Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»

Факультет компьютерных наук
Основная образовательная программа
«Прикладная математика и информатика»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Программный проект на тему

Улучшение интеграции ClickHouse с внешними базами данных

Выполнил студент группы БПМИ165, 4 курса,
Бобровский Артемий Андреевич

Руководитель ВКР:
Приглашенный преподаватель, Базовая кафедра Яндекс,
Миловидов Алексей Николаевич

Москва, 2020

Аннотация

Пользователи используют различные системы управления базами данных для решения своих задач, что делает возможность интеграции с внешними данными важным направлением разработки любой СУБД.

Работа посвящена улучшению интеграции аналитической СУБД ClickHouse с внешними базами данных. В ClickHouse уже существует архитектура, упрощающая обращение к таблицам, созданным в других СУБД, но далеко не вся необходимая функциональность реализована.

В процессе работы изучена архитектура ClickHouse, а также решения для интеграции внешних данных, использованные в других СУБД. На основе полученных знаний, внесены изменения в исходный код ClickHouse.

Как результат работы, поддержана возможность записи данных в табличную функцию ODBC, а также реализованы доступные для чтения таблицы ClickHouse на основе существующих коллекций MongoDB. Реализованный функционал будет использован в проекте ClickHouse.

Ключевые слова: СУБД, интеграция данных, C++, внешние базы данных, ODBC, MongoDB

Abstract

Users rely on various database management systems to solve their problems, which makes the ability to integrate with external data sources an important direction in the development of any database management system.

This paper is focused on the improvement of ClickHouse DBMS integration with external databases. ClickHouse already has an architecture that simplifies integration with foreign data, but there are many features not implemented.

In this paper, ClickHouse architecture and other DBMS' solutions are studied and described. Based on the knowledge gained, I have added integration features to ClickHouse source code.

As a result of the work, the ability to write data to the ODBC table function is supported. Readable ClickHouse tables based on existing MongoDB collections are implemented. The implemented functionality will be used in the ClickHouse project.

Keywords: DBMS, data integration, C++, external data sources, ODBC, MongoDB

Содержание

Введение

Система управления базами данных (СУБД) - это совокупность программных и лингвистических средств, которая позволяет пользователям создавать и обновлять базы данных, контролирует к ним доступ. IT-компании во всему миру работают с данными при помощи СУБД, делая к ним множество запросов каждый день.

Многие компании разрабатывают несколько сервисов для различных целей, либо взаимодействуют с различными внешними источниками данных. Из-за этого возникает необходимость обращаться к совершенно различным базам данных, а возможность удобной и эффективной интеграции с внешними базами становится важным преимуществом любой СУБД.

ClickHouse – это система управления базами данных с открытым исходным кодом, созданная в компании Yandex [1]. ClickHouse используется многими сервисами Яндекса и внешними пользователями. Разработчики проекта постоянно работают над интеграцией ClickHouse с внешними источниками. Хотя многие цели, такие как поддержка различных источников внешних словарей и интеграция с MySQL, уже достигнуты, перед ClickHouse еще стоит задача расширения круга внешних источников, с которыми необходимо создать возможность интеграции.

1 Описание задач

В ходе выпускной квалификационной работы решалась задача улучшения интеграции ClickHouse с внешними базами данных, включающая в себя разработку решений по нескольким направлениям:

1) Поддержка записи в табличную функцию ODBC
2) Движок для чтения из таблиц MongoDB

ODBC (Open DataBase Connectivity) – это часто применяющийся API для доступа к внешним базам данных, позволяющий обращаться к ним с помощью SQL-запросов. Для использования ODBC на стороне внешней базы данных должен существовать ODBC-драйвер, в системе должен быть установлен ODBC-менеджер, соединяющийся с драйвером, в конфигурации менеджера можно хранить информацию, необходимую для подключения к внешним базам. На стороне приложения (в данном случае им выступает ClickHouse) необходимо разработать модуль, способный необходимым образом общаться с менеджером. В ClickHouse подобный модуль уже существовал, но был пригоден только для чтения данных. Одной из задач, выполняемых в ходе ВКР, стала поддержка записи во внешнюю базу при помощи ODBC. Для ее решения нужно было научиться подготавливать пользовательские данные для передачи, а затем передавать их в нужную базу.

Вторым направлением в улучшении интеграции ClickHouse с внешними базами стала реализация движка таблиц для чтения из MongoDB. MongoDB – популярная система управления базами данных, особенностью которой является использование JSON-подобных документов в качестве записей. Обращение к базе данных происходит с помощью нетипичного NoSQL синтаксиса, но в нашем случае общение с удаленным сервером производится через Poco API для C++. Движок таблицы в ClickHouse – ее тип, свойство, которым обладает любая пользовательская таблица. Тип задается при создании таблицы и определяет место и способ хранения данных, возможность многопоточного доступа и параметры репликации. Несколько движков используются для интеграции данных (ODBC, JDBC, MySQL, Kafka). Требовалось реализовать инте-

грационный движок для Mongo, позволяющий обращаться к коллекциям (набор записей, аналог таблицы) MongoDB для чтения, как к таблицам ClickHouse.

Важной частью работы является написание интеграционных тестов для реализованного функционала, а также документации, поясняющей его пользовательский интерфейс.

2 Анализ существующих решений

Для лучшего понимания темы я изучал различные подходы к интеграции между СУБД. Как правило, реализация, требуемая от разработчика зависит от архитектурных решений, используемых в СУБД, а те, в свою очередь, обусловлены целями их создания, нуждами пользователей и решаемыми задачами.

2.1 PrestoDB

PrestoDB - это механизм запросов с открытым исходным кодом, созданный для Facebook [2]. Создание SQL-интерфейса для популярных NoSQL-источников было одной из первоначальных целей разработчиков Presto. PrestoDB интегрируется с YARN, Cassandra, Kafka и другими NoSQL-источникам. Кроме того, он может запрашивать данные из реляционных баз и имеет драйвер JDBC. Presto изначально разрабатывался для обработки больших данных. Поэтому в нем реализовано множество оптимизаций чтения данных. Примером такой оптимизации может служить работа с форматами, хранящими данные по столбцам (Parquet, RCFile, ORC). Такие файлы преобразуются в страницы, состоящие из блоков. Один блок соответствует столбцу и занимает последовательные ячейки памяти. Блоки можно помечать как lazy. Такие блоки будут прочитаны только при фактическом доступе к ним, что позволяет сэкономить время на чтении неиспользуемых данных.

Важным архитектурным решением является наличие Connector'ов, позволяющих взаимодействовать с внешними базами данных с помощью одного API (Connector API). PrestoDB имеет connector'ы для более, чем 20 внешних источников. Таким образом, высокая адаптивность PrestoDB достигнута при помощи архитектурных решений.

2.2 PostgreSQL

Удобная интеграция с внешними базами данных важна не только для проектов, ставящих интеграцию одной из главных целей. Разра-

ботчики всех популярных SQL- и NoSQL-баз данных стараются уделять время интеграции с другими проектами.

PostgreSQL использует так называемые Foreign data wrappers (FDW) [3] для подключения других баз данных. Реализовано множество FDW для внешних источников. Из Postgres можно подключиться к другим реляционным СУБД, таким как Oracle, SQLite и MySQL, базам данных NoSQL, например, MongoDB, системам для обработки больших данных, таким как Hive. Postgres поддерживает множество форматов данных (JSON, CSV, Parquet и др.) с помощью FDW. FDW также созданы для обращения к веб-API Google, Twitter, Telegram и т. д. Стоит отметить, что большинство FDW официально не поддерживаются группой разработки PostgreSQL и создаются пользователями.

2.3 Database Federation

Описанные выше методы интеграции создают пользовательские интерфейсы, позволяющие подключаться к внешним источникам данных. Однако существуют механизмы, помогающие пользователю делать запросы к нескольким различным базам данных через один интерфейс. Таким механизмом является Database Federation. Он описывается в статье “Data integration through database federation” [4] от разработчиков IBM.

Database Federation добавляет дополнительный слой между запросом пользователя и различными источниками данных. Внутри этого слоя источники данных линкуются друг с другом. Пользователь делает SQL-запросы к виртуальной базе данных, как к обычной базе данных, поэтому дополнительный слой обеспечивает прозрачность системы для пользователя.

В DB2 этот слой состоит из обработчика запросов, менеджера данных, управляющего локальным хранилищем, и механизмов расширения: определяемых пользователем функций и оболочек для внешних источников данных.

DB2 поддерживает несколько способов объединения внешних источников. В некоторых случаях на дополнительном слое переписывается

для каждой базы пользовательский запрос, в некоторых - объединяются в одну базу данные, полученные из нескольких источников. В более сложных случаях используются обе техники одновременно. Эти махинации скрыты от пользователя, но очевидно, что они негативно влияют на производительность системы.

2.4 ClickHouse

Несмотря на то, что изначально ClickHouse был создан для хранения и анализа больших данных для Яндекс.Метрики, сейчас эта СУБД старается удовлетворить потребности множества разных пользователей по всему миру. Это значит, что для интеграции с другими источниками данных ClickHouse необходимо гибкое решение, которое охватывает как можно больше популярных источников и не является слишком специфичным, как DatabaseFederation.

Архитектура ClickHouse, созданная для интеграции, похожа на существующую в PrestoDB и PostgreSQL. Важно, что весь код, отвечающий за взаимодействие с внешними БД, поддерживается и проверяется командой ClickHouse. Для удобной интеграции других СУБД с ClickHouse, официально подерживается ODBC-драйвер [5]. Более подробное описание архитектуры есть в разделе ниже.

3 Обзор архитектуры ClickHouse

Как было сказано выше, действия, необходимые для улучшения интеграции ClickHouse, продиктованы, в первую очередь, архитектурой ClickHouse. Поэтому для понимания поставленных задач, выбора их решений и восприятия описания решений необходимо ознакомиться с такими элементами архитектуры ClickHouse, как движки таблиц, табличные функции, потоки блоков.

Как отмечалось выше, в ClickHouse для описания таблиц, в том числе внешних, используются движки таблиц, которым соответствуют имплементации интерфейса IStorage.

Своего рода расширением IStorage является табличная функция (ITableFunction). Табличная функция позволяет создавать таблицу на соответствующем ей движке, но может быть вызвана не только в запросе CREATE TABLE. Как правило, интерфейс для вызова табличной функции совпадает с интерфейсов объявления типа таблицы, но при запросе к табличной функции необязательно указывать схему. Для пользователя табличная функция упрощает запросы к таблицам, а также является одним из методов создания таблиц. Но отсутствие схемы при вызове является сложностью реализации табличной функции, так как схему необходимо узнать перед созданием IStorage. Поэтому для MongoDB, не имеющей схемы, требовалась только поддержка движка, но не табличной функции.

Для передачи данных в ClickHouse используются интерфейсы IBlockInputStream и IBlockOutputStream. Их наследники переопределяют способ чтения или записи блока данных. Наследники IStorage, как правило, в методах read и write создают Stream'ы для передачи произвольных данных в одном из поддерживаемых форматов с учетом специфики того или иного источника. Например, StorageURL отвечает за передачу блоков данных по HTTP-протоколу. Для записи и чтения из внешнего источника необходимы соответственно IBlockOutputStream и IBlockInputStream для этого источника, которые будут вызываться в его IStorage.

4 Проделанная работа

4.1 Выбор решения

Благодаря уже существующей архитектуре ClickHouse, выбор решения не был сложным. Для записи в уже поддерживаемый источник необходима была реализация IBlockOutputStream для него. В IStorage для этого требуется переопределение функции write. Функция должна подготавливать данные для передачи через IBlockOutputStream, передавать их и обрабатывать результат передачи.

Создание нового движка подразумевает имплементацию еще одного наследника IStorage, который должен уметь конструироваться из аргументов, переданных пользователем. В случае с MongoDB требовалось реализовать и функцию read, ответственную за чтение данных.

4.2 Задача записи в табличную функцию ODBC

Из рассуждений выше ясно, что нужно было реализовать функцию write для StorageXDBC (IStorage, отвечающий за движки ODBC и JDBC). Для ее решения я вносил изменения в код clickhouse-server и clickhouse-odbc-bridge. clickhouse-server – сервер СУБД ClickHouse, на него поступает пользовательский запрос (движки таблиц реализованы в его коде), и он передает этот запрос при помощи ODBC-менеджера во внешнюю базу через clickhouse-odbc-bridge.

Наличие дополнительного звена (odbc-bridge, является HTTP-сервером) обусловлено тем, что пользователь может попытаться произвести подключение к любому ODBC-драйверу, и корректная работа на стороне внешнего источника данных не может быть гарантирована. Таким образом, динамическая линковка с неизвестным драйвером через менеджер может привести к падению сервера, что недопустимо. Но в архитектуре, созданной в ClickHouse, при такой проблеме пострадает только odbc-bridge, который может быть легко перезапущен сервером. Наличие дополнительного звена усложняет задачу, так как требует реализации передачи данных между звеньями.

Реализация записи с помощью ODBC состоит из следующих этапов:

1) Подготовка HTTP-запроса к ODBC-bridge, передача блоков данных
2) Получение данных на стороне ODBC-bridge.
3) Сериализация блоков данных для передачи в менеджер ODBC.
4) Формирование запроса к ODBC-менеджеру.
5) Обработка возможных ошибок, ответ серверу ClickHouse.

Первый этап включает в себя конструирование url запроса (туда входит информация о внешней базе данных, таблице, ее схема, название формата блоков), а также передача блоков по собранному url при помощи StorageURL. На втором этапе odbc-bridge понимает, что ему пришел запрос на запись, проводится аутентификация во внешней базе, парсинг url-параметров.

Блоки данных поступают в ODBCBlockOutputStream (реализованный на стороне odbc-bridge) в HTTP POST-запросе при помощи Chunked Transfer Encoding [6]. На третьем этапе, после десериализации блоков механизмами ClickHouse, происходит итерация по строкам блока и превращение данных в формат, пригодный для передачи по сети (класс Poco::Dynamic::Var, отвечающий за переменную одного из поддерживаемых Poco типов). На четвертом этапе для каждой строки формируется INSERT-запрос с количеством аргументов, равным количеству переданных столбцов, который направляется в ODBC-менеджер средствами API в библиотеке Poco. Таким образом, ODBCBlockOutputStream обеспечиваем передачу во внешнюю базу произвольного количества значений произвольных типов. Успешность этой передачи контроллируется в функции write на стороне сервера ClickHouse. Сервер получает HTTP-ответ от odbc-bridge и выводит пользователю информацию о завершении операции или сообщение об ошибке.

Для внешнего пользователя, уже знакомого с табличной функцией ODBC и синтаксисом SQL, поддержка записи не усложняет работу с ClickHouse. Для записи используется уже существующая функция

```
odbc(connection_string, database_name, table_name)
```

SQL-запрос для вставки во внешнюю базу будет выглядеть как

```
INSERT INTO odbc(...) VALUES (...)
```

4.3 Задача разработки движка таблиц MongoDB

Для реализации табличного движка для чтения из MongoDB необходимо было имплементировать наследника IStorage для этой СУБД и метод для чтения из нее.

Был реализован конструктор StorageMongoDB, позволяющий создать таблицу по адресу внешней базы, названию базы данных, коллекции и данным для аутентификации. Таким образом, для создания таблицы на движке Mongo пользователю необходимо отправить подобный SQL-запрос:

```
CREATE TABLE table_name(
    ‘name‘ String, ‘num‘ UInt32   // описание схемы
)
ENGINE=Mongo(
    ’host:port’,
    ’db_name’,
    ’collection_name’,
    ’username’,
    ’password’
)
```

После этого к таблице table_name можно обращаться в режиме только для чтения. Перед чтением данных на уровне реализации выполняются следующие этапы:

1) Аутентификация во внешней базе данных
2) Подготовка запроса к MongoDBBlockInputStream
3) Чтение данных при помощи MongoDBBlockInputStream
4) Получения ответа от MongoDBBlockInputStream

Обращения к внешней базе MongoDB производятся, как и в прошлом случае, при помощи библиотеки Poco. Использовались методы аутентификации и чтения данных, уже работающие в ClickHouse с MongoDB как источником внешних словарей.

При подготовке запроса в MongoDBBlockInputStream конструируется образец блока данных, основанный на пользовательском запросе и заданной при создании таблицы схеме.

Отсутсвие понятия схемы в MongoDB ограничивает оптимизацию запроса в эту СУБД, но MongoDBBlockInputStream позволяет получить из внешней базы требуемые данные, обрабатывая возникающие ошибки.

4.4 Тестирование

Для проверки корректности реализаций использовались интеграционные тесты. Такие тесты запускают внешние СУБД при помощи Docker, создают внешние таблицы в контейнерах, после чего запросами к серверу ClickHouse проверяют, корректно ли работает интеграция с созданными таблицами. Тесты реализованы на языке Python.

В случае с ODBC внешним источником является сам ClickHouse. Тест проверяет как интеграцию с внешней ODBC-базой, так и odbc-driver ClickHouse.

Мной были написаны запросы, проверяющие работу реализованного функционала. Для отправки запросов и проверки корректности работы и результатов использовался уже существующий и описанный выше механизм.

Заключение

В результате работы были изучены различные механизмы интеграции между базами данных. Для выбора решения поставленных задач я подробно ознакомился с архитектурой ClickHouse и существующими решениями внутри этой СУБД. После этого мной было получено решение двух практических задач.

Был реализован механизм записи во внешние базы данных при помощи ODBC, позволяющий пользователю обращаться к ним INSERT-запросами с помощью уже известного интерфейса.

Также реализован интерфейс, позволяющий создать в ClickHouse таблицу на основе существующей коллекции MongoDB и обращаться к ней запросами на чтение.

После реализации функциональность была отправлена в репозиторий ClickHouse в формате pull request. Готовятся техническая документация для пользователей и дополнительные тесты. Решение задачи о записи в ODBC прошло code review, было доработано командой ClickHouse и добавлено в репозиторий.

Ссылки на код решений:

ODBC: https://github.com/ClickHouse/ClickHouse/pull/10901

MongoDB: https://github.com/ClickHouse/ClickHouse/pull/10931

Список использованных источников

1. ClickHouse documentation. — Yandex, LLC., 2016-2020.

2. *R. Sethi M. Traverso, D. Sundstrom et al.* Presto: SQL on Everything / D. Sundstrom et al. R. Sethi, M. Traverso. — Facebook, Inc., 2019.

3. *Hanada, S.* PostgreSQL Documentation. F.31. postgresfdw / S. Hanada. — The PostgreSQL Global Development Group, 1998-2020.

4. *L. Haas E. Lin, M. Roth.* Data integration through database federation / M. Roth L. Haas, E. Lin. — IBM Systems Journal, Vol. 41, 2002.

5. ODBC Driver for ClickHouse. — https://github.com/ClickHouse/clickhouse-odbc, 2020.

6. RFC 2616, 3.6.1, Chunked Transfer Coding. — Network Working Group, 1999.