

**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте

на тему Embedded ClickHouse as a Python module

Выполнил:

студент группы БПМИ209

19.05.2022

Дата



Подпись

Лаврентий Викторович Гришин

И.О. Фамилия

Принял:

руководитель проекта

Алексей Николаевич Миловидов

Имя, Отчество, Фамилия

технический директор

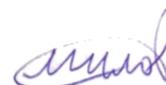
Должность

ClickHouse.inc

Место работы

Дата 19.05.2022

_____ (по 10-тибалльной шкале)



Подпись

Москва 2022

Реферат

Цель моего проекта - создание функций-оберток для столбцовой СУБД ClickHouse с помощью библиотеки `rubind11`, которые позволяют скомпилировать python-модуль и использовать функции ClickHouse в проектах на Python, а также в мою задачу входит написание необходимого кода для компиляции.

В процессе работы были изучены необходимые части исходного кода ClickHouse, документация данной СУБД и документация библиотеки `rubind11`, исходный код СУБД DuckDB, основы работы с системой сборки проектов CMake.

Результатом работы является директория, состоящая из двух файлов: `.cpp` файл с двумя функциями-обертками для функции ClickHouse `executeQuery`, код для создания самого python-модуля, и CMake файл позволяющий скомпилировать данный код в библиотеку Python.

Содержание

Реферат	Error! Bookmark not defined.
Определения и термины	4
Введение	5
Обзор аналогов	6
Основная часть	7
Описание функции query	7
Описание функции create_insert	7
Вспомогательные функции	8
Создание библиотеки	9
CMakeList.txt	9
Расположение файла библиотеки	10
Заключение	10
Список источников	12

Определения и термины

- СУБД – система управления базами данных
- ClickHouse – столбцовая аналитическая СУБД
- pybind11 – библиотека, позволяющая создавать Python-библиотеки на языке C++
- DuckDB – СУБД, написанная на C++ и предоставляющая python API, также реализованного с помощью pybind11
- numpy – библиотека для работы с многомерными массивами
- SQL-запросы - это наборы команд для работы с реляционными (табличными) базами данных
- CMake — система сборки проектов.

Введение

На сегодняшний день существует огромное количество сервисов и приложений которые работают с большими объемами данных, зачастую, их необходимо не только хранить, но и анализировать. ClickHouse - аналитическая столбцовая СУБД предоставляющая широкий функционал и хорошо зарекомендовавшая себя среди различных IT-компаний по всему миру. На данный момент у ClickHouse нет интерфейса в Python, и цель моего проекта - положить начало его созданию. Python-модуль предоставляющий функционал данной СУБД сильно облегчил бы задачу для разработчиков на этом популярном и востребованном сегодня языке. Зачастую, именно Python используют для анализа больших объемов данных, так как он имеет подходящие для этого библиотеки - например numpy.

Актуальность проблемы

Python наиболее популярный и подходящий язык для анализа больших данных и работы с ними. Популярность обусловлена несколькими факторами: язык имеет нужные для этого библиотеки - например numpy, имеет лаконичный синтаксис, обладает хорошей читаемостью кода и более прост чем C или C++. Подавляющее большинство специалистов в области анализа данных используют именно Python.

Цель и задачи проекта

Цель данного проекта – создать базовый Python-интерфейс для ClickHouse, позволяющий выполнять простые SQL-запросы вида SELECT, CREATE TABLE и INSERT INTO, также необходимо уметь взаимодействовать с numpy-массивами и python-словарями. Основная работа предстояла с функцией ClickHouse executeQuery, которая как раз и предназначена для обработки и выполнения запросов.

Для создания Python-модуля было решено работать с библиотекой pybind11. Данная библиотека предоставляет функционал, позволяющий осуществить трансляцию типов из Python в C++ и наоборот, а также создать сам python-модуль.

Создание библиотеки подразумевает создание привязки функций Python к функциям, написанным на C++. Типы Python можно использовать в C++ коде с помощью специальных оберток собственного пространства имен библиотеки – namespace pybind11.

Обзор аналогов

Аналогичная задача была решена командой СУБД DuckDB, которая предоставляет полноценный python API. Сама идея была позаимствована именно у них, как и некоторые детали реализации.

DuckDB обладает директорией с необходимыми функциями-обертками и другими инструментами позволяющими скомпилировать python-модуль. Основное отличие подхода разработчиков DuckDB от решения ClickHouse в том, что у нас нет необходимости создавать Python-объект connection - который предоставляет соединение с базой данных и через который нужно осуществлять с ней взаимодействие. При использовании нашего модуля достаточно просто вызвать нужную функцию и все.

Основная часть

В первую очередь, мне было необходимо ознакомиться с архитектурой ClickHouse, изучить документацию `pybind11`, посмотреть как решена аналогичная задача в DuckDB.

Выполнить поставленную задачу было решено следующим образом – написать две функции-обертки: `query` - функция для обработки SELECT - запросов и `create_insert` – функция для обработки CREATE TABLE - запросов и INSERT INTO - запросов, более того, `create_insert` должна уметь работать с python-словарями из numpy-массивов.

Замечание: пространство имен `py` эквивалентно пространству имен `pybind11` тк в коде прописано

```
namespace py = pybind11;
```

для удобства.

Описание функции query

Функция `query` обладает следующей сигнатурой:

```
py::dict query(const std::string & query)
```

Где принимаемое значение – это строка самого запроса из кода в python, которая преобразуется в `std::string` автоматически с помощью библиотеки `pybind11` при вызове функции, `py::dict` – это результат работы данной функции, а именно – python-словарь состоящий из питоновских списков (`list`) со строками (если результат запроса содержал в качестве ответа строки) или из numpy массивов, если в результате были получены наборы численных значений.

Словарь имеет структуру `{ {имя колонки: np.array(значения)}, ... }`

то есть ключ - имя колонки, значение - лист или `np.array`.

Функция работает следующим образом:

1. создает глобальный контекст запроса если он еще не создан
создание контекста частично позаимствовано из исходного кода Clickhouse local
2. создает локальный контекст и контекст запроса – `QueryContex`
3. вызывает функцию `executeQuery` и получает объект `BlockIO`
4. использует `pipeline.pulling()` или `pipeline.completed()` (создавая соответствующий `pipeleneExecutor`) для получения результатов данных
5. конвертирует их в подходящий тип (перебираем возможные типы для специально для этого написанной функции `tryConvert`, это нужно потому что мы заранее не знаем, какого типа будет ответ на запрос) и добавляем в словарь, который мы затем возвращаем пользователю.

Описание функции create_insert

Функция `create_insert` обладает следующей сигнатурой:

```
void create_insert(std::string table_name, const py::dict & data)
```

`create_insert` принимает на вход имя таблицы и python-словарь с данными.

Далее, создает таблицу (если таковая еще не создана) с именем `table_name`, и вставляет в нее данные из словаря с объектами из Python, например списками или numpy-массивами.

Как известно, ClickHouse использует для хранения данных колонки (Column). Функция ожидает данные в следующем формате: `{ {имя колонки : pr.array([...])}, ... }`, при этом имя колонки должно быть строкой, а данные в списке или numpy массиве для каждого ключа (имени колонки) должны быть одного типа.

Функция `create_insert` работает следующим образом:

1. проверяет, существует ли таблица с именем `table_name`
2. создает глобальный контекст если он еще не создан
3. создает локальный контекст и контекст запроса
4. если таблица не существует, то создает ее с помощью функции `executeQuery`
5. вызывает функцию `executeQuery` для получения BlockIO и получения pipeline'a
6. создает `PushingPipelineExecutor` для обработки запроса вида INSERT INTO
7. начинает обрабатывать словарь с данными итерируясь по ключам
8. создает колонку с именем равным ключу словаря
9. идет по передат значение полученное по ключу из словаря в специальную функцию `tryConverToColumn`, которая перебирает возможные типы, в которые элементы этого списка или numpy-массива могут быть сконвертированы
10. функция `tryConverToColumn` преобразует переданный объект в список и итерируется по нему, добавляя значения в колонку
11. добавляет эту колонку в таблицу с именем `table_name`

Вспомогательные функции

Для удобства и упрощения кода были созданы еще некоторые функции и класс `ClickHouseInstanceHolder`.

Начнем с этого класса.

Он необходим для создания глобального контекста, причем он может быть создан только один раз.

`ClickHouseInstanceHolder` содержит конструктор, который и создает глобальный контекст, устанавливает `currentDatabase`, устанавливает для него `userConfig`, включает табличные функции и так далее.

Также в этом классе присутствует поле

```
ContextHolder holder;
```

– это и есть глобальный контекст.

Функция `setupUser` отвечает за настройку информации о пользователе (пользователь создается с дефолтной информацией как и в ClickHouse local)

Функция `createMemoryDatabaseIfNotExist` создает базу данных если такая еще не существует и возвращает указатель на нее.

Функции `tryConvert` и `tryConvertToColumn` пытаются преобразовать значения к определенному типу, и если это получается, они конвертируют все данные, которые им передали и функция либо формирует ответ для пользователя из этих данных либо вставляет их в колонку (соответственно для каждой функции). Если приведение типов удалось, то функции возвращают значение `true` и программа идет дальше, в противном случае, возвращается значение `false` и происходит попытка приведения к другому типу.

Создание библиотеки

Код для создания самой библиотеки выглядит довольно просто – нужно всего лишь указать имена функций, описания и ссылки на имплементацию самих функций:

```
PYBIND11_MODULE(clickhouse_py, m)
{
    m.doc() = "pybind11 clickhouse python plugin";
    m.def("query", &query, "execute SELECT query");
    m.def("create_insert", &create_insert, "create a table if it does
not exist, or move data to a table");
}
```

`m` – это сам модуль - `clickhouse_py`.

`m.doc` – описание модуля

`m.def` – добавление в модуль функции.

CMakeList.txt

Также в директории моего проекта присутствует `CMakeList.txt`, он нужен для сборки проекта и компиляции самой библиотеки, в нем устанавливаются необходимые зависимости, я приведу часть файла:

```
add_subdirectory(pybind11)
find_package(PythonLibs REQUIRED)
include_directories(${PYTHON_INCLUDE_DIRS})

pybind11_add_module(clickhouse_py mylib.cpp)
target_link_libraries(clickhouse_py PRIVATE readpassphrase dbms
clickhouse_storages_system clickhouse_functions
clickhouse_aggregate_functions clickhouse_table_functions
${PYTHON_LIBRARIES})
```

здесь добавляется сам модуль, который создается из файла `mylib.cpp` и устанавливаются зависимости с другими частями `ClickHouse`, которые необходимы для работы библиотеки.

Расположение файла библиотеки

После компиляции всего проекта, в папке `/build/python_module/` появится файл `clickhouse_py.cpython-310-x86_64-linux-gnu.so` – это и есть итоговая библиотека. Для подключения библиотеки в python файл необходимо добавить строку

```
import clickhouse_py
```

Заключение

В результате были написаны функции `create_insert` и `query`, которые позволяют обрабатывать запросы из Python и работать с numpy массивами, списками и словарями на стороне ClickHouse. Создана директория, которая при сборке создает файл python-модуля, содержащего эти функции.

В перспективе, можно продолжить увеличивать функционал данной библиотеки и заниматься улучшением производительности, также сделать совместимость с pandas.

СПИСОК ИСТОЧНИКОВ

1. ClickHouse - <https://clickhouse.com/docs/en/>
2. ClickHouse LocalConnection - <https://github.com/ClickHouse/ClickHouse/blob/8b41a5835202e28037c8e9232ac56bc22864594c/src/Client/LocalConnection.cpp>
3. Pybind11 - <https://pybind11.readthedocs.io/en/stable/>
4. DuckDB - <https://duckdb.org/docs/>
5. DuckDB Python module - <https://github.com/duckdb/duckdb/tree/0d3368c18cb77aaa4c581ff2c71d7b45d88be7d9/tools/pythonpkg>
6. Numpy - <https://numpy.org/doc/stable/>