
Яндекс

Яндекс

Оптимизация строк в ClickHouse

Николай Кочетов
Разработчик ClickHouse

Хранение строк в ClickHouse

Строковые типы данных

String

- Вариант по умолчанию
- Overhead — 9 байт на строку (в оперативке)
- Использовать пока не тормозит

FixedString

- Точно знаем размер в байтах (IP, MD5)
- Произвольные бинарные данные
- Не использовать для ограниченных по размеру строк

Строковые типы данных

Запросы из таблиц с одинаковами данными

```
SELECT sum(ignore(val)) FROM table_1

Processed 1.00 billion rows, 4.00 GB (1.86 billion rows/s., 7.46 GB/s.)

SELECT sum(ignore(val)) FROM table_2

Processed 1.00 billion rows, 17.89 GB (683.57 million rows/s., 12.23 GB/s.)
```

Строковые типы данных

Размер сжатых данных

```
SELECT
    table,
    formatReadableSize(sum(data_compressed_bytes)) AS compressed_size
FROM system.parts
WHERE active AND (table LIKE 'table_%')
GROUP BY table
```

```
┌─table───┬─compressed_size─┐
│ table_1 │ 3.90 GiB        │
│ table_2 │ 3.74 GiB        │
└─────────┴─────────────────┘
```

> Таблицы хранят первый миллиард чисел в виде UInt64 и String

Строки низкой гранулярности

Enum8, Enum16

- Множество строк заранее известное
- Множество строк (почти) никогда не изменяется

Достоинства

- Хранение и обработка в виде чисел
- Дешевый GROUP BY, IN, DISTINCT, ORDER BY
- Оптимизация частных случаев (e.g. сравнение с константной строкой)

Недостатки

- ALTER

ALTER Enum

Почему возникает проблема?

- Структура Enum записана в схеме таблицы
- Ждем завершения чтения из таблицы

Можем ли сделать лучше?

- Хранить структуру вне таблицы (ZooKeeper)
- Не дожидаться завершения запросов на чтение

Потенциальнае проблемы

- Синхронизация
- fetch куска с другой релики

Внешние словари

Храним строки — в словаре, ключи — в таблице

Преимущества

- Динамически изменяемое множество строк
- Нет проблемы альтеров
- Храним словарь где-то еще

Недостатки

- Неудобный (явный) синтаксис
- Отсутствие неявных оптимизаций
- Храним словарь где-то еще

Локальные словари

Отказываемся от общих глобальных словарей

> **Нет синхронизации — нет проблем**

Храним словарь локально

- На блок (в памяти)
- На кусок (при записи на диск)
- В кеше (при чтении)

Словарное кодирование строк

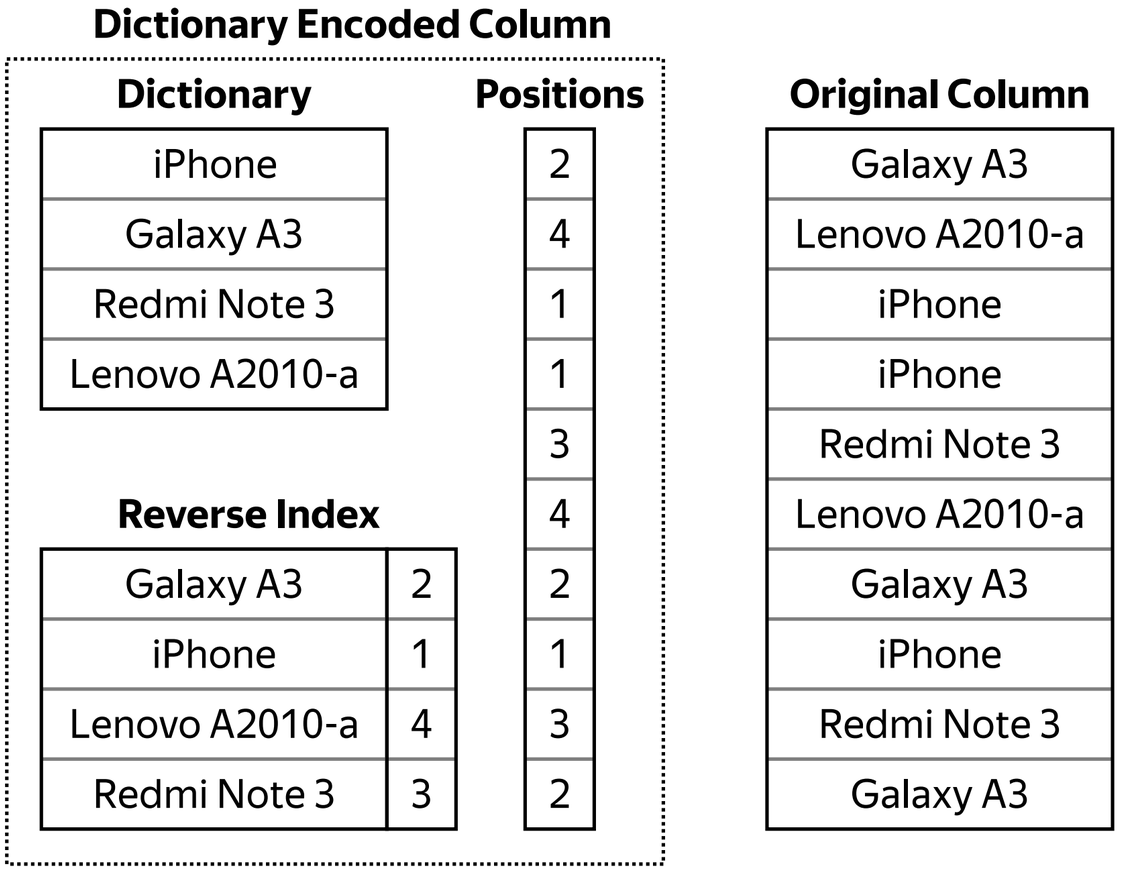
StringWithDictionary

Тип данных для строк со словарным кодированием

- Формат хранения
- Представление в памяти
- Обработка данных

Состав:

- Словарь
- Столбец позиций
- Обратный индекс



LowCardinality

LowCardinality(Type) - тип данных со словарным кодированием.

- StringWithDictionary — alias для LowCardinality(String).
- Поддержан для строк, чисел, Date, DateTime, Nullable.
- Сохраняется для некоторых функций

```
SELECT
    toLowCardinality('') AS s,
    toTypeName(s),
    toTypeName(length(s))
```

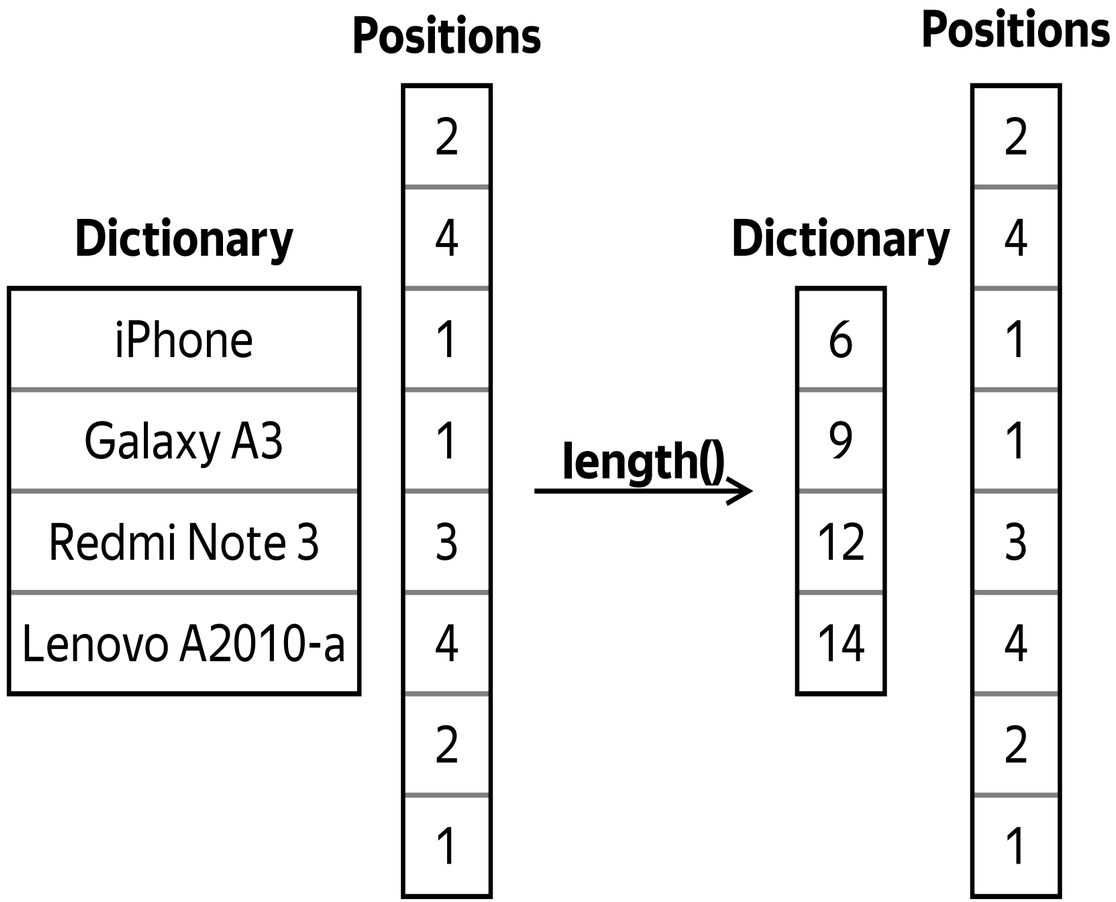
```
┌─s─┬─toTypeName(toLowCardinality(''))─┬─toTypeName(length(toLowCardinality('')))─┐
│   │ LowCardinality(String)           │ LowCardinality(UInt64)                   │
└───┴──────────────────────────────────┴──────────────────────────────────────────┘
```

Оптимизация выполнения запросов

В простых случаях выполняем функции над словарями

Запланированное

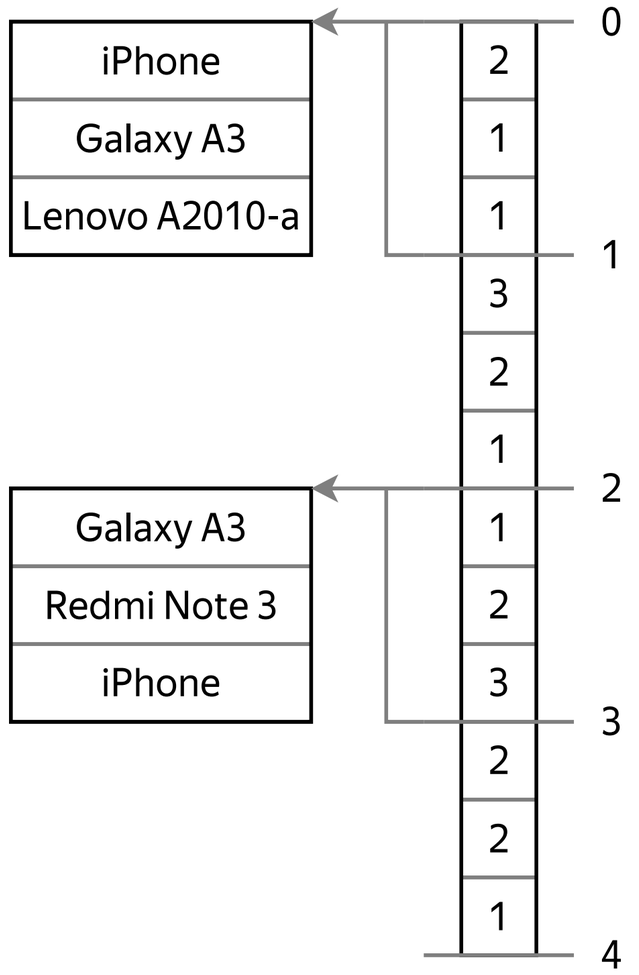
- Кеширование вычислений над словарями
- Оптимизация GROUP BY
- Специализация агрегатных функций



Строки высокой кардинальности

Что будет, если вставить много различных строк?

- Максимальный размер словаря: low_cardinality_max_dictionary_size
- Запись частей словаря локально
- Нескольких словарей на кусок: low_cardinality_use_single_dictionary_for_part
- Преобразование в обычный столбец (в планах)



Объем хранимых данных

Сколько можем сэкономить на объеме данных?

Столбец	COUNT DISTINCT	String	Dictionary	Enum
CodePage	62	72.18 MiB	26.97 MiB	26.20 MiB
PhoneModel	48044	439.20 MiB	440.61 MiB	-
URL	137103569	13.15 GiB	11.28 GiB	-

lz4, zstd используют словарное сжатие

Оценка скорости работы

TLC trip record dataset

База с описанием поездок на такси в Нью-Йорке
https://github.com/toddwschneider/nyc-taxi-data

Более 1.1 миллиарда поездок с Января 2009 по Июнь 2015

- Время начала и окончания поездки
- Имя локации начала и окончания поездки
- Способ оплаты
- Число пассажиров
- Вид такси (yellow taxi, green taxi, Uber)

TLC trip record dataset

Где чаще всего заказывавали такси?

```
SELECT pickup_ntaname FROM trips
GROUP BY pickup_ntaname
ORDER BY count() DESC
```

pickup_ntaname	count()
Midtown-Midtown South	
Hudson Yards-Chelsea-Flatiron-Union Square	
West Village	
Upper East Side-Carnegie Hill	
Turtle Bay-East Midtown	
SoHo-TriBeCa-Civic Center-Little Italy	
Upper West Side	
Murray Hill-Kips Bay	
Clinton	
Lenox Hill-Roosevelt Island	

TLC trip record dataset

Буде использовать 3 варианта хранения имен локаций

- String
- StringWithDictionary
- Enum16

Запрос	String	Dictionary	Enum16
Где чаще всего заказывавали такси?	1.696 sec.	1.465 sec.	0.317 sec.

TLC trip record dataset

Где чаще всего заказывавали такси в компании друзей?

```
SELECT pickup_ntaname FROM trips
WHERE passenger_count > 1
GROUP BY pickup_ntaname
ORDER BY count() DESC
```

pickup_ntaname	count()
Midtown-Midtown South	
Hudson Yards-Chelsea-Flatiron-Union Square	
West Village	
Upper East Side-Carnegie Hill	
Turtle Bay-East Midtown	
SoHo-TriBeCa-Civic Center-Little Italy	
Upper West Side	
Murray Hill-Kips Bay	
Clinton	

TLC trip record dataset

Запрос	String	Dictionary	Enum16
Где чаще всего заказывавали такси?	1.696 sec.	1.465 sec.	0.317 sec.
Где чаще всего заказывавали такси в компании друзей?	1.414 sec.	0.715 sec.	0.385 sec.

Почему второй запрос для StringWithDictionary работает в 2 раза быстрее?

TLC trip record dataset

Запрос	String	Dictionary	Enum16
Где чаще всего заказывавали такси?	1.696 sec.	1.465 sec.	0.317 sec.
Где чаще всего заказывавали такси в компании друзей?	1.414 sec.	0.715 sec.	0.385 sec.

Почему второй запрос для StringWithDictionary работает в 2 раза быстрее?

Фильртация столбца по условию в WHERE происходит только для индексов

TLC trip record dataset

Как найти самый популярный парк?

```
SELECT pickup_ntaname FROM trips
WHERE lower(pickup_ntaname) like '%park%'
GROUP BY pickup_ntaname
ORDER BY count() DESC
```

```
┌─pickup_ntaname─────────────────────────┐
 Battery Park City-Lower Manhattan
 park-cemetery-etc-Manhattan
 Park Slope-Gowanus
 park-cemetery-etc-Queens
 Rego Park
 Sunset Park West
 park-cemetery-etc-Brooklyn
 Baisley Park
 Bedford Park-Fordham North
```

TLC trip record dataset

Запрос	String	Dictionary	Enum16
Где чаще всего заказывавали такси?	1.696 sec.	1.465 sec.	0.317 sec.
Где чаще всего заказывавали такси в компании друзей?	1.414 sec.	0.715 sec.	0.385 sec.
Как найти самый популярный парк?	1.356 sec.	0.440 sec.	1.675 sec.

Почему вариант с Enum стал тормозить?

- LIKE не оптимизорован для Enum
- Происходит преобразование в строку

Enum требует ручной оптимизации в коде

TLC trip record dataset

Пример медленной функции

```
SELECT pickup_ntaname,
       UUIDNumToString(sipHash128(pickup_ntaname)) AS hash
FROM trips
GROUP BY hash
ORDER BY count() DESC
```

```
┌─hash─────────────────────────────────┬─count()─┐
│ 97b2232f-3b26-9e79-f0fa-5b40dc229d59 │ 69197026 │
│ 0423ed58-b11a-cec3-c602-c7afcfe6d22b │ 38314720 │
│ 77f8dcf1-8605-407f-f9de-9c9a7dc6359d │ 29425006 │
│ 390413cd-a587-c7d8-c3e6-074fff85e665 │ 28733526 │
│ 860eb0cc-b085-880d-6fe2-13c70ae91f7b │ 27900211 │
│ c01bfa22-503d-49f3-0614-9c18c1ed7010 │ 20842627 │
│ b7949119-37fc-0eec-6acd-9aa80d219e44 │ 19698235 │
│ f7671fd6-73c7-985e-326b-bc00b0df4095 │ 18929212 │
└──────────────────────────────────────┴──────────┘
```

TLC trip record dataset

Запрос	String	Dictionary	Enum16
Где чаще всего заказывавали такси?	1.696 sec.	1.465 sec.	0.317 sec.
Где чаще всего заказывавали такси в компании друзей?	1.414 sec.	0.715 sec.	0.385 sec.
Как найти самый популярный парк?	1.356 sec.	0.440 sec.	1.675 sec.
Хеш от строки	3.110 sec.	1.369 sec.	3.671 sec.

Планы

Добавить оптимизации

- Кеширование вычислений над словарями
- Выполнение GROUP BY
- Специализация агрегатных функций
- Деградация к классическому столбцу

Добиться скорости работы не ниже, чем у String в любых случаях

Неявно заменить String на StringWithDictionary (возможно)