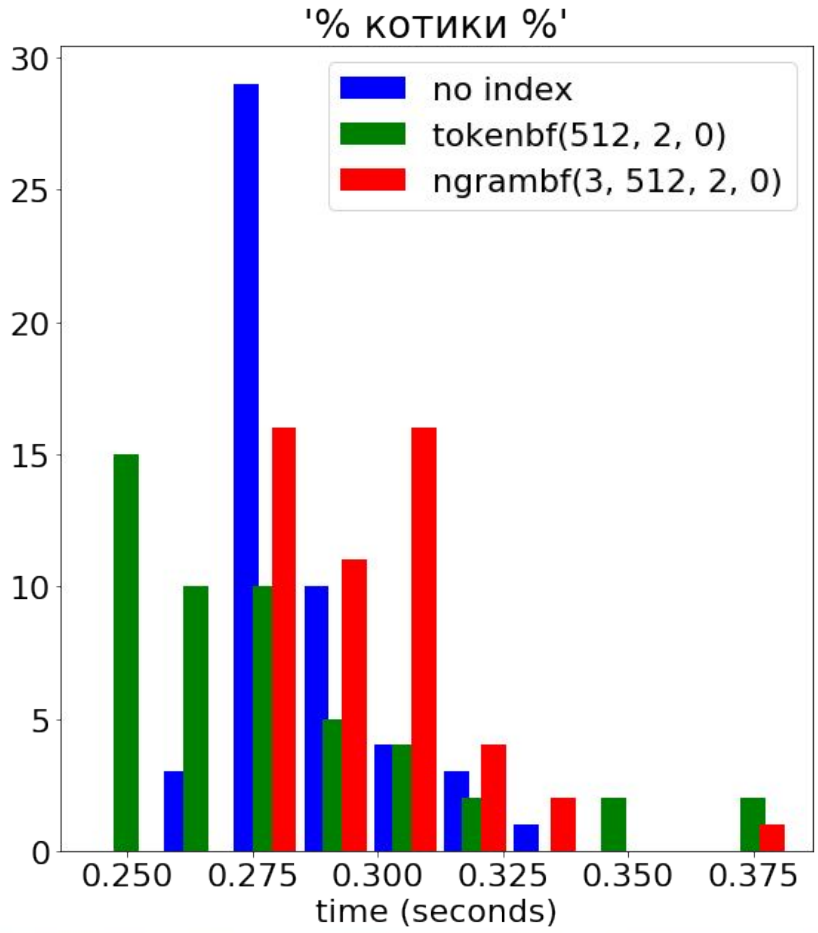


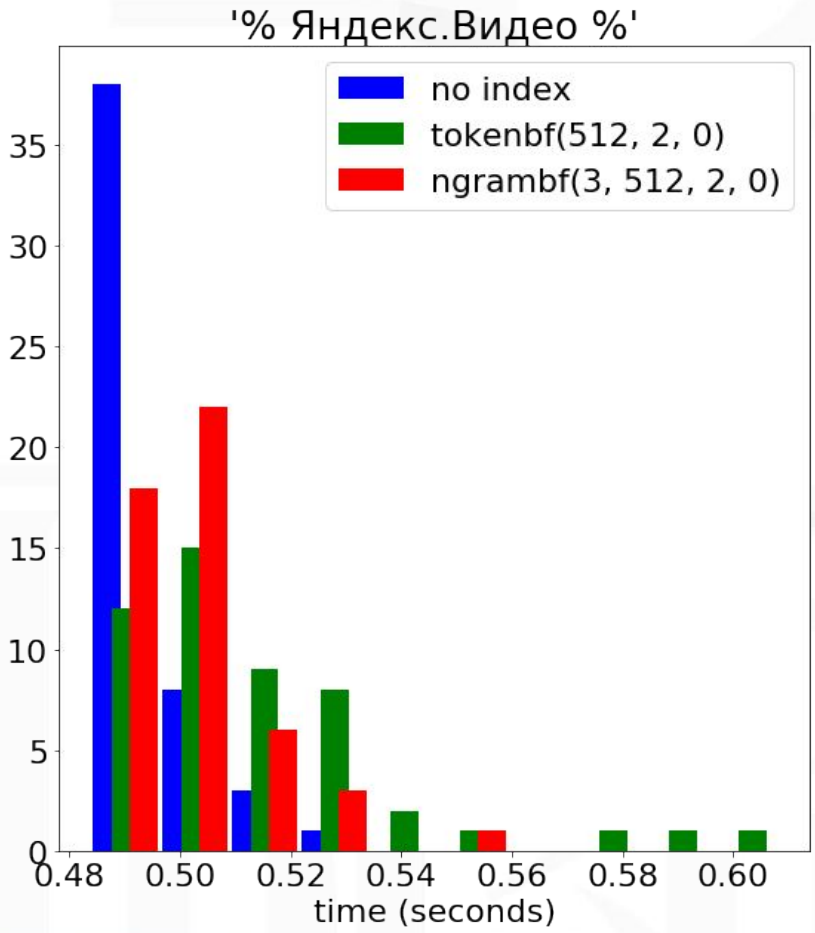


Измерения скорости запросов

tokenbf_v1(512, 2, 0) и ngram_v1(3, 512, 2, 0)

select uniq(URL) from {table} where Title like '{text}'



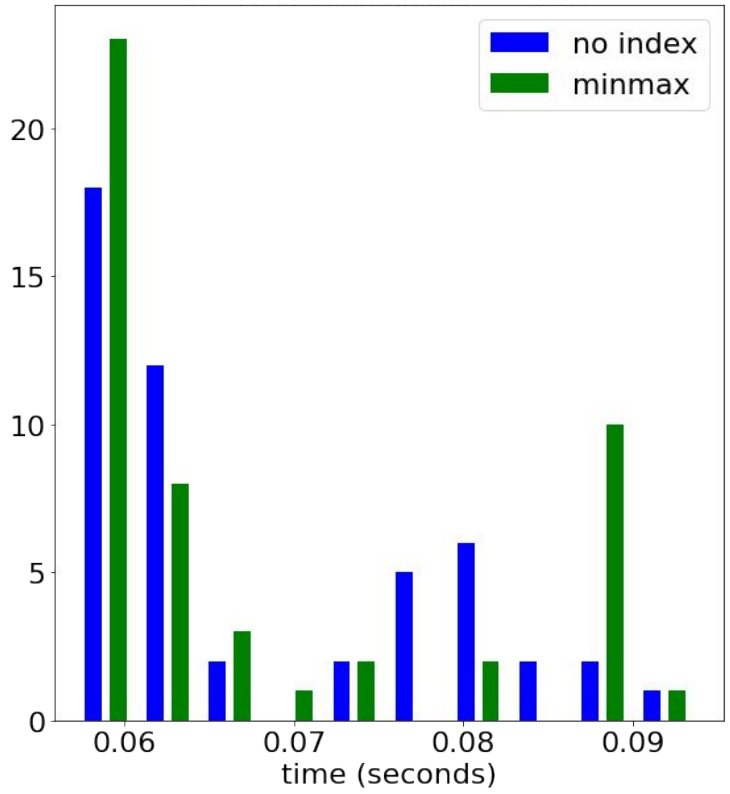


Измерения скорости запросов

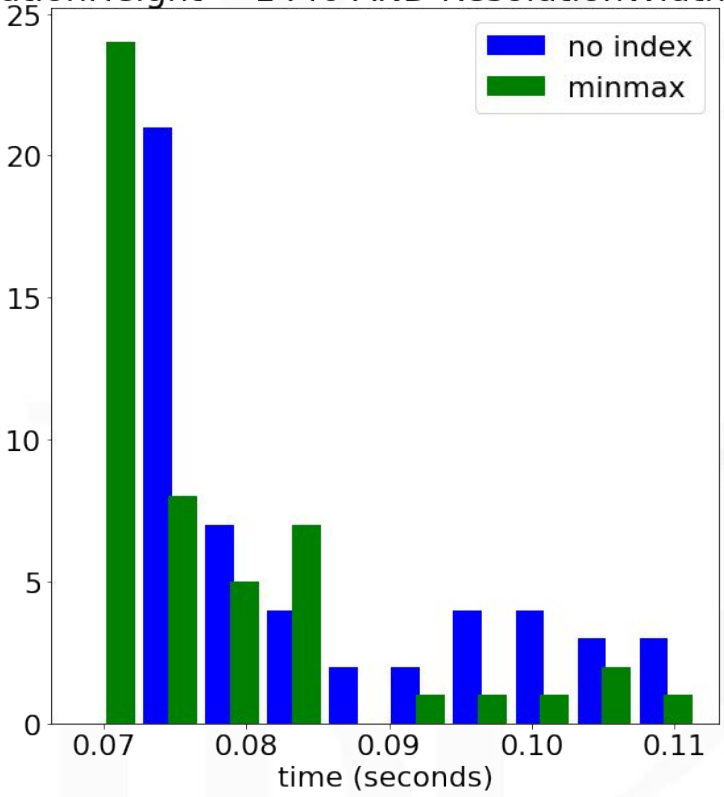
minmax GRANULARITY 2

select uniq(URLDomain) from {table} where {where}

'ResolutionWidth < 640'

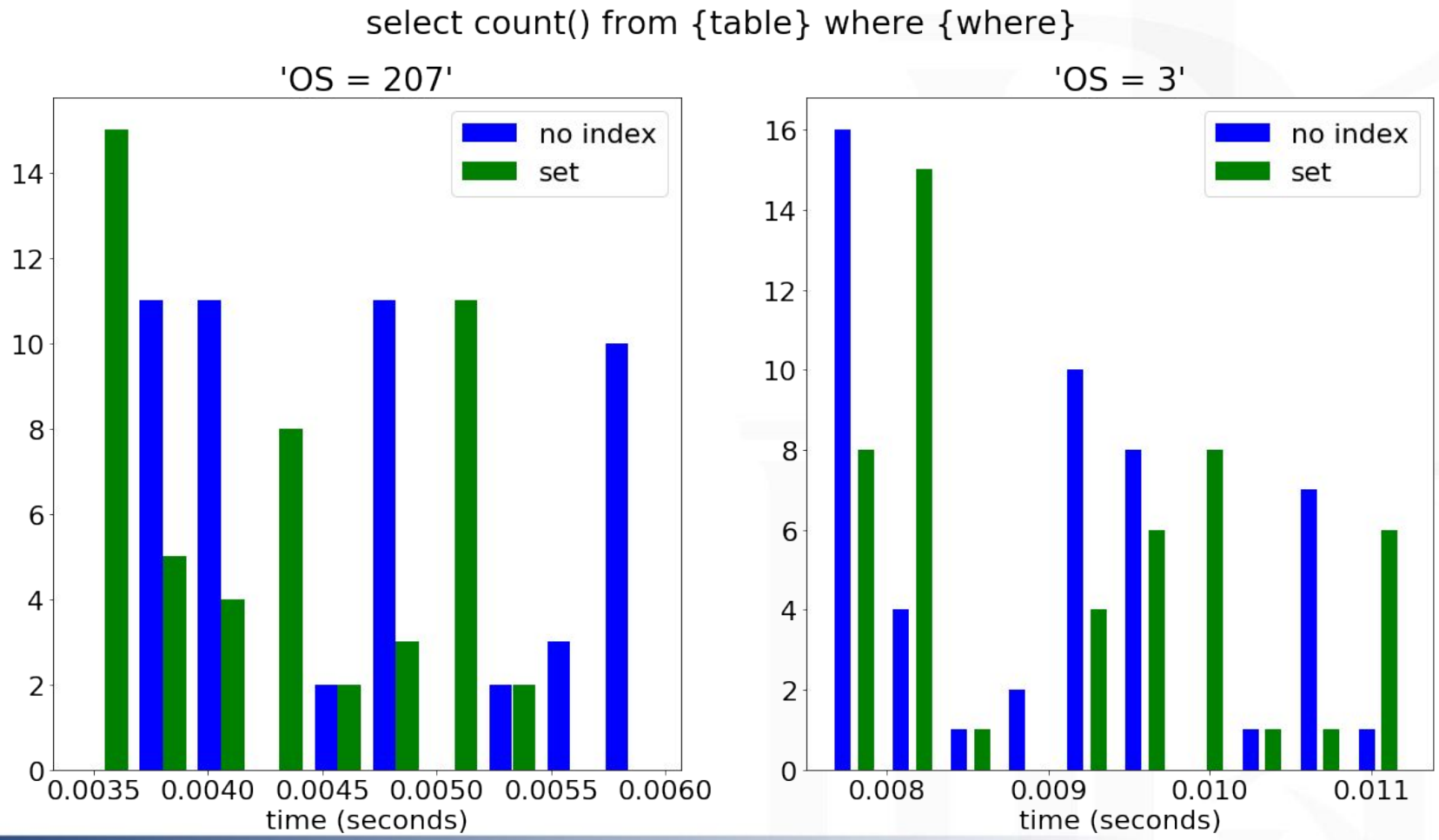


'ResolutionHeight > 1440 AND ResolutionWidth > 2560'



Измерения скорости запросов

set



ВЫВОДЫ ПО РАБОТЕ

- Написан код, позволяющий легко добавлять новые вторичные индексы в СУБД Clickhouse.
- Min/max, set и bloom filter индексы добавлены в код СУБД Clickhouse в экспериментальном режиме (требуется включение специальной настройки).
- Показано увеличение скорости работы некоторых типов запросов при правильном подборе skip индексов.

Направления дальнейшей работы

- Задание способов сжатия индексов (в коде уже присутствует для столбцов).
- Кэширование индексов, а также полностью сохраненные в оперативной памяти индексы.
- Новые типы индексов
  - Imprints [6]
  - С внешним словарем (комбинация set индекса с LowCardinality столбцами) [4]
  - Новые токенизаторы для индексов на фильтрах Блума, например, выделяющие токены определенного вида по регулярному выражению.
  - Индексы имеющие интерпретацию. [5]
  - Специальные индексы для поиска аномалий в данных.
- Оптимизация функций хеширования для фильтров Блума.
- Комбинирование индексов.
- Индексы по целым кускам. (Частично уже есть в Clickhouse)
- Выполнять выбрасывание блоков данных параллельно на разных кусках.
- Мультигранулярность. Задается функция, по размеру куска выдающая массив значений гранулярностей для индекса, потом выбрасывание проводится от индекса с большей гранулярностью до индекса с меньшей гранулярностью.
- Кэширование результатов вычисления set индекса. (например, LRU).
- Использование для более сильной оптимизации простых запросов, например, min/max и set можно было бы использовать для оптимизации запросов вида SELECT min(column) FROM …

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] ClickHouse Documentation [Электронный ресурс] / Yandex. — Режим доступа: https://clickhouse.yandex/docs/en/, свободный. (дата обращения: 21.03.18).

[2] Eaton C. How Data Skipping works in BLU Acceleration [Электронный ресурс] / IBM. — Режим доступа: https://it.toolbox.com/blogs/ceaton55/how-data-skipping-works-in-blu-acceleration-part-1-031814, свободный. (дата обращения: 21.03.19).

[3] Hall, A. Processing a Trillion Cells per Mouse Click / Alexander Hall, Olaf Bachmann, Robert B ̈ussow, Silviu G ̆anceanu, Marc Nunkesser // [Электронный ресурс] / Google, Inc. — Режим доступа: http://vldb.org/pvldb/vol5/p1436_alexanderhall_vldb2012.pdf, свободный. (дата обращения: 21.03.18).

[4] Ionescu A. Processing Petabytes of Data in Seconds with Databricks Delta [Электронный ресурс] / Databricks. — Режим доступа: https://databricks.com/blog/2018/07/31/processing-petabytes-of-data-in-seconds-with-databricks-delta.html, свободный. (дата обращения: 21.03.19).

[5] Ta-Shma P. Data Skipping for IBM Cloud SQL Query [Электронный ресурс] / IBM. — Режим доступа: https://www.ibm.com/blogs/bluemix/2019/03/data-skipping-for-ibm-cloud-sql-query/, свободный. (дата обращения: 21.03.19).

[6] Sidirourgos, L. Column Imprints: A Secondary Index Structure / Lefteris Sidirourgos, Martin Kersten // [Электронный ресурс] / Centrum Wiskunde & Informatica — Режим доступа: https://homepages.cwi.nl/~manegold/COMMIT/imprints.pdf, свободный. (дата обращения: 21.03.18).

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Спасибо за внимание!

Васильев Никита Сергеевич
vasnikserg@yandex.ru

Москва - 2019