Система автоматического сбора информации о работе NoSQL баз данных

Научный руководитель: *Доцент,к.э.н Самохвалов Алексей Эдуардович*
Консультант: *Доцент,к.т.н. Виноградова Мария Валерьевна*

Студент: Журавлев Николай Вадимович

МГТУ им. Н.Э. Баумана

Россия, Москва, 2025

Постановка задачи

Цель разработки

Создание системы автоматического сбора информации о работе NoSQL баз данных, которая позволяет сделать процесс получения и обработки информации из разных БД более удобным для пользователей

Задачи разработки

1. Произвести анализ средств извлечения информации из Nosql СУБД
2. Исследовать методы и технологии извлечения информации из нескольких СУБД
3. Разработать структуру и архитектуру системы
4. Разработать грамматику языка для запросов к системе
5. Создать метод и алгоритм разбиения запроса к нескольким СУБД на подзапросы для каждой
6. Разработать макет интерфейса системы
7. Произвести тестирование разработанной системы

Функциональные возможности системы по взаимодействию с СУБД

Имеются следующие функциональные возможности взаимодействии с СУБД:

1. Взаимодействие с данными (добавление, удаление, обновление, чтение) в различных БД из одной системы с помощью созданного языка запросов;
2. Взаимодействие с индексами, а именно создание, удаление, просмотр созданных;
3. Получение списка всех таблиц из БД;
4. Просмотр структуры выбранной таблицы из любой БД;
5. Просмотр количества записей в выбранной таблице;
6. Просмотр времени выполнения запроса;
7. Просмотр загрузки оперативной памяти при исполнении запроса;
8. Просмотр плана запроса;
9. Запуск запроса на языке СУБД.

Анализ средств извлечения информации из MongoDB

Для добавления элемента в коллекцию:
db.collection.insert({ "elem_name": "elem_value"})

Всю информацию из коллекции можно удалить через метод remove:
db.collection.remove({ "elem_name ": "elem_value"})

Для обновления коллекции используется метод update:
db.collection.update({ $set: { "elem_name": "new_elem_value"}})

Для поиска используется find:
db.collection.find({ "elem_name": "elem_value"})

Для получения индексов коллекции используется команда:
db.collection.getIndexes()

Для добавления индексов коллекции используется команда:
db.collection.createIndex({ "elem_name1”: 1, "elem_name2”: -1 })

Для удаления индексов коллекции используется команда:
db.collection.dropIndex("index_name”)

Для вывода всех коллекций используется команда:
db.getCollectionNames()

Для вывода строения коллекции используется команда:
db.collection.aggregate([{"$project": {"arrayofkeyvalue": {"$objectToArray": "$$ROOT"}}},
{"$unwind": "$arrayofkeyvalue"},{"$group": {"_id": None, "allkeys": {"$addToSet":
"$arrayofkeyvalue.k"}}}])

Для вывода количества записей в коллекции, используется команда:
db.collection.countDocuments()

Анализ средств извлечения информации из Neo4j

Для простого добавления узла используется метод create:
CREATE (node:label key1: value1, key2: value2, ...)

Узел можно удалить через метод remove:
MATCH (node attribute1: ’value1’) REMOVE node.attribute2

Для обновления узла используется метод update:
MATCH (node attribute1: ’value1’) SET node.attribute2=’value2’ RETURN node.attribute1, node.attribute2

Для поиска значений необходимо указать в конце строки return N:
MATCH (n) WHERE (n.id = 0) RETURN n;

Для получения индексов меток используется команда:
SHOW INDEXES WHERE "name"in labelsOrTypes

Для добавления индексов меток используется команда:
CREATE INDEX name_index FOR (n:name_label) ON (n.prop1, n.prop2)

Для удаления индексов таблицы используется команда:
DROP INDEX name_index

Для вывода всех меток используется команда:
MATCH (n) RETURN DISTINCT labels(n) AS labels

Для вывода строения узлов с конкретными метками используется команда:
MATCH (n:name_label) UNWIND keys(n) AS key RETURN key

Для вывода количества записей с определёнными метками, используется команда:
MATCH (n:name_label) RETURN count(n) AS count

Анализ средств извлечения информации из Cassandra

Для простого добавления строки в таблицу используется метод insert:
INSERT INTO table_name (id, attribute1) VALUES (now(), 'value1');

Запись можно удалить через метод delete:
DELETE FROM table_name WHERE id=54daf810-9aeb-11ea-b1d1-3148925e06e7;

Для обновления строки используется метод update:
UPDATE table_name SET attribute1 = 'value1', WHERE
id=54daf810-9aeb-11ea-b1d1-3148925e06e7;

Для поиска используется команда select:
SELECT attribute1, MAX(attribute2) FROM table_name GROUP BY attribute1;

Для получения индексов таблицы используется команда:
SELECT index_name FROM system_schema.indexes WHERE table_name = 'table_name'
ALLOW FILTERING

Для добавления индексов таблицы используется команда:
CREATE CUSTOM INDEX name_index ON name_table (field_index) USING 'type_index'

Для удаления индексов таблицы используется команда:
DROP INDEX name_index

Для вывода всех таблиц используется команда:
SELECT table_name FROM system_schema.tables

Для вывода строения таблицы используется команда:
SELECT column_name FROM system_schema.columns WHERE table_name = 'table_name'
ALLOW FILTERING

Для вывода количества записей в таблице, используется команда:
SELECT COUNT(*) FROM table_name

Методы извлечения информации из нескольких СУБД. Часть 1

Узконаправленные системы

В данной группе систем специально создаются системы, которые включают в себя заранее определённые БД, как правило связанные общей тематикой. Примером таких систем является DiscoveryLink.
Недостатками является узконаправленное применение, и доступность только БД для конкретной тематики.

Ручное объединение БД

Данная группа пытается вручную объединить несколько разных СУБД. Для этого имеется несколько подходов.
Первый подход. Получить все имеющиеся схемы БД и, сравнивая и редактируя их, можно добиться слияния в единую схему БД.
Второй подход – разработка интегрированной схемы. Этапы разработки интегрированной схемы:

1. Предварительная интеграция, где входные схемы преобразуются, чтобы сделать их более однородными (как синтаксически, так и семантически);
2. Идентификация соответствия, посвященная идентификации и описанию межсхемных отношений;
3. Интеграция, заключительный этап, который разрешает межсхемные конфликты и объединяет соответствующие элементы в интегрированную схему.

Недостатками является необходимость продумывать действия при каждой новой БД и ошибки из-за человеческого фактора.

Методы извлечения информации из нескольких СУБД. Часть 2

Создание новой СУБД

Для решения задачи связи нескольких СУБД можно разработать совершенно новые СУБД. Например, hStorage-DB
Недостатком является отсутствие обмена опытом из-за малой распространённости.

Унифицировать имеющиеся модели БД

Можно каким-либо образом унифицировать модели БД и тогда с ними можно будет удобно взаимодействовать. Основная идея заключается в том, чтобы из разных БД передавать информацию через какой-либо общий вид. Примером общего вида, может являться JSON. Так же можно сделать внутри системы отдельную глобальную БД, которая будет объединять необходимые БД автоматически через себя по заранее продуманным правилам. Например, MOMIS.
Недостатком является необходимо продумывать действия при каждой новой БД.

Взаимодействие в виде графа

Система, в которой все схемы отображаются в виде графа и доступны для взаимодействия через графический интерфейс. В такой системе каждый узел - отдельная схема БД.
Недостатком является ограничение взаимодействия из-за формы отображения в виде графа.

Стандартные компоненты, используемые в системе

В системе использовались следующие библиотеки:

- Для взаимодействия с браузером пользователя используется flask;
- Для взаимодействия с MongoDB используется pymongo;
- Для взаимодействия с Neo4j используется neo4j;
- Для взаимодействия с Cassandra используется cassandra;
- Для взаимодействия с PostgreSQL используется psycopg2;
- Для измерения времени выполнения запроса используется time;
- Для измерения затраченной оперативной памяти на выполнения запроса используется memory_profiler.

Диаграмма классов программы

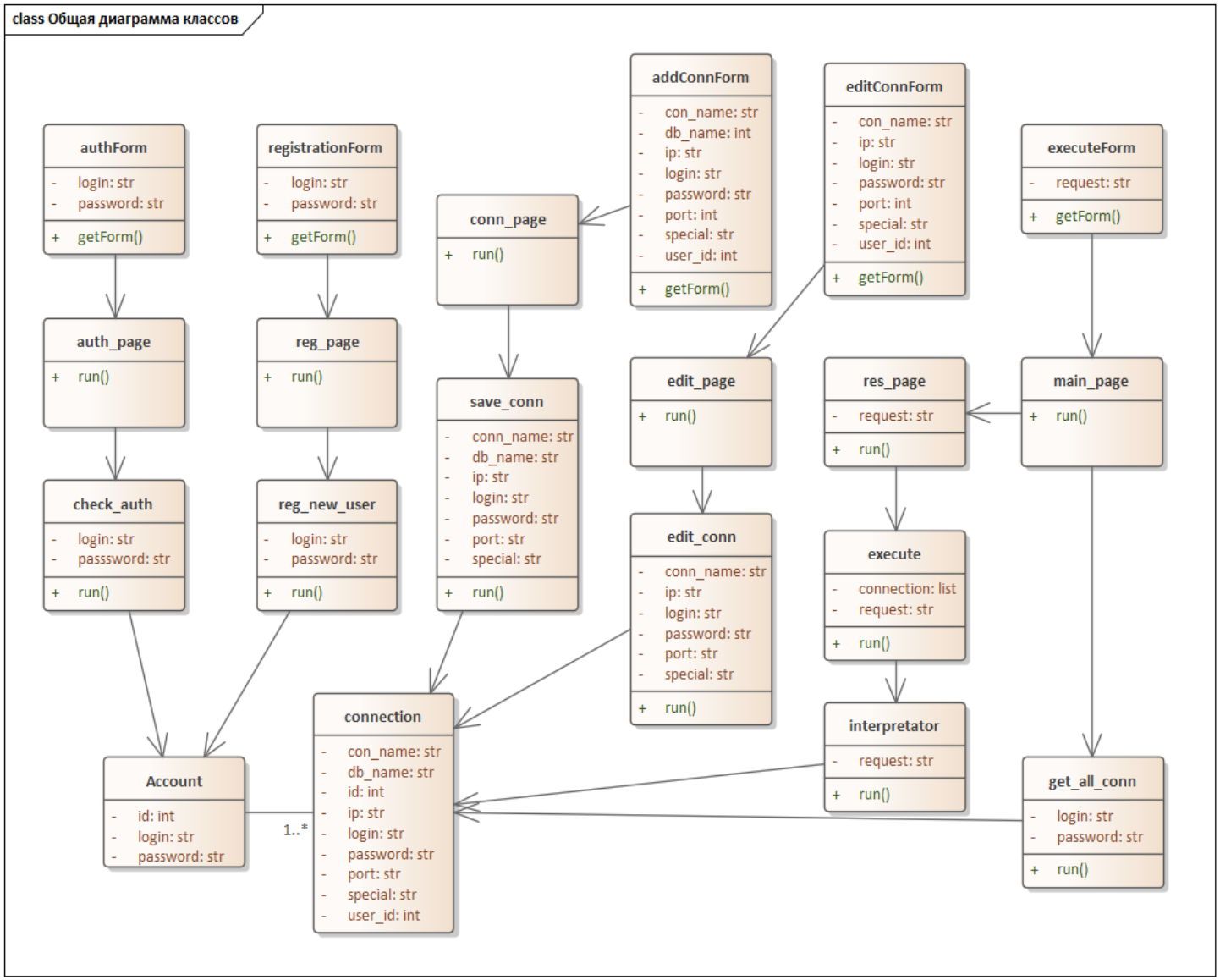
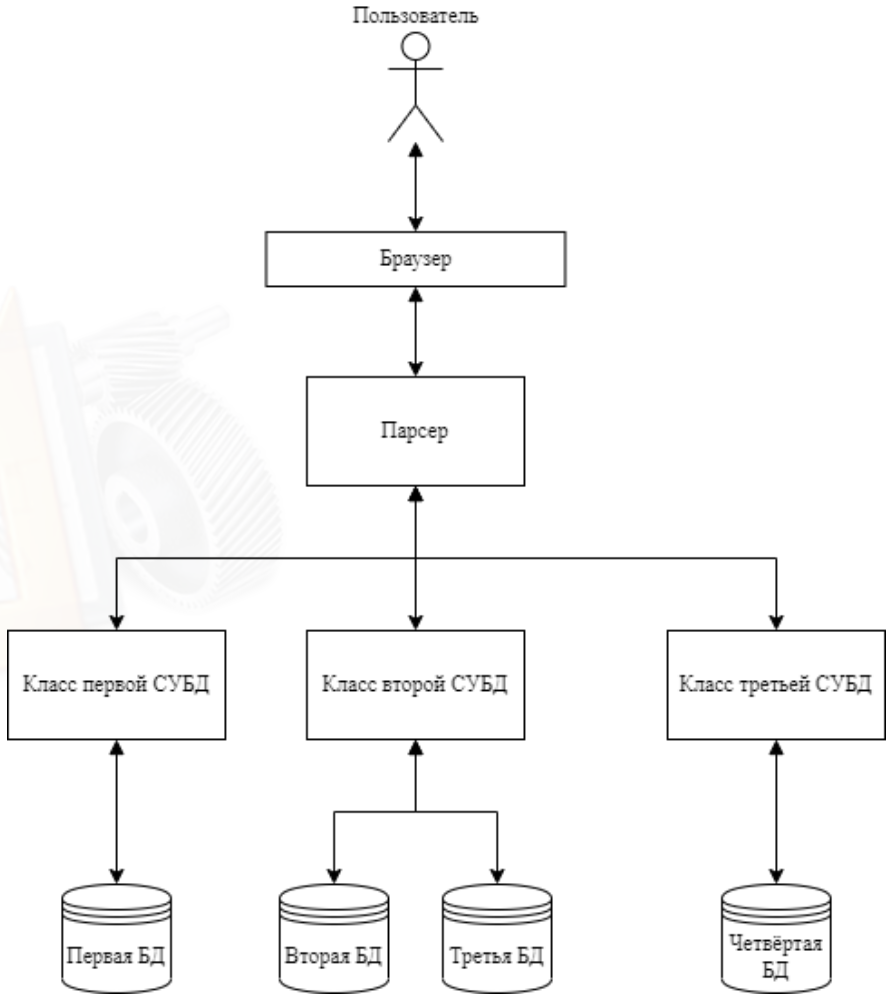


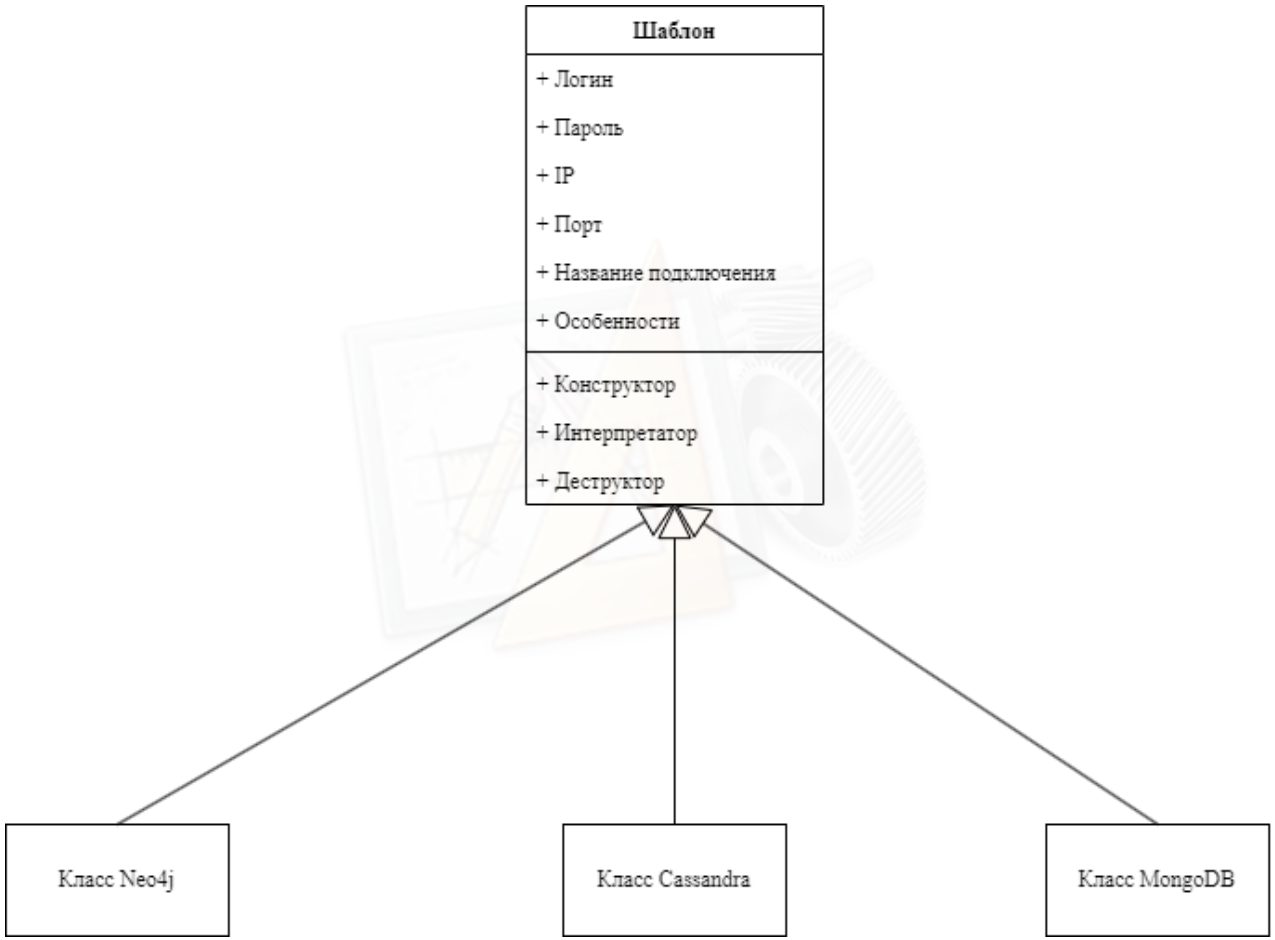
Схема обработки запроса

При обработке запроса можно выделить 3 части:

- Первая часть - ввод запроса пользователем. Пользователь в браузере вводит запрос на получение нужных ему данных в соответствии с созданным языком запросов.
- Вторая часть - парсер запросов. В этом элементе системы запрос пользователя разбивается на подзапросы для каждой БД.
- Третья часть - классы СУБД. Каждый такой класс представляет собой возможность взаимодействия с СУБД для выполнения запроса, а его объекты представляют каждую БД в этой СУБД.



Классы для взаимодействия с СУБД



Грамматика языка запросов к системе. Часть 1

$$\Gamma = \{V_T, V_A, \langle I \rangle, R\}, \text{ где} \quad (1)$$

V_T — терминальный алфавит
V_A — нетерминальный алфавит
$\langle I \rangle$ — начальный символ грамматики
R - множество порождающих правил

$V_T = \{a, b, \ldots, z, A, B, \ldots, Z, 0, 1, \ldots, 9, ., (,), ,, =, !=, <, >, <=, >=,$
AND,OR,where, create, read, update, delete, [,], create_index, delete_index, exec, index, all, show,count }

$V_A = \{\langle I \rangle, \langle$операция чтения$\rangle, \langle$доступ$\rangle, \langle$условие$\rangle, \langle$операция$\rangle,$
$\langle$подключение$\rangle, \langle$таблица$\rangle, \langle$столбец$\rangle, \langle$сравнение$\rangle, \langle$константа$\rangle, \langle$строка$\rangle,$
$\langle$буква$\rangle, \langle$цифра$\rangle, \langle$число$\rangle, \langle$список$\rangle, \langle$элемент списка$\rangle, \langle$список списков$\rangle\}$

R = { $\langle I \rangle \rightarrow \langle$подключение$\rangle.\langle$операция$\rangle,$
$\langle$операция чтения$\rangle \rightarrow$ read($\langle$доступ$\rangle$) | read($\langle$доступ$\rangle$).where($\langle$условие$\rangle$),
$\langle$операция$\rangle \rightarrow$ create($\langle$доступ$\rangle, \langle$доступ$\rangle$, [$\langle$список списков$\rangle$]) |
update($\langle$доступ$\rangle, \langle$доступ$\rangle$, [$\langle$список$\rangle$]) |
update($\langle$доступ$\rangle, \langle$доступ$\rangle, \langle$список$\rangle$).where($\langle$условие$\rangle$) |
delete.where($\langle$условие$\rangle$) |
create_index($\langle$константа$\rangle, \langle$константа$\rangle$, [$\langle$список$\rangle$]) |
delete_index($\langle$константа$\rangle$) | exec($\langle$константа$\rangle$) | index($\langle$константа$\rangle$) |
all() | show($\langle$константа$\rangle$) | count($\langle$константа$\rangle$),

Грамматика языка запросов к системе. Часть 2

⟨доступ⟩ → ⟨подключение⟩.⟨таблица⟩.⟨столбец⟩,
⟨условие⟩ → ⟨доступ⟩⟨сравнение⟩⟨константа⟩ |
⟨условие⟩ AND ⟨условие⟩ | ⟨условие⟩ OR ⟨условие⟩ |
⟨доступ⟩⟨сравнение⟩⟨операция чтения⟩,
⟨сравнение⟩ →$=|!=|<|>|\leqslant|\geqslant$,
⟨константа⟩ → ”⟨строка⟩” | ⟨число⟩,
⟨подключение⟩ → ⟨строка⟩,
⟨таблица⟩ → ⟨строка⟩,
⟨столбец⟩ → ⟨строка⟩,
⟨строка⟩ → ⟨строка⟩⟨буква⟩ | ⟨строка⟩⟨цифра⟩ | $,
⟨буква⟩ → $a \mid b \mid \cdots \mid z \mid A \mid B \mid \cdots \mid Z \mid \ \mid \ \mid \cdots \mid \ \mid \ \mid \ \mid \cdots \mid$,
⟨цифра⟩ → $0 \mid 1 \mid \cdots \mid 9$,
⟨число⟩ → ⟨число⟩⟨цифра⟩ | $,
⟨список⟩ → ⟨список⟩, ⟨элемент списка⟩ | ⟨элемент списка⟩,
⟨элемент списка⟩ → ⟨константа⟩ | ⟨элемент списка⟩, ⟨константа⟩,
⟨список списков⟩ → ⟨список⟩ | ⟨список списков⟩, ⟨список⟩}

Команды запросов к системе

В системе представлены следующие операторы:

1. **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.
2. **where** – оператор, в котором в скобках описывается фильтр, по которому выбираются данные из БД. В условии используются оператор доступа к данным и операторы сравнения.
3. **create** – у оператора в скобках сначала через запятую указываются поля для заполнения, после чего в конце через запятую указываются вставляемые данные в виде массива списков.
4. **read** – у оператора в скобках указываются операторы доступа к данным, которые должны быть прочитаны.
5. **update** – у оператора в скобках сначала через запятую указываются поля, в которых будут производится изменения, после чего в конце через запятую указываются вставляемые данные в виде списка.
6. **delete** – удаление данных из БД. После данного оператора указывается оператор where.
7. **index** – получение индексов для конкретной таблицы.
8. **all** – получить список всех таблиц, имеющихся в БД.
9. **show** – показ структуры выбранной таблицы.
10. **count** – количество записей в таблице.
11. **create_index** – добавление индекса.
12. **delete_index** – удаление индекса.
13. **exec** – исполняет код, который указан в аргументе, на языке запросов СУБД.

Пример преобразования запроса

1. Запрос на чтение:
   dbpeople.read(dbpeople.personal_data.name).where(dbpeople.personal_data.id=1)

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

   Cassandra:
   SELECT name FROM personal_data WHERE id = 1;

   Neo4j:
   MATCH (p:personal_data) WHERE p.id = 1 RETURN p.name
2. Запрос на создание данных:
   dbpeople.create(dbpeople.personal_data.id, dbpeople.personal_data.name, [[2, "PetrovPP "]])

   MongoDB:
   db.dbpeople.insertOne({"personal_data ": {"id": 2, "name": "PetrovPP "}});

   Cassandra:
   INSERT INTO dbpeople (id, name) VALUES (2, 'PetrovPP');

   Neo4j:
   CREATE (p:Person id: 2, name: 'PetrovPP');
3. Запрос на получение информации о индексах:
   dbpeople.index(human)

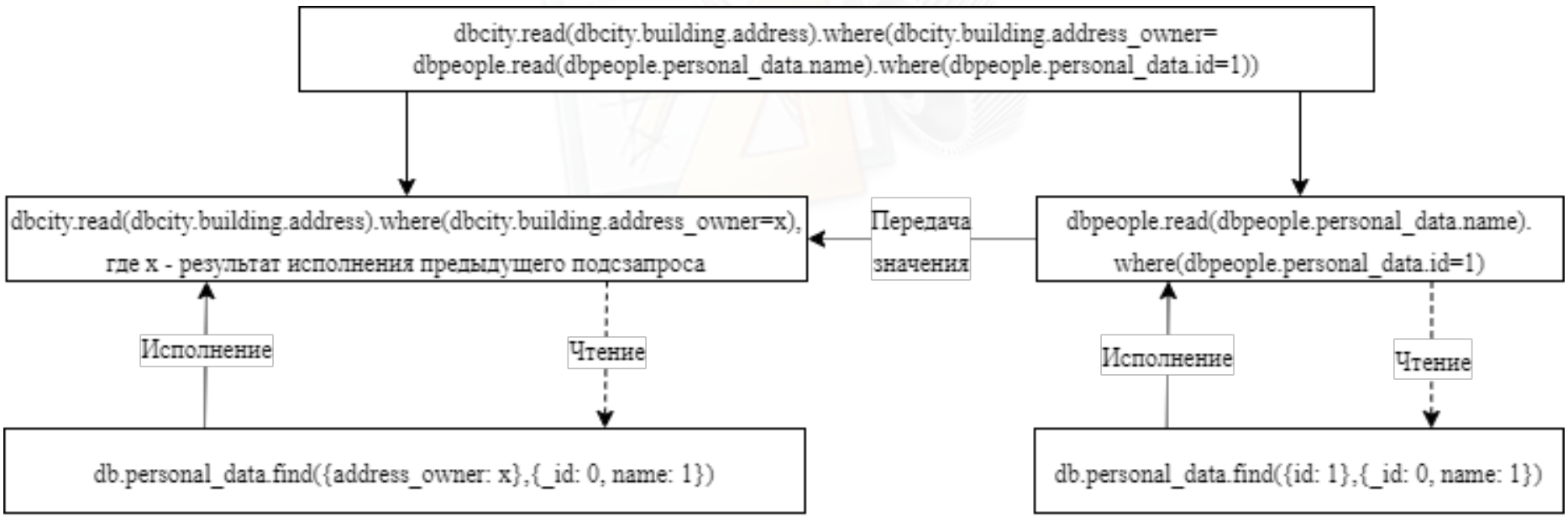
   MongoDB:
   db.human.getIndexes();

   Cassandra:
   SELECT index_name FROM system_schema.indexes WHERE table_name = 'human' ALLOW FILTERING

   Neo4j:
   SHOW INDEXES WHERE "human"in labelsOrTypes

Пример исполнения запроса

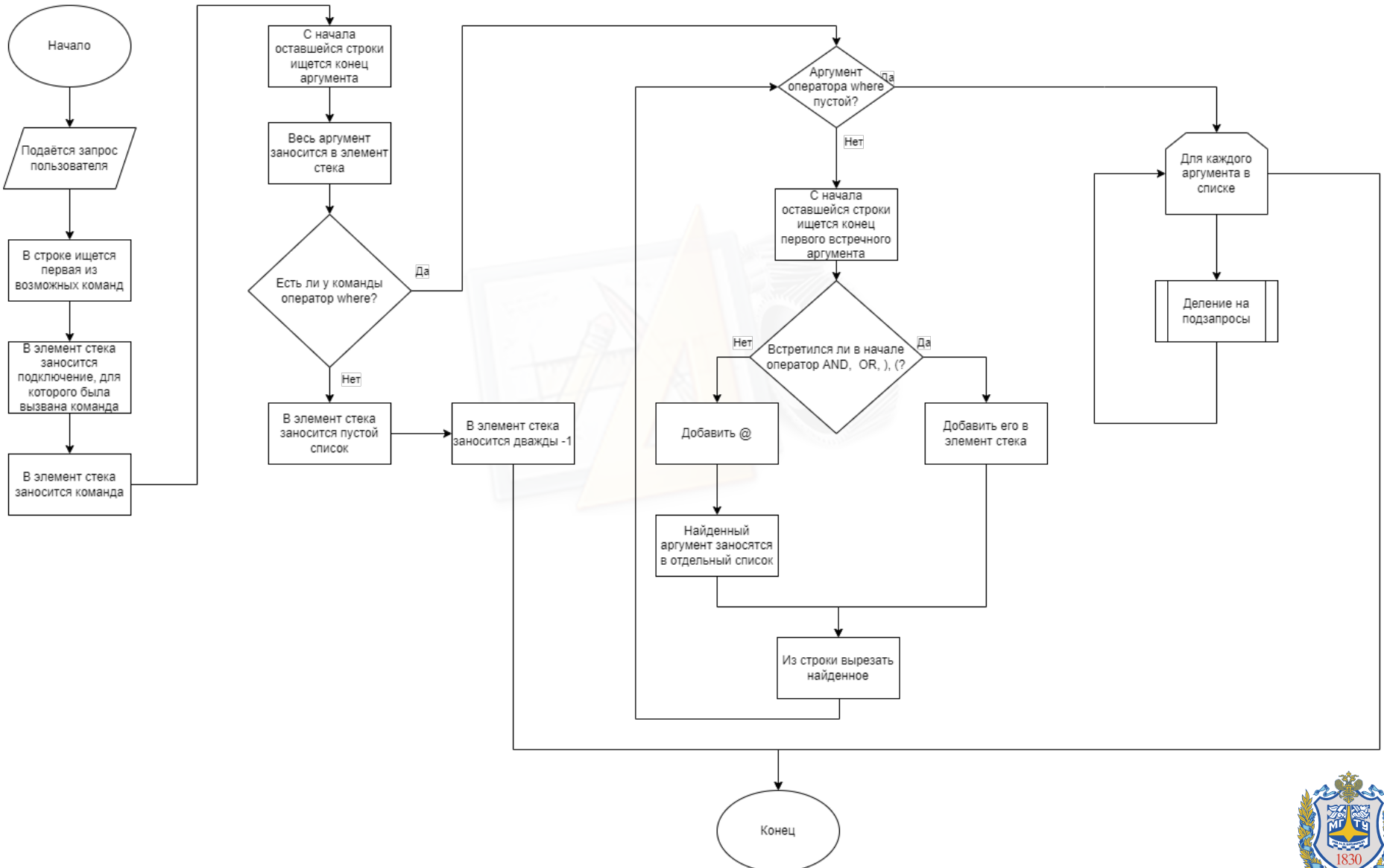
1. Парсер начинает выполнять с наиболее вложенного подзапроса:(dbpeople.personal_data.name).where(dbpeople.personal_data.id=1), где dbpeople – БД в СУБД MongoDB.
2. dbpeople.personal_data.name преобразуется в _id: 0, name: 1
3. where(dbpeople.personal_data.id=1) преобразуется в _id: 1
4. Далее конвертируется часть запроса dbpeople.read() и исполняется в виде следующего запроса: db.personal_data.find(_id: 1,_id: 0, name: 1).



Метод деления на подзапросы

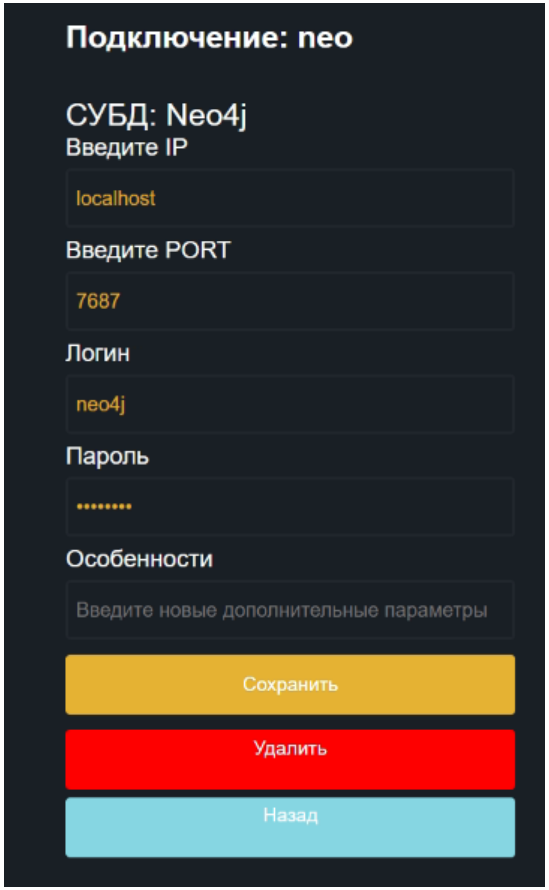
1. В строке ищется первая из возможных команд (create, read, update и т.п.).
2. В элемент стека заносится подключение, для которого была вызвана команда, и затем сама команда.
3. С начала оставшейся строки ищется конец аргумента.
4. Весь аргумент заносится в элемент стека.
5. Далее проверяется, имеется ли у команды оператор where. Если он отсутствует, то в элемент стека заносится пустой список и на место координат вставки результата записывается -1.
6. В случае наличия оператора where ищется конец его аргумента.
7. Начинается посимвольное прохождение аргумента, пока не будет встречен оператор AND или OR.
8. Символы до первого аргумента, вносится отдельно в список where и вместо аргумента ставится @. Найденный аргумент заносятся в отдельный список. Затем заносится сам оператор AND или OR.
9. Если была пройдена не вся строка, то возврат к пункту 7.
10. Для каждого аргумента в списке ищется оператор сравнения.
11. Каждый аргумент разделителя, если они простые, заносится вместо @ и сам разделитель между ними.
12. Если же они составные, то для каждой части в элемент стека заносится итерация и место, которое помечается символом % и возврат к пункту 1

Алгоритм деления на подзапросы

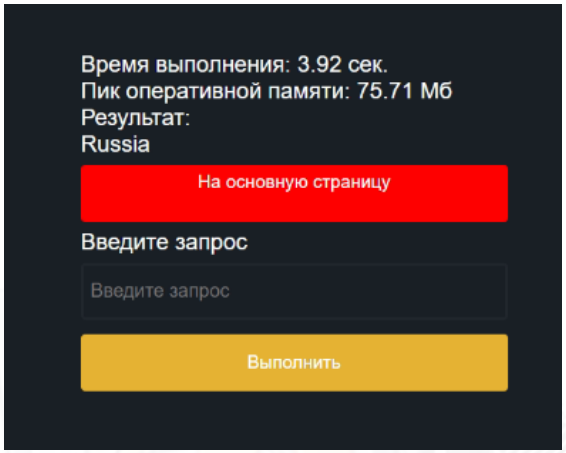


Интерфейс системы

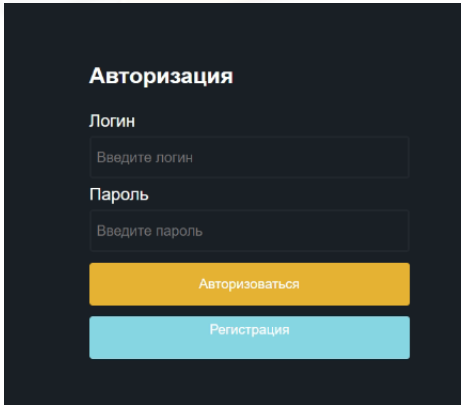
Редактирование подключения



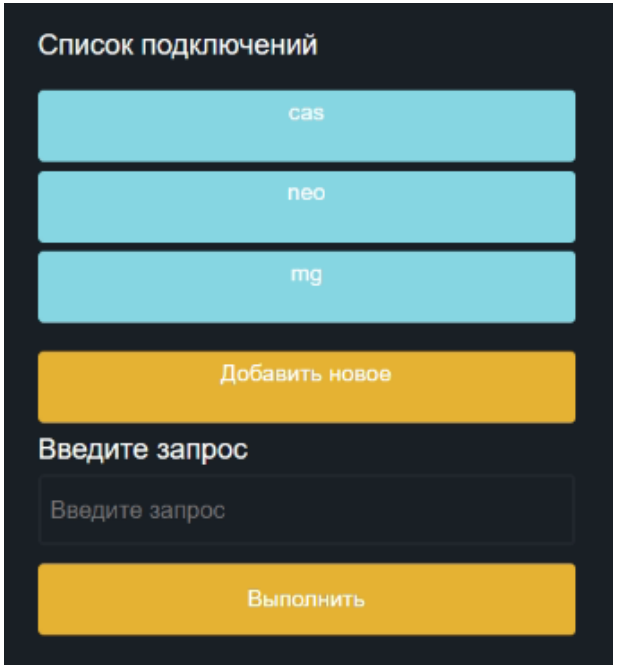
Страница результата



Авторизация



Основная страница



Функциональное тестирование

Цель тестирования: Проверка корректности и полноты выполняемых системой функций.
Метод тестирования: Причина / Следствие (Cause/Effect — CE).
Параметры тестирования некоторых тест-кейсов:

1. Тест-кейс 1. Чтение данных из MongoDB
   Необходимо создать тестовое БД, в которой содержится запись с id = 0, name = IvanovII. На главной странице необходимо ввести запрос на чтение.
   Входные данные:
   Запрос: mg.read(mg.human.name).where(mg.human.id=0)
   Ожидаемый результат: IvanovII
2. Тест-кейс 2. Чтение данных из Neo4j
   Необходимо создать тестовое БД, в которой содержится запись city_name = Msk human_name = IvanovII. На главной странице необходимо ввести запрос на чтение.
   Входные данные:
   Запрос: neo.read(neo.city.city_name).where(neo.city.human_name="IvanovII")
   Ожидаемый результат: Msk
3. Тест-кейс 3. Чтение данных из Cassandra
   Необходимо создать тестовое БД, в которой содержится запись city_name = Msk, country_name = Russia . На главной странице необходимо ввести запрос на чтение.
   Входные данные:
   Запрос: cas.read(cas.country.country_name).where(cas.country.city_name="Msk")
   Ожидаемый результат: Russia

Вывод: Исходя из результатов тестирования можно сделать вывод, что все необходимые функции работают корректно

Модульное тестирования

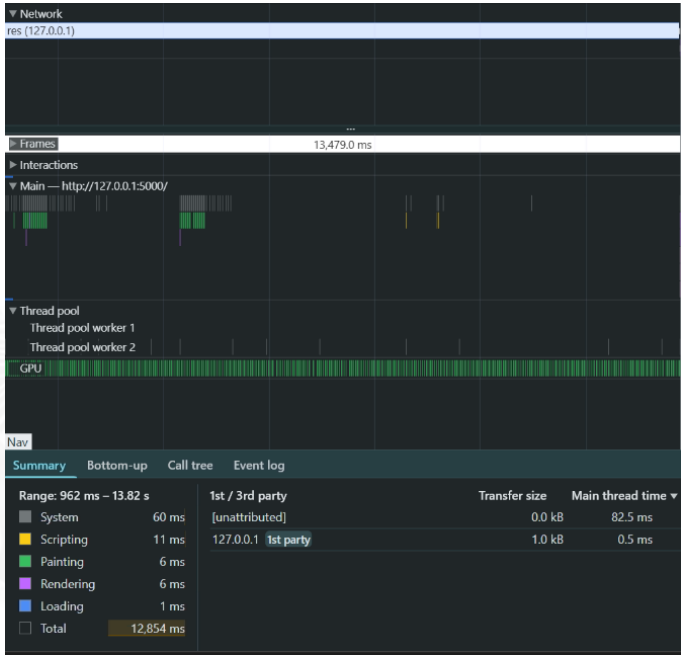
Цель - проверить функциональность и найти ошибки в частях приложения, которые доступны и могут быть протестированы по-отдельности.
Метод тестирования - Причина / Следствие (Cause/Effect — CE).
Пример тестирования преобразования из запроса на созданном языке в универсальный вид, который после будет интерпретироваться каждой СУБД, представлен на рисунке.
Входными данными является запрос - conn1.read(conn1.table1.field1).
Выходными данными является результат, который будет выведен в результате вызова mock-функции.
Вывод: покрытие кода тестами составляет 80%, что означает, что основные функции программы работают корректно.

```
def test_execute_simple_request(self):
    parser = ParserDB("conn1.read(conn1.table1.field1)", self.connections)
    result, err, stat = parser.execute()

    self.assertEqual(result, "result1")
    self.assertEqual(err, "")
    self.assertEqual(stat["time"], 0.1)
    self.assertEqual(stat["pik_memory"], 100)
    self.mock_conn1.interpreter.assert_called_with(['conn1', 'read', 'conn1.table1.field1', -1, -1])
```

Веб-тестирование производительности

Для тестирования выполнения запроса был выбран следующий запрос:
cas.read(cas.country.country_name)
.where(cas.country.city_name=
neo.read(neo.city.city_name)
.where(neo.city.human_name=
mg.read(mg.human.name).where(mg.human.id=0)))
Полученные результаты тестирования:



1. Загрузка (Loading) страницы заняла 1 мс
2. Выполнение и загрузка скрипта (Scripting) страницы заняло 11 мс
3. Рендеринг, а именно отрисовка страницы (Rendering) страницы занял 6 мс
4. Погрузка css - стилей страницы (Painting) страницы заняла 6 мс
5. Погрузка системных браузерных настроек (System) страницы заняла 60 мс
6. Общая загрузка (Total) заняла 12,854 мс

Вывод: Исходя из результатов, можно сделать вывод, что приложение загружает страницу с удовлетворительной скоростью

Заключение

- Для каждой СУБД были изучены запросы, которые необходимы для запросов системы при взаимодействии с ними.
- Для выделения преимуществ и недостатков системы были изучены аналогичные работы по данной тематике. Была разработана грамматика языка для запросов к системе, алгоритм деления этого запроса на подзапросы для каждой СУБД и функции преобразования в запрос на язык СУБД, для которой он предназначается.
- В итоге, в данной работе была разработана и протестирована с помощью 3 видов тестирования система, которая позволяет пользователю получать и анализировать информацию из разных БД, основанных на разных СУБД.

Список статей

- Алексеев А.С., Журавлев Н.В., Волков А.С., Самохвалов А.Э. Система для сбора и взаимодействия данных между различными СУБД // ИИАСУ’25. Сборник статей всероссийской научной конференции. – М.: ИНФРА-М. – 2025. – Т.2. – С. 400-405.
- Алексеев А.С., Журавлёв Н.В., Саргсян О.Г. МЭС по выбору СУБД для решения задач // МИВАР’25.

Спасибо за внимание!