

Оглавление

Введение	3
Описание взаимодействия	4
Синтаксис запросов между разными СУБД	5
Классы СУБД.....	7
Парсер запросов.....	8
Заключение.....	9
Список литературы.....	9

Введение

В настоящее время наблюдается продолжающийся рост систем, поддерживающих огромный объем реляционных и нереляционных форм данных. Примерами моделей данных, которые поддерживают многомодельные базы данных, являются документные, графические, реляционные модели и модели ключ-значение [1]. Наличие единой системы данных для управления, как хорошо структурированными данными, так и данными NoSQL выгодно пользователям, поскольку для каждой конкретной задачи существуют более предпочтительные варианты хранения данных в зависимости от типа СУБД, таким образом, предоставление разных структур хранения данных в одной системе позволяет сделать информацию более доступной и понятной для пользователей. Такая система также улучшает визуализацию и понимание данных.

Запрос на систему для взаимодействия нескольких баз данных между собой выражен в следующей статье [2]. Опосредовано нужна в данной системе упоминается в самых различных сферах, например, при исследовании биологии [3, 4] или политологии [5]. Попытка создания аналогичной системы уже были, однако она была представлена для специализированной сферы, что представлено в статье [6], также уже была попытка создать систему связывающую различные БД через JSON запросы [7], а рассматриваемая в данной работе система также является универсальной, что выражается в поддержке нескольких СУБД и возможности подключать множество баз данных и брать информацию из нескольких источников одновременно, однако работает по другому принципу. В описываемой системе не производится сбор данных из разных БД, а идёт обращение непосредственно к СУБД.

Система разработана на языке Python с использованием фреймворка Flask.

Описание взаимодействия

Всего в системе можно выделить 3 части - ввод запроса пользователем, парсер запроса для каждой СУБД, исполнение запроса каждой СУБД. Визуально их взаимодействие показано на рис. 1, текстовое описание представлено далее.

Первая часть - ввод запроса пользователем. Пользователь в браузере вводит запрос на получение нужных ему данных в соответствии с специальным синтаксисом, строение которого описано далее.

Вторая часть - парсер запросов. В этом элементе системы запрос пользователя разбивается на части для каждой БД. Эти части запроса в нужной для корректного исполнения последовательности подаются в соответствующие классы определённой СУБД, после чего возвращают результат, который уже обрабатывается и выводится пользователю или добавляется в запрос для следующего класса СУБД, если это не конечный результат.

Третья часть, которую можно выделить, это - классы СУБД. Каждый такой класс представляет собой возможность взаимодействия с СУБД для выполнения запроса, а его объекты представляют каждую БД в этой СУБД. Каждый объект класса должен преобразовывать часть запроса, пришедшего к нему от парсера, в запрос на язык, который является командой для СУБД, к классу которой он относится. После преобразования он обращается с этой командой к БД, к которой он относится, и возвращает результат в парсер.

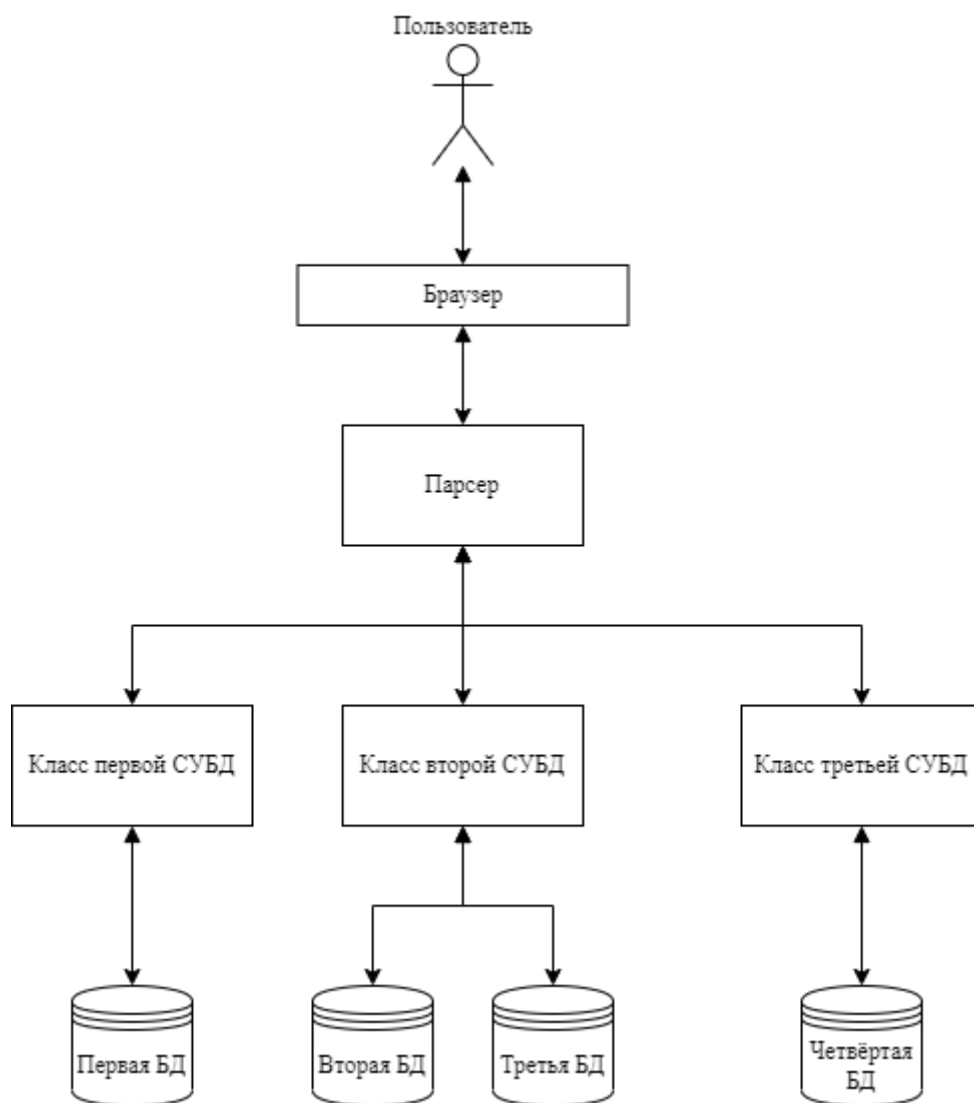


Рисунок 1. Взаимодействие между элементами

Синтаксис запросов между разными СУБД

Чтобы была возможность делить запросы для исполнения на разных СУБД необходимо сделать свой синтаксис запросов. Далее представлено описание синтаксис основных запросов.

Запрос будет записываться одной строкой, в которой будут писаться название БД, путь к данным из БД и операторы с разделителями в виде точек.

Пример запроса:

```
dbcity.read(dbcity.building.address).where(dbpeople.read(dbpeople.personal_data.name).where(dbpeople.personal_data.id=1)=dbcity.building.address_owner)
```

Сначала пишется название БД. После этого через разделитель - точку, пишется оператор, применяемый к указанной БД.

В системе будут представлены следующие операторы:

- `x.y.z`, где `x` – название БД, `y` – название сущности, являющийся аналогом таблицы из реляционных БД, из ранее обозначенной базы данных, `z` – название сущности, являющийся аналогом столбцов из реляционных БД, из ранее обозначенной таблицы. Таким образом из БД можно получить необходимые данные.

- `where` – оператор, в котором в скобках описывается фильтр, по которому выбираются данные из БД. В условии используются оператор доступа к данным и операторы сравнения. Сравнимыми элементами могут являться данные из БД или константы. Условия могут вложенными.

- `create` – вставка данных в БД. У оператора в скобках сначала через запятую указываются операторы доступа к данным, которые означают новые поля, после чего в конце через запятую указываются вставляемые данные в виде массива списков. Например, `dbcity.create(dbcity.building.address, dbcity.building.owner, [[“Wall Street”, “Stan Smith”], [“Broadway”, “John Doe”]])`.

- `read` – чтение данных из БД. У оператора в скобках указываются операторы доступа к данным, которые должны быть прочитаны. Например, `dbcity.read(dbcity.building.address)`.

- `update` – изменение данных в БД. У оператора в скобках сначала через запятую указываются операторы доступа к данным, которые означают поля, в которых будут производиться изменения, после чего в конце через запятую указываются вставляемые данные в виде списка. Например, `dbcity.update(dbcity.building.address, dbcity.building.owner, [“Broadway”, “John Doe”])`

- `delete` – удаление данных из БД. После данного оператора указывается оператор `where`. Например, `dbcity.delete.where(dbcity.building.owner=“John Doe”)`.

Классы СУБД

В системе для каждой СУБД создаётся собственный класс. Для каждой БД, которую подключает пользователь создаётся собственный объект соответствующего класса СУБД. Все классы наследуются от шаблона, пример для Neo4j, Cassandra и MongoDB представлен на рис. 2.

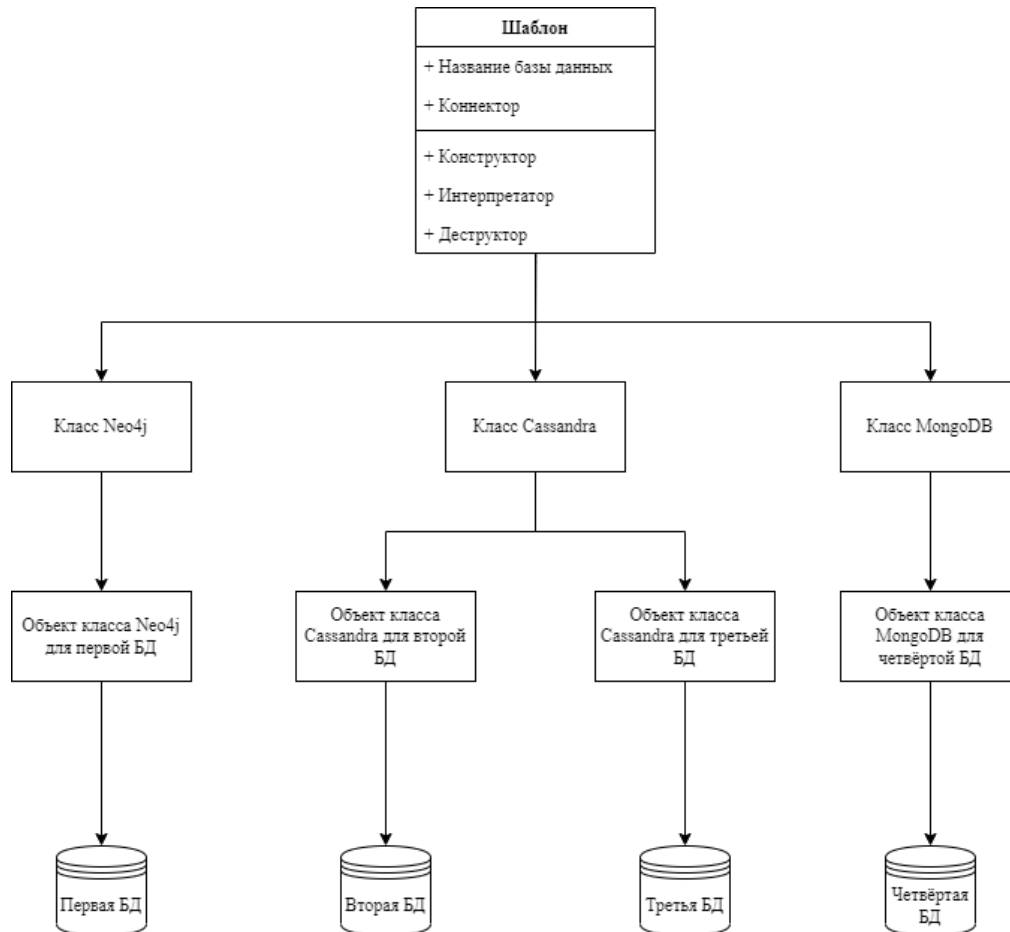


Рисунок 2. Схема работы классов СУБД

Сам шаблон должен иметь следующую структуру полей:

- Коннектор - переменная, которая содержит подключение к БД, которое впоследствии будет использоваться в методах для взаимодействия с данными.
- Название базы - название, которое необходимо для различия между разными БД одной СУБД.

Так же шаблон должен содержать следующие методы:

- Конструктор. Метод, которому на вход подаётся - адрес базы данных; порт базы данных; логин для авторизации; пароль для авторизации; название базы данных. Данный метод создаёт и хранит подключение и название базы данных.
- Интерпретатор. Метод, которому на вход подаётся - запрос на синтаксисе, описанном в предыдущей главе. Данный метод преобразует полученный запрос в формат, нужный для конкретной СУБД и исполняет его. В конце возвращает массив кортежей с полученным результатом.
- Деструктор. Метод, который закрывает подключение для текущей БД.

Парсер запросов

В описываемой в данной статье системе запрос от пользователя необходимо разделить на подзапросы. После этого разбитые подзапросы конвертируются в запросы, которые могут воспринимать СУБД, для которых эти запросы предназначены и исполняются.

Сначала запрос читается слева направо и разбивается на подзапросы. Далее подзапросы конвертируются в читаемый для СУБД вид и исполняются справа налево с передачей результата запроса. Данный процесс подробно изображен на рисунке 3.

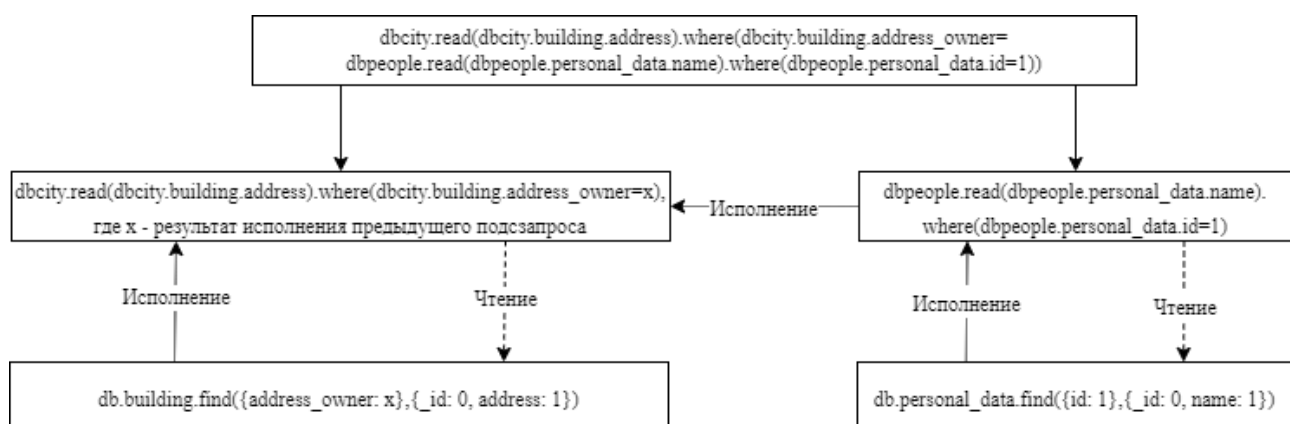


Рисунок 3. Деление основного запроса на подзапросы для каждой БД

Конвертацию подзапросов осуществляет общий для всех классов БД метод Интерпретатор. Рассмотрим работу одного из них, а именно интерпретатора класса MongoDB.

Был выделен следующий подзапрос:

```
dbpeople.read(dbpeople.personal_data.name).where(dbpeople.personal_data.id=1)
```

Сначала идёт подключение к базе данных dbpeople. Парсер начинает выполнять с более меньшего подзапроса:

```
(dbpeople.personal_data.name).where(dbpeople.personal_data.id=1),
```

 где dbpeople – БД в СУБД MongoDB.

Часть запроса dbpeople.personal_data.name преобразуется в {_id: 0, name: 1}, where(dbpeople.personal_data.id=1) преобразуется в {_id: 1}.

Далее конвертируется часть запроса dbpeople.read() и выполняется в виде следующего запроса:

```
db.personal_data.find({_id: 1},{_id: 0, name: 1}).
```

Заключение

В данной работе было проведено теоретическое обоснование возможности и необходимости создания системы для управления данными, представленными в нереляционных БД. Данная система позволит пользователю получать информацию из разных БД доступней и представлять её в более понятном виде. Также были описаны основные элементы системы и их взаимодействие. Более подробно были рассмотрены классы, через которые происходит взаимодействие системы с БД. Для того, чтобы была возможность одним запросом обращаться к нескольким БД был разработан синтаксис для этих запросов и парсер, который разделяет запрос пользователя на подзапросы. Данная система может полезна для экспертов в области электронной коммерции и аналитиков данных.

Список литературы

1. Бердыбаев Р.Ш., Гнатюк С.О., Тынимбаев С., Азаров И.С. Анализ современных баз данных для использования в системах Siem // Вестник Алматинского университета энергетики и связи, Т. 54, №3, С. 33-47, 2021. DOI 10.51775/1999-9801_2021_54_3_33.
2. Blinova O.V., Pankratova E. V., Farkhadov M.P. Principles of construction and analysis of architectures for modern scientific information systems // 7th International Scientific Conference. 2023. P. 111-113. DOI 10.25728/icct.2024.032.

3. Schäfer R.A., Rabsch D., Scholz G.E., Stadler P.F., Hess W.R., Backofen R., Fallmann J., Voß B. RNA interaction format: a general data format for RNA interactions // Bioinformatics, Vol. 39, №11, P. 1-3, 2023. DOI 10.1093/bioinformatics/btad665.
4. Martha V.S, Liu ZH., Guo Li., Su Zh., Ye Ya., Fang H., Ding D., Tong W., Xu X. Constructing a robust protein-protein interaction network by integrating multiple public databases // Bioinformatics, Vol. 12, №10, P. 1-10, 2011. DOI 10.1186/1471-2105-12-s10-s7.
5. Панов П.В. База данных «субнациональный регионализм и многоуровневая политика (reg-mlg)» // Вестник Пермского университета. ПОЛИТОЛОГИЯ, Т. 15, №4, С 111-120, 2021. DOI 10.17072/2218-1067-2021-4-111-120.
6. Haas L.M., Rice J.E., Schwarz P.M., Swope W.C. DISCOVERYLINK: a system for integrated access to life sciences data sources // IBM systems journal, Vol. 40, №2, P. 489-511, 2001. DOI 10.1147/sj.402.0489.
7. Zhang L., Pang Ke., Xu J., Niu B. JSON-based control model for SQL and NoSQL data conversion in hybrid cloud database // Journal of cloud computing, Vol. 11, №1, P. 1-12, 2022. DOI 10.1186/s13677-022-00302-9.