Структуры данных для вероятностной фильтрации по подзапросам в ClickHouse

Руководитель: Миловидов Алексей Николаевич
Студент: Ибрагимов Рузель Ильфакович

ClickHouse

- Колоночная база данных
- Создана для выполнения аналитических запросов
- Имеет открытый исходный код
- Разрабатывается компанией Яндекс
- Не тормозит

Структура данных: фильтр

- это множество
- добавление элемента в множество
- проверка элемента на принадлежность множеству
- удаление элемента из множества (необязательно)
- Примеры: HashSet, HashTable, unordered_set, set

Вероятностный фильтр

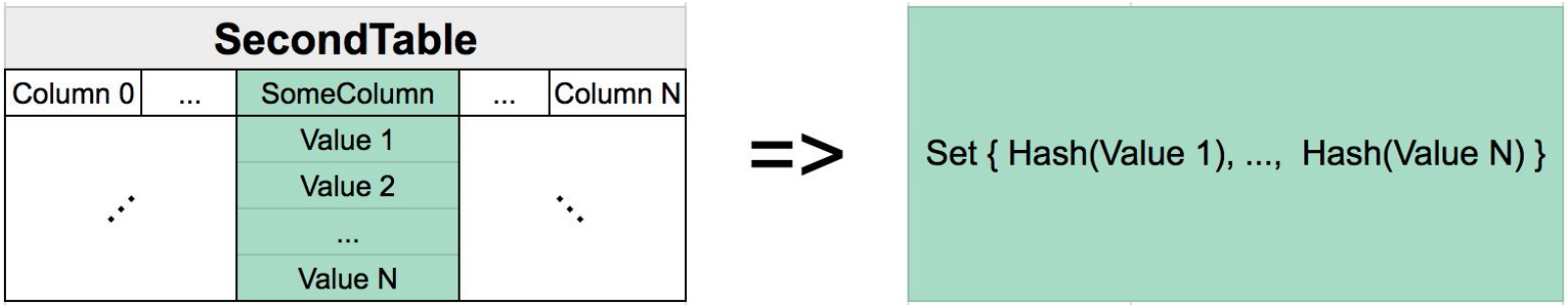
- добавление элемента в множество
- проверка элемента на принадлежность множеству (иногда может соврать)
- удаление элемента из множества (необязательно)
- остальное: размер, количество добавленных элементов, инициализация

Как фильтр используется в базах данных

- Запрос, содержащий оператор **IN:**

```
SELECT COUNT() FROM FirstTable WHERE SomeColumn IN SecondTable;
```

По колонке **someColumn** строится хэш-таблица (фильтр), затем каждый элемент первой таблицы проверяется на принадлежность хэш-таблице (фильтру).



Пример использования вероятностного фильтра

- Пусть перед аналитиком стоит задача: *срочно* посчитать *примерный* процент пользователей yandex.ru, которые были на maps.yandex.ru
- Точное решение выглядит примерно так:

```
SELECT COUNT() FROM YandexUsersInfo WHERE USER_ID IN YandexMapsUsersInfo;
```

- Пусть *примерный* = с точностью в 2 раза
- Аналитик торопится и ему важно время выполнения запроса, а не точноть

Пример использования вероятностного фильтра

- Вместо хэш-таблицы можно использовать вероятностный фильтр
- False positive, false negative
- Вероятностный фильтр чаще всего быстрее хэш-таблицы и на один элемент уходит меньше памяти

Примеры вероятностных фильтров

- Фильтр Блума - самый известный и, наверное, самый простой в написании (Бёртон Блум, 1970); А также его оптимизации
- Cuckoo filter (не путать с cuckoo hashing)
- Chaotic Map / Filter (Андрей Плахов, 2014)
- Quotient / Cascade / Vacuum filters

Возможности использования вероятностных фильтров в других БД

- **PostgreSQL** - можно создать отдельным запросом индекс из фильтра Блума, а затем использовать его
- **RocksDB** - можно в качестве хэш-таблицы передать аргументом фильтр Блума, указав количество бит на 1 элемент.
- **Cassandra** - можно создать отдельную таблицу, тип которой будет фильтр Блума
- **MySql**, **LevelDB** - нет возможности применения вероятностных фильтров

Добавление в ClickHouse вероятностной фильтрации

- Был использован фильтр Блума, а также проверен Chaotic filter
- Была добавлена возможность сделать запрос указав параметрами размер фильтра Блума и количество хэш-функций

```
SELECT COUNT()
FROM FirstTable
WHERE SomeColumn IN BLOOMFILTER
(
    SELECT SomeColumn
    FROM SecondTable
)
SETTINGS bloomfilter_storage_len = 10000000, bloomfilter_hashes_count = 1;
```

Тестирование производительности

- Для тестирования использовались таблицы с 25M целых 64 битных чисел и 15M целых 64 битных чисел
- Первая содержала перестановку {0, ... , 25M}
- Вторая содержала перестановку {-5M, …, 10M}
- Общих элементов 10M
- Выполнялась проверка вхождения элементов первой таблицы (25M) во вторую таблицу (15M)
- Числа были выбраны с учетом размера L3-кэша (~16mb)

Таблица значений тестирования

Длина, байт	Кол-во хэш функций	Улучшение, %	FP rate, %	время, с.
-	-	0	0	1,234
3M	1	75	46	0,314
5M	2	57	27,8	0,536
10M	3	17	8	1,027
13M	3	2	4,3	1,213
15M	3	4	3	1,188
15M	4	-16	2,4	1,432

Chaotic Filter

- Проблемы с реализацией из-за 64-битного хэша
- С 64-битным хэшом работает очень плохо, FP & FN ~ 95%

Заключение

- Была добавлена возможность сделать вероятностный запрос с параметрами (и ничего не падает, ура!)
- Синтаксис передачи параметров необходимо изменить
- Написать Cuckoo / Vacuum и другие фильтры
- Проверить Chaotic с 128-битным хэшем

Пример теста

```
alkorops.sas.yp-c.yandex.net :) SELECT (count() - 10000000) / 15000000 * 100 AS FP_perc FROM M25Ints WHERE number IN BLOOMFILTER (SELECT number FROM M1
5Ints) SETTINGS bloomfilter_storage_len=15000000, bloomfilter_hashes_count=3;

SELECT ((count() - 10000000) / 15000000) * 100 AS FP_perc
FROM M25Ints
WHERE number IN BLOOMFILTER
(
    SELECT number
    FROM M15Ints
)
SETTINGS bloomfilter_storage_len = 15000000, bloomfilter_hashes_count = 3

┌─FP_perc─┐
│  3.0467 │
└─────────┘

1 rows in set. Elapsed: 1.162 sec. Processed 40.00 million rows, 320.00 MB (34.42 million rows/s., 275.38 MB/s.)

alkorops.sas.yp-c.yandex.net :) SELECT (count() - 10000000) / 15000000 * 100 AS FP_perc FROM M25Ints WHERE number IN BLOOMFILTER (SELECT number FROM M1
5Ints) SETTINGS bloomfilter_storage_len=10000000, bloomfilter_hashes_count=3;

SELECT ((count() - 10000000) / 15000000) * 100 AS FP_perc
FROM M25Ints
WHERE number IN BLOOMFILTER
(
    SELECT number
    FROM M15Ints
)
SETTINGS bloomfilter_storage_len = 10000000, bloomfilter_hashes_count = 3

┌─────────FP_perc─┐
│ 7.979033333333334 │
└─────────────────┘

1 rows in set. Elapsed: 1.134 sec. Processed 40.00 million rows, 320.00 MB (35.27 million rows/s., 282.17 MB/s.)
```