Краткий Обзор СУБД Clickhouse

Раздаточный материал к первой части 7-го урока, посвященному изучению СУБД Clickhouse.

|  |  |
| --- | --- |
| СОДЕРЖАНИЕ | |
| 1 | Обзор Clickhouse |
| 2 | Базы данных в Clickhouse |
| 3 | Движки таблиц. Семейство MergeTree |
| 4 | Индексы |
| 5 | Типы данных и мутации данных |
| 6 | Репликация и шардирование |
| 7 | Использование словарей |
| 8 | Использование JOIN |
| 9 | Источники информации (ссылки) |

Обзор Clickhouse

**Clickhouse** – это колоночная аналитическая СУБД с открытым кодом, позволяющая выполнять аналитические запросы в режиме реального времени на структурированных больших данных.

В ней можно хранить витрины данных, к которым имеют доступ тысячи сотрудников. Также она может использоваться для аналитики над событиями и логами. Или быть базой данных для временных рядов. Среди основных свойств можно выделить быстроту выполнения запросов, работоспособность при большом количестве пользователей, отказоустойчивость и масштабируемость, быструю вставку данных и возможность интеграция со множеством внешних источников. Также следует отметить отсутствие транзакций и её нерезидентность (все данные хранятся на дисках в виде файлов).

Clickhouse хорошо подходит для быстрых запросов к широким таблицам фактов и к витринам. Он хорошо работает с where условиями и быстро строит агрегаты, в нем можно гибко задавать схемы шардирования и репликации. К недостаткам можно отнести отсутствие транзакций и особенности применения джоинов.

Базы данных в Clickhouse

По умолчанию в Clickhouse есть системная БД system и база по умолчанию default (задаётся при установке). Список БД кластера можно увидеть в системной таблице system.databases. Создаются базы данных запросом CREATE DATABASE с указанием её имени и движка. В таблице приведены различные движки баз данных. Движок Atomic является основным. Каждая таблица имеет уникальный идентификатор и хранит данные в определённой папке. На каждую таблицу также создаётся ссылка, за счёт чего запросы DROP и RENAME TABLE не являются блокирующими.

Изображение выглядит как текст

Автоматически созданное описание

Запросы RENAME выполняются без изменения UUID и перемещения табличных данных. Эти запросы не ожидают завершения использующих таблицу запросов и выполняются мгновенно. Запросы EXCHANGE атомарно меняет местами две таблицы. Например, вместо неатомарной операции: RENAME TABLE вы можете использовать один атомарный запрос: EXCHANGE TABLES.



При выполнении запроса DROP TABLE данные не удаляются немедленно. Таблица помечается как удаленная, метаданные перемещаются в специальную папку /var/lib/clickhouse/metadata\_dropped/, а база данных уведомляет об этом фоновый поток. Задержка перед окончательным удалением данных задается специальной настройкой database\_atomic\_delay\_before\_drop\_table\_sec. Можно активировать синхронный режим с помощью другой настройки database\_atomic\_wait\_for\_drop\_and\_detach\_synchronously. В этом случае таблица будет ждать только завершения SELECT, INSERT и других запросов, которые используют таблицу. Она будет сразу удалена, когда перестанет использоваться.

Также в таблице указаны другие движки, например для доступа к данным MySQL и PostrgeSQL.

Движки таблиц. Семейство MergeTree

**Семейство MergeTree** – основное семейство движков для хранения больших данных (они являются наиболее функциональными).

Рассмотрим обычный движок MergeTree. Когда у вас есть огромное количество данных, которые должны быть вставлены в таблицу, вы должны быстро записать их по частям, а затем объединить части по некоторым правилам в фоновом режиме. То есть основная идея, которая заложена в основу этого семейства состоит в том, чтобы при вставке данных записывать их в виде отдельных частей, а не единым целым, а затем объединить эти части поэтапно между собой в фоновом режиме.

Такие таблицы поддерживают партиционирование (можно оперировать партициями), репликацию и семплирование данных, и так как они хранят данные, отсортированные по первичному ключу, то поиск в таблицах оптимизирован при помощи разрежённого индекса.

Единицей хранения в таких таблицах является кусок. Clickhouse пишет данные в куски и постепенно сливает их в один. Хранение куска доступно как в истинно колоночном формате, так и в компактном (все колонки хранятся в одном файле). Присутствует мультиверсионность данных, в конкретный момент времени может храниться более одной копии – неслитые между собой куски и их объединенный вариант. Старые куски, данные которых были слиты в новый, постепенно удаляются - на срок существования влияет табличная настройка old\_parts\_lifetime. Каждый кусок хранится в отдельной директории внутри папки таблицы. Куски разных партиций не сливаются между собой. С кусками доступны операции перемещения, отсоединения и т.д, как и с партициями.

Рассмотрим механизм слияния кусков:

Каждый вставленный в базу кусок встаёт в очередь на слияние (merge). Все куски, находящиеся на диске, сортируются в порядке возрастания первичного ключа. Когда накапливается достаточно кусков похожего объема - запускается merge, и они объединяются в один новый отсортированный кусок большего размера. При слиянии образуется новый уровень хранения, содержащий новые куски. Старый уровень хранения помечается как неактивный, но не удаляется, пока не завершены запросы, использующие его в качестве источника данных. Новые запросы получают данные из кусков нового активного уровня. Система стремится к состоянию, когда для партиции есть всего один отсортированный кусок данных. Куски разных партиций не объединяются. Процесс слияния управляется системой, но можно его форсировать командой OPTIMIZE TABLE для принудительного внепланового объединения кусков таблицы или для конкретной партиции. Если указать FINAL, то оптимизация выполняется даже в том случае, если все данные уже лежат в одном куске данных. Также в таком случае процесс слияния идёт до конца. Старые куски удаляются с дисков. Кроме того, слияние является принудительным, даже если выполняются параллельные слияния.

Перечислим другие особенности, связанные с механизмом хранения:

Данные необходимо вставлять небольшим количеством больших кусков. ClickHouse не предназначен для OLTP нагрузки. Рекомендуется вставлять данные раз в секунду, миллионными пачками. В системе работает защита от слишком частых вставок: если запущены 50 одновременных merge-процессов - база не даст вам вставить новый набор данных. Для конкурентного доступа к таблице используется многослойность. При одновременном чтении и обновлении таблицы, данные будут читаться из набора кусков, актуального на момент запроса, а не из вновь формируемого. Чтения из таблицы автоматически распараллеливаются по партициям и кускам. Нет длинных блокировок. Вставки не мешают чтению.

Изображение выглядит как текст

Автоматически созданное описание

Рассмотрим подробно скрипт создания таблицы. Таблица создаётся по команде CREATE TABLE. Необходимо указать базу данных и имя создаваемой таблицы, имена полей с их типами. DEFAULT expr может задать значение по умолчанию. Если в запросе INSERT не указан соответствующий столбец, то он будет заполнен путём вычисления соответствующего выражения.

Поле с MATERIALIZED expr не может быть указан при INSERT, то есть, он всегда вычисляется. При INSERT без указания списка столбцов, такие столбцы не рассматриваются. Также этот столбец не подставляется при использовании звёздочки в запросе SELECT.

ALIAS expr столбец вообще не хранится в таблице. Его значения не могут быть вставлены в таблицу, он не выводится при использовании \* в запросе SELECT. Поле должно быть явно указано в запросе SELECT. В таком случае, во время разбора запроса, алиас раскрывается.

TTL expr1 определяет время жизни значений для столбца. Когда срок действия значений в столбце истечёт, ClickHouse заменит их значениями по умолчанию для заданного типа данных столбца. Если срок действия всех значений столбца в части данных истек, ClickHouse удаляет столбец из куска данных в файловой системе. Секцию TTL нельзя использовать для ключевых столбцов. Пример задания TTL указан на слайде.

INDEX expr1 это индексы пропуска данных. Индексы агрегируют для заданного выражения некоторые данные, а потом при SELECT запросе используют для пропуска блоков данных, на которых секция WHERE не может быть выполнена, тем самым уменьшая объём данных, читаемых с диска (пропускаемый блок состоит из гранул данных в количестве равном гранулярности данного индекса). При создании индекса в папках кусков появляются дополнительные файлы с засечками и смещениями. Выполнение запроса для создания индекса не приводит к его непосредственному появлению, индекс будет вычислен только при последующем слиянии или принудительно запросе следующего вида. Наиболее часто используемые типы вторичных индексов:

* **minmax** — хранит минимум и максимум выражения, используя их для пропуска блоков аналогично первичному ключу.
* **set(max\_rows)** — хранит уникальные значения выражения на блоке в количестве не более max\_rows, используя их для пропуска блоков, оценивая выполнимость where выражения на хранимых данных.
* **bloom\_filter([false\_positive])** — фильтр Блума для указанных столбцов.

Необязательный параметр false\_positive — это вероятность получения ложноположительного срабатывания. Возможные значения: (0, 1). Значение по умолчанию: 0.025.

Далее указывается название движка. Могут приниматься параметры и есть регистрозависимость. Список движков можно смотреть в системной таблице system.table\_engines. Движок MergeTree не имеет параметров.

Ключ сортировки. В виде одной или нескольких колонок. Может не задаваться, если нет намерения расширить сортировку на большее число колонок, чем указано в первичном ключе. ClickHouse использует ключ сортировки в качестве первичного ключа, если первичный ключ не задан в секции PRIMARY KEY.

Далее можно задать ключ партиционирования. В такой случае таблица создается партиционированной. Для партиционирования по месяцам используйте выражение, указанное на слайде, где date\_column — столбец с датой типа Date.

Первичный ключ. Если он отличается от ключа сортировки. Это необязательный параметр. По умолчанию первичный ключ совпадает с ключом сортировки (который задаётся секцией ORDER BY). Поэтому в большинстве случаев секцию PRIMARY KEY отдельно указывать не нужно.

Ключ семплирования в секции SAMPLE позволяет выполнять запросы приближённо, сокращая объём считываемых с диска данных. Например, чтобы посчитать статистику по всем посещениям, можно обработать 1/10 общего количества и результат домножить на 10. Сэмплирование имеет смысл, когда точность результата не важна, например, для оценочных расчетов. Также, когда возможности аппаратной части не позволяют соответствовать строгим критериям. Например, время ответа должно быть < 100 мс. То есть в случаях, когда точность расчета имеет более низкий приоритет. И когда точность результата участвует в бизнес-модели какого-либо сервиса. Например, пользователи с бесплатной подпиской на сервис могут получать отчеты с меньшей точностью, чем другие пользователи с премиум подпиской.

Если задано выражение SAMPLE k, запрос будет выполнен для k доли данных. k – число в интервале от 0 до 1. Поддерживается как дробная, так и десятичная форма записи. Если задано выражение SAMPLE n, запрос будет выполнен для выборки из не менее n строк (но не значительно больше этого значения). n – это достаточно большое целое число. Виртуальный столбец \_sample\_factor автоматически создается в тех таблицах, для которых задано выражение SAMPLE BY. При задании SAMPLE k OFFSET m, выборка будет сформирована из k доли данных со смещением на долю m. Здесь k и m – числа в интервале от 0 до 1. Поддерживается как дробная, так и десятичная форма записи. В первом примере выборка будет сформирована по 1/10 доле всех данных. Во втором выборка, которая состоит из 1/10 доли данных, взята из второй половины данных.

TTL expr – это список правил, определяющих длительности хранения строк, а также задающих правила перемещения частей на определённые тома или диски. Выражение должно возвращать столбец Date или DateTime. Типы правил, приведённых на слайде, указывают действия, которые будут выполнены с частью: удаление строк (прореживание), перемещение на определённый диск или том, или агрегирование данных в устаревших строках. В секции WHERE можно задать условие удаления или агрегирования устаревших строк (для перемещения и сжатия условие WHERE не применимо).

Отметим другие разновидности движков MergeTree. Помимо стандартного, существуют ещё следующие:

* ReplacingMergeTree, который помимо всего прочего, выполняет ещё и удаление дублирующихся записей с одинаковым значением ключа сортировки. Дедупликация данных производится лишь во время слияний.
* SummingMergeTree при слиянии кусков данных все строки с одинаковым ключом сортировки заменяет на одну, которая хранит только суммы значений из столбцов с цифровым типом данных.
* CollapsingMergeTree добавляет в алгоритм слияния кусков данных дополнительную логику сворачивания или удаления строк. CollapsingMergeTree асинхронно сворачивает пары строк, если все поля в ключе сортировки эквивалентны, но за исключением специального поля Sign, которое может принимать значения 1 и -1. Строки без пары сохраняются. Если Sign = 1, то это означает, что строка является состоянием объекта, назовём её строкой состояния. Если Sign = -1, то это означает отмену состояния объекта с теми же атрибутами, назовём её строкой отмены состояния.

Например, мы хотим рассчитать, сколько страниц проверили пользователи на каком-то сайте и как долго они там находились. В какой-то момент времени мы записываем строку с состоянием действий пользователя. Через некоторое время мы регистрируем изменение активности пользователя и записываем уже две строки с разными sign. Вторая строка содержит текущее состояние, которая и останется при последующих слияниях.

* VersionedCollapsingMergeTree движок предназначен для тех же задач, что и CollapsingMergeTree, но с другим алгоритмом свёртывания, который позволяет вставлять данные в любом порядке в несколько потоков. Столбец Version помогает свернуть строки правильно, даже если они вставлены в таблицу в неправильном порядке. CollapsingMergeTree движок требует строго последовательную вставку всех данных.
* В AggregatingMergeTree все строки с одинаковым  ключом сортировки заменяется на одну, которая хранит объединение состояний агрегатных функций.

Подробнее с этими движками можете ознакомиться по ссылке на официальную документацию Clickhouse в конце в разделе Источники информации.

Log движки разработаны для сценариев, когда необходимо быстро записывать много таблиц с небольшим объёмом данных (менее 1 миллиона строк), а затем читать их целиком. Они бывают трех типов:

* Типичный способ использования TinyLog движка — записать сначала данные один раз, а затем читать их столько раз, сколько это необходимо. Запросы выполняются в один поток. То есть, этот движок предназначен для сравнительно маленьких таблиц (до 1 000 000 строк).
* Движок StripeLog хранит все столбцы в одном файле. При каждом запросе INSERT, ClickHouse добавляет блок данных в конец файла таблицы, записывая столбцы один за другим. ClickHouse использует несколько потоков при выборе данных. Каждый поток считывает отдельный блок данных и возвращает результирующие строки независимо по мере завершения.
* Log отличается тем, что вместе с файлами столбцов лежит небольшой файл "засечек". Засечки пишутся на каждый блок данных и содержат смещение: то есть с какого места нужно читать файл, чтобы пропустить заданное количество строк. Это позволяет читать данные из таблицы в несколько потоков. При конкурентном доступе к данным чтения могут выполняться одновременно, а если при записи в таблицу произошёл сбой, то таблица станет битой, и чтения из нее будут возвращать ошибку.

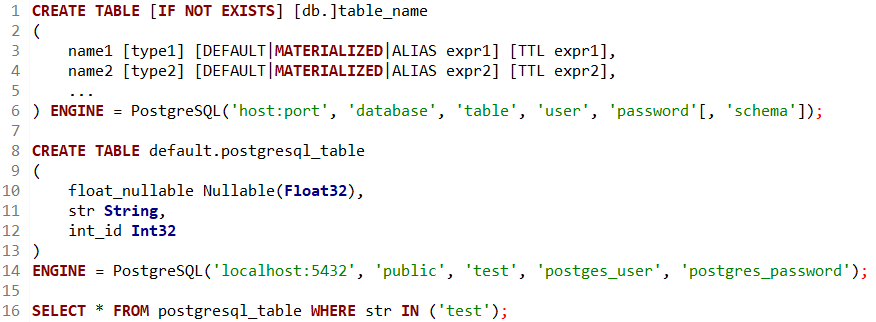
Основные свойства движков Семейства Log:

1. Хранят данные на диске.
2. Добавляют данные в конец файла при записи.
3. Поддерживают блокировки для конкурентного доступа к данным.
4. Не поддерживают операции мутации.
5. Не поддерживают индексы.
6. Записывают данные не атомарно.

Перейдём к интеграционным движкам. Для интеграции с внешними системами ClickHouse предоставляет различные средства, включая движки таблиц. Конфигурирование интеграционных движков осуществляется с помощью запросов CREATE TABLE или ALTER TABLE, как и у других табличных движков. Настроенная интеграция выглядит как обычная таблица, но запросы к ней передаются через прокси во внешнюю систему.

* ODBC – подключение к внешним базам данных с помощью драйвера ODBC.
* JDBC – подключение к внешним базам данных с помощью драйвера JDBC.
* MySQL – позволяет выполнять запросы SELECT и INSERT над данными, хранящимися на удалённом MySQL сервере.
* MongoDB – позволяет читать данные из коллекций СУБД MongoDB. Запись (INSERT запросы) не поддерживается.
* HDFS – обеспечивает интеграцию с экосистемой Apache Hadoop, позволяя управлять данными в HDFS посредством CH.
* S3 – обеспечивает интеграцию с экосистемой Amazon S3.
* Kafka – работает с Apache Kafka.
* EmbeddedRocksDB – позволяет интегрировать ClickHouse с rocksdb.
* RabbitMQ – работает с RabbitMQ.
* **PostgreSQL** – позволяет выполнять запросы SELECT и INSERT для таблиц на удаленном сервере PostgreSQL.

Имена столбцов должны быть такими же, как в исходной таблице, можно использовать некоторые из этих столбцов и в любом порядке. Типы столбцов могут отличаться от типов в исходной таблице. Система пытается привести значения к типам данных ClickHouse. Простые условия для WHERE исполняются на стороне PostgreSQL. Операции объединения, агрегации, сортировки и ограничения LIMIT выполняются на стороне ClickHouse. Указанная настройка определяет, как обрабатывать Nullable столбцы. Значение настройки по умолчанию равно 1. Если значение 0, то табличная функция вместо NULL выставляет значения по умолчанию для скалярного типа. Приведём пример реализации таблицы с таким движком.



Остальные движки таблиц уникальны по своему назначению и еще не сгруппированы в семейства, поэтому они помещены в эту специальную категорию. Кратко опишем их.

* **Distributed**. Не хранит данные самостоятельно, а позволяет обрабатывать запросы распределённо, на нескольких серверах. Чтение автоматически распараллеливается. При чтении будут использованы индексы таблиц на удалённых серверах, если есть.
* **Merge.** Не хранит данные самостоятельно, а позволяет читать одновременно из произвольного количества других таблиц. Чтение автоматически распараллеливается. Запись в таблицу не поддерживается. При чтении будут использованы индексы используемых таблиц.
* **Set**. Представляет собой множество, постоянно находящееся в оперативке. Предназначено для использования в правой части оператора IN (делать SELECT нельзя). В таблицу можно вставлять данные INSERT-ом.
* **Buffer.** Буферизует записываемые данные в оперативке, периодически сбрасывая их в другую таблицу. При чтении производится чтение данных одновременно из буфера и из другой таблицы.
* **MaterializedView.** Для реализации материализованных представлений. Для хранения данных использует другой движок, который был указан при создании представления.
* **Null.** При записи в таблицу типа Null, данные игнорируются. При чтении из таблицы типа Null, возвращается пустота.
* **Memory.** Хранит данные в оперативке, в несжатом виде. Данные хранятся именно в таком виде, в каком они получаются при чтении.
* **View**. Для реализации представлений. Не хранит данные, а хранит только указанный запрос SELECT.
* **Join**. Подготовленная структура данных для использования в операциях JOIN.
* **File**. Управляет данными в одном файле на диске в указанном формате.
* **Dictionary**. Отображает данные словаря как таблицу ClickHouse.
* **URL**. Управляет данными на удаленном HTTP/HTTPS сервере.

Индексы

В Clickhouse есть два вида индексов – первичный и вторичный. У таблиц MergeTree есть первичный ключ. Для каждой таблицы может быть только один. Он не обеспечивает уникальности, дубликаты строк с одним первичным ключом могут быть вставлены в таблицы. Первичный ключ используется в качестве индекса для поиска блоков данных. Основная задача такого индекса — иметь возможность отфильтровать большую таблицу с данными.

Ограничений на объём ключа нет, но длинный первичный ключ будет негативно влиять на производительность вставки и потребление памяти. Поэтому в ряде случаев его делают меньше ключа сортировки. На производительность ClickHouse при запросах SELECT лишние столбцы в первичном ключе не влияют. Для каждой колонки в одном куске есть как минимум два файла: – файл с данными колонки и файл с засечками - метками индекса для набора данных.

Также для таблиц семейства MergeTree можно задать дополнительные индексы в секции столбцов (мы уже рассмотрели это ранее). Такие индексы агрегируют для заданного выражения некоторые данные, а потом при SELECT запросе используют для пропуска блоков данных. Пропускаемый блок данных состоит из гранул в количестве, равном гранулярности данного индекса. При создании индекса в папках кусков появляются дополнительные файлы с засечками и смещениями.

Выполнение запроса для создания индекса не приводит к его непосредственному появлению, сам индекс будет вычислен только при последующем слиянии или может быть вычислен принудительно с помощью указанного запроса. Наиболее часто используемые вторичные индексы перечислялись ранее. Подробнее их можно изучить в официальной документации.

Типы данных и мутации данных

Изображение выглядит как стол

Автоматически созданное описание

* [AggregateFunction | ClickHouse Docs](https://clickhouse.com/docs/ru/sql-reference/data-types/aggregatefunction)
* [SimpleAggregateFunction(func, type) | ClickHouse Docs](https://clickhouse.com/docs/ru/sql-reference/data-types/simpleaggregatefunction)

Рассмотрим возможные манипуляции с таблицами и данными посредством операции ALTER. Мы можем манипулировать столбцами таблицы – добавлять и удалять колонки, очищать и комментировать их, изменять тип колонки. Есть возможность менять ключ сортировки, добавлять или удалять вторичный индекс, ограничения, манипулировать партициями или кусками – с возможными операторами ознакомьтесь самостоятельно.

Изображение выглядит как текст

Автоматически созданное описание





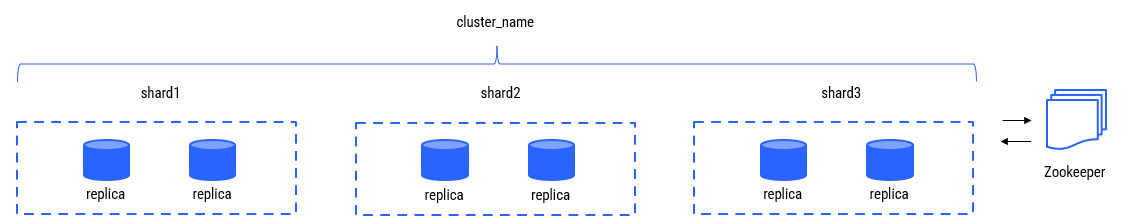
Изображение выглядит как текст

Автоматически созданное описание

Мутации – это разновидность запроса ALTER, позволяющая изменять или удалять данные в таблице. В отличие от стандартных запросов DELETE и UPDATE, рассчитанных на точечное изменение данных, применения мутаций – это достаточно тяжелые изменения, затрагивающие большое число строк в таблице.

Репликация и шардирование

Рассмотрим процессы репликации и шардирования данных. Смысл репликации заключается в том, что данные с одного сервера базы данных постоянно копируются (т. е. реплицируются) на один или несколько других (называемых репликами). Шардирование — это стратегия горизонтального масштабирования кластера, при которой части одной базы данных ClickHouse размещаются на разных шардах. Шард состоит из одного или нескольких хостов-реплик.



Запрос на запись или чтение в шард может быть отправлен на любую его реплику. Репликация не зависит от шардирования. На каждом шарде репликация работает независимо. Шардирование и репликация может гибко настраиваться отдельно для каждой таблицы. Репликация в Clickhouse работает посредством Apache Zookeeper, где хранятся метаданные о репликах.

Репликация работает на уровне отдельных таблиц, а не всего сервера. То есть, на сервере могут быть расположены одновременно реплицируемые и не реплицируемые таблицы. Реплицируются данные запросов INSERT и ALTER. Запросы CREATE, DROP, ATTACH, DETACH и RENAME выполняются на одном сервере и не реплицируются. То есть запрос CREATE TABLE создаёт новую реплицируемую таблицу на том сервере, где его выполнили.

Если таблица уже существует на других серверах, запрос добавляет новую реплику. DROP TABLE удаляет реплику, расположенную на том сервере, где выполняется запрос. Запрос RENAME переименовывает таблицу на одной из реплик. Другими словами, реплицируемые таблицы на разных репликах могут называться по-разному.

Для создания реплицируемых таблиц на всех хостах необходимо указать кластер zookeeper`а. Можно указать любой имеющийся у вас ZooKeeper-кластер - система будет использовать в нём одну директорию для своих данных (директория указывается при создании реплицируемой таблицы). Если в конфигурационном файле не настроен ZooKeeper, то вы не сможете создать реплицируемые таблицы.

Репликация поддерживается только для таблиц семейства MergeTree с приставкой Replicated. Если вы добавляете новую реплику после того, как таблица на других репликах уже содержит некоторые данные, то после выполнения запроса, данные на новую реплику будут скачаны с других реплик. То есть, новая реплика синхронизирует себя с остальными.

Используемый движок имеет 2 параметра – директория в Zookeeper с указанием шарда и имя реплики. В параметрах могут использоваться переменные в фигурных скобках – макросы. Для каждого сервера они задаются в его конфигурационном файле. Можно посмотреть макросы для хоста, выбрав данные из системной таблицы system.macros.

Шардирование данных между таблицами реализуется при помощи движка Distributed. Таблицы с движком Distributed не хранят данные, а позволяют обрабатывать запросы распределенно, на нескольких серверах. Движок Distributed принимает параметры: имя кластера в конфигурационном файле сервера, имя удалённой базы данных, имя удалённой таблицы, ключ шардирования (не обязательно).

У каждого шарда в конфигурационном файле может быть задан «вес» (weight). По умолчанию, вес равен единице. Данные будут распределяться по шардам в количестве, пропорциональном весу шарда.

Кластер в конфигурационном файле – это логическая сущность, объединяющая хосты. Таких кластеров для одной машины может быть сколько угодно. Таким образом достигается гибкость распределения данных между хостами. В системной таблице clusters можно посмотреть устройство созданных кластеров.

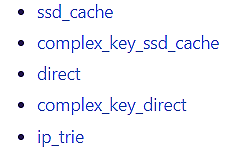
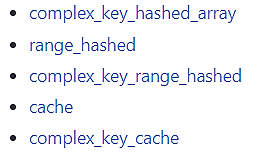
Использование словарей

Словарь — это отображение данных в виде «ключ -> атрибуты», которое удобно использовать для различного вида справочников. ClickHouse поддерживает специальные функции для работы со словарями, которые можно использовать в запросах. Проще и эффективнее вызывать в SQL запросах словари, чем выполнять JOIN’ы с таблицами-справочниками. Существует возможность создавать собственные словари из различных источников данных.

Источником данных для словаря может быть локальный текстовый/исполняемый файл, HTTP(s) ресурс или другая СУБД. ClickHouse полностью или частично хранит словари в оперативной памяти. Позволяет создавать внешние словари с помощью xml-файлов или DDL-запросов. Путь к конфигурации указывается в указанном параметре dictionaries\_config. Словари могут загружаться при старте сервера или при первом использовании, в зависимости от следующей настройки dictionaries\_lazy\_load. Системная таблица system.dictionaries содержит информацию о словарях.

Есть различные способы хранения словарей:

**Flat** подразумевает под собой случай, когда словарь полностью хранится в оперативной памяти в виде плоских массивов. Объём памяти, занимаемой словарём пропорционален размеру самого большого по размеру ключа. Также словарь **Hashed** может полностью храниться в оперативной памяти в виде хэш-таблиц. Для каждой колонки создаётся своя хеш-таблица. **Cache** словарь хранится в кэше, состоящем из фиксированного количества ячеек. Ячейки содержат часто используемые элементы. Словарь типа cache показывает высокую производительность лишь при достаточно больших hitrate (рекомендуется 99% и выше). Хитрейт – это частота успешных обращений. Посмотреть средний hitrate можно в таблице system.dictionaries. Существует множество других способов хранения словарей, про которые можно почитать в документации Clickhouse по ссылке в Источниках информации.



Создаются словари по команде CREATE DICTIONARY, после указания ключей и атрибутов также задаётся первичный ключ, источник данных, способ хранения, интервал обновления в секундах, какие-либо настройки и любой комментарий к словарю или название.

Изображение выглядит как стол

Автоматически созданное описание

Время обновления задаётся одним числом или интервалом. Если lifetime = 0, то обновление словаря производится не будет. Можно задать интервал, внутри которого ClickHouse равномерно-случайно выберет время для обновления. Можно принудительно обновить словарь, если выполнить команду SYSTEM RELOAD DICTIONARY. Существует множество вариантов для выбора источника данных, для которых нужно прописывать определенные параметры. Также ClickHouse поддерживает иерархические словари с числовыми ключом. ClickHouse поддерживает специальное свойство  HIERARCHICAL для атрибутов  словаря.

Использовать словари можно эффективно в запросах SELECT, применяя различные функции. Например, можно получить значение атрибута из словаря по определённому значению ключа. Можно просто проверить, существует ли запись с указанным ключом. Есть возможность вывести цепочку предков для заданного ключа в иерархическом словаре или вернуть потомка первого уровня. Почитать про другие функции и посмотреть примеры создания словарей можно по ссылке в Источниках информации.

Использование JOIN

Рассмотрим особенности применения join’ов. JOIN фактически создаёт новую таблицу путем объединения столбцов из одной или нескольких таблиц. Помимо стандартных типов джоинов поддерживаются также следующие:

**LEFT/RIGHT SEMI JOIN** выбирает только строки из левой таблицы, которые имеют соответствия в правой. Он соединяет две таблицы по заданным условиям, по которым строки из правой таблицы фильтруются и на выходе остаются только соответствующие им строки из левой таблицы.

**LEFT/RIGHT ANTI JOIN** выбирает только строки из левой таблицы, которые НЕ имеют соответствия в правой.

**LEFT/RIGHT/INNER ANY JOIN** частично или полностью убирает декартово произведение.

Для работы алгоритма **ASOF/LEFT ASOF JOIN** необходим специальный столбец в таблицах. Этот столбец должен содержать упорядоченную последовательность, может быть одного из следующих типов: Int, UInt, Float, Date, DateTime, Decimal, и не может быть единственным столбцом в секции JOIN. ASOF JOIN принимает метку времени пользовательского события из table\_1 и находит такое событие в table\_2 метка времени которого наиболее близка к метке времени события из table\_1 в соответствии с условием на ближайшее совпадение.

Есть такое понятие как строгость джоина. При значении ALL — если в правой таблице несколько совпадающих строк, данные умножаются на количество этих строк. Это нормальное поведение JOIN как в стандартном SQL. При значении ANY — если в правой таблице несколько соответствующих строк, то соединяется только первая найденная.

Если в настройке join\_default\_strictness не проставлено значение и если ALL или ANY не указаны в запросе, то ClickHouse генерирует исключение, join не отработает. В секции USING может указаться один или несколько столбцов для соединения, что обозначает условие на равенство этих столбцов.

То, как будет выполняться соединение, задаётся специальной опцией join\_algorithm: hash — используется алгоритм соединения хешированием, partial\_merge — используется алгоритм соединения слиянием сортированных списков, prefer\_partial\_merge — используется алгоритм соединения слиянием сортированных списков, когда это возможно, auto — сервер ClickHouse пытается на лету заменить алгоритм hash на merge, чтобы избежать переполнения памяти.

При использовании алгоритма hash правая часть JOIN загружается в оперативную память. При использовании алгоритма partial\_merge сервер сортирует данные и сбрасывает их на диск. Работа merge в ClickHouse немного отличается от классической реализации. Система сортирует правую таблицу по блокам на основе ключей соединения и для готовых блоков строит индексы min-max. Затем сортирует куски левой таблицы на основе ключей соединения и объединяет их с правой таблицей операцией JOIN. Индексы используются для пропуска тех блоков из правой таблицы, которые не участвуют в данной операции JOIN.

По умолчанию ClickHouse использует алгоритм hash join. ClickHouse берет правую таблицу и создает для нее хеш-таблицу в оперативной памяти. При автоматическом алгоритме после некоторого порога потребления памяти система переходит к алгоритму merge.

Если вы хотите ограничить потребление памяти во время выполнения операции JOIN, используйте настройки max\_rows\_in\_join — ограничивает количество строк в хеш-таблице и max\_bytes\_in\_join — ограничивает размер хеш-таблицы. По достижении любого из этих ограничений ClickHouse действует в соответствии с настройкой join\_overflow\_mode.

При запуске JOIN, отсутствует оптимизация порядка выполнения по отношению к другим стадиям запроса. Соединение (поиск в «правой» таблице) выполняется до фильтрации в WHERE и до агрегации. Каждый раз для выполнения запроса с одинаковым JOIN, подзапрос выполняется заново — результат не кэшируется. В некоторых случаях более эффективно использовать IN вместо JOIN. Или использовать словари.

Есть два пути для выполнения соединения с участием распределённых таблиц. При использовании обычного JOIN , запрос отправляется на удалённые серверы. На каждом из них выполняются подзапросы для формирования «правой» таблицы, и с этой таблицей выполняется соединение. То есть, «правая» таблица формируется на каждом сервере отдельно. При использовании GLOBAL ... JOIN, сначала сервер-инициатор запроса запускает подзапрос для вычисления правой таблицы. Эта временная таблица передаётся на каждый удалённый сервер, и на них выполняются запросы с использованием переданных временных данных.

Для Ареновской версии Greenplum разработан специальный платный Tkhemali Connector, который позволяет параллельно и транзакционно интегрировать эти 2 системы. Он используется для быстрой перегрузки таблиц из ADB в QuickMarts. Благодаря этому Connector большое количество пользователей обращаясь к Clickhouse, одновременно могут быстро получать данные из витрин, построенных в основном хранилище Greenplum. Для записи данных в таблицу Clickhouse в Greenplum создается внешняя WRITABLE таблица с соответствующей строкой подключения. Вставляя данные в эту внешнюю таблицу, данные загрузятся в соответствующую таблицу Clickhouse.

Стоит подчеркнуть, что в Clickhouse отсутствует возможность написания пользовательских функций.

Источники информации

1. [Введение | ClickHouse Docs](https://clickhouse.com/docs/category/%D0%B2%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5)
2. [Движки таблиц | ClickHouse Docs](https://clickhouse.com/docs/ru/engines/table-engines)
3. [Движки баз данных | ClickHouse Docs](https://clickhouse.com/docs/ru/engines/database-engines)
4. [SQL выражения в ClickHouse | ClickHouse Docs](https://clickhouse.com/docs/ru/sql-reference/statements)
5. [Синтаксис | ClickHouse Docs](https://clickhouse.com/docs/ru/sql-reference/syntax)
6. [Распределенные DDL запросы (секция ON CLUSTER) | ClickHouse Docs](https://clickhouse.com/docs/ru/sql-reference/distributed-ddl)
7. [Функции | ClickHouse Docs](https://clickhouse.com/docs/ru/sql-reference/functions)
8. [Агрегатные функции | ClickHouse Docs](https://clickhouse.com/docs/ru/sql-reference/aggregate-functions)
9. [Табличные функции | ClickHouse Docs](https://clickhouse.com/docs/ru/sql-reference/table-functions)
10. [Словари | ClickHouse Docs](https://clickhouse.com/docs/ru/sql-reference/dictionaries)
11. [Типы данных | ClickHouse Docs](https://clickhouse.com/docs/ru/sql-reference/data-types)
12. [Операторы | ClickHouse Docs](https://clickhouse.com/docs/ru/sql-reference/operators)
13. [Системные таблицы | ClickHouse Docs](https://clickhouse.com/docs/ru/operations/system-tables)
14. [Коннектор ADB и Clickhouse — Arenadata documentation](https://docs.arenadata.io/adb/adbClickhouse/index.html)