|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_ИНФОРМАТИКА, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И СИСТЕМЫ УПРАВЛЕНИЯ\_\_

КАФЕДРА \_\_\_\_\_\_\_\_\_\_\_СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ***

***НА ТЕМУ:***

***Веб-сервер для взаимодействия с разнородными\_\_\_\_ базами данных \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

Студент \_\_\_ИУ5-35М\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **\_\_\_\_\_А.С.Алексеев\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_Ю.Е.Гапанюк\_\_**

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_ Ю.Е.Гапанюк \_\_**

(Подпись, дата) (И.О.Фамилия)

*2024 г.*

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой \_\_\_\_\_\_\_\_\_\_

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(И.О.Фамилия)

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_\_ г.

**ЗАДАНИЕ**

**на выполнение научно-исследовательской работы**

по теме \_\_Веб-сервер для взаимодействия с разнородными базами данных\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент группы \_\_\_\_ ИУ5-35М\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Алексеев Андрей Сергеевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

\_\_\_\_\_\_\_\_учебная\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Источник тематики (кафедра, предприятие, НИР) \_\_\_\_\_\_\_\_\_кафедра\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

График выполнения НИР: 25% к \_\_\_ нед., 50% к \_\_\_ нед., 75% к \_\_ нед., 100% к \_\_\_ нед.

Техническое задание Теоретическое и практическое исследование технологий взаимодействия с мультимодельной базой данных на примере СУБД PosgreSQL, Mongo, SQLite, Neo4j\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Оформление научно-исследовательской работы:***

Расчетно-пояснительная записка на \_\_\_\_\_ листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания « \_9\_ » \_\_\_февраля\_\_ 2024 г.

**Руководитель НИР**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_ **Ю.Е.Гапанюк** \_\_

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_ А.С.Алексеев\_\_**

(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

**Оглавление**

[ВВЕДЕНИЕ 4](#_Toc168420104)

[РЕАЛИЗАЦИЯ 6](#_Toc168420105)

[ЗАКЛЮЧЕНИЕ 16](#_Toc168420106)

[СПИСОК ЛИТЕРАТУРЫ 17](#_Toc168420107)

# ВВЕДЕНИЕ

В настоящее время наблюдается продолжающийся рост систем, поддерживающих огромный объем реляционных или нереляционных форм данных. Примерами моделей данных, которые поддерживают многомодельные базы данных, являются документные, графические, реляционные модели и модели ключ-значение. Наличие единой платформы данных для управления как хорошо структурированными данными, так и данными NoSQL выгодно пользователям, поскольку одни пользователи могут предпочитать работать с данными в табличной форме, другие могут предпочитать графические представления или диаграммы. Предоставление разных представлений данных позволяет удовлетворить эти потребности и сделать информацию более доступной и понятной для пользователей. Также такая платформа улучшит визуализацию и понимания данных.

Всё большее распространение многомодельных баз данных, предназначенных для поддержки нескольких моделей данных в рамках единой интегрированной серверной части имеет свои обоснования. Разные БД могут иметь свои специфические типы данных, которые могут быть полезными для работы с конкретными приложениями или сценариями. Взаимодействие с разными БД позволяет получить доступ к различным типам данных и использовать их в соответствии с требованиями проекта. Взаимодействие сразу с несколькими БД может быть необходимым для работы с распределенными системами, где данные хранятся на разных серверах или локациях. Такая архитектура может обеспечить более высокую доступность и масштабируемость данных, а также улучшить производительность. Помимо этого, взаимодействие с разными БД может быть полезным при интеграции с внешними источниками данных, такими как сторонние сервисы, веб-API и другие системы. Это позволяет получать данные из разных источников и комбинировать их для более полной и точной информации. Также каждая БД имеет свои особенности и функциональности, и взаимодействие с несколькими БД может дать возможность использовать различные возможности каждой из них. Например, некоторые БД могут предоставлять мощные инструменты аналитики и отчетности, в то время как другие могут быть оптимизированы для быстрого чтения и записи данных. Наконец, взаимодействие сразу с несколькими БД может быть необходимым при миграции данных между разными системами. Это может включать синхронизацию, перенос или конвертацию данных из одной БД в другую. В целом, взаимодействие с разными БД позволяет получить доступ к более широкому спектру данных, использовать различные функциональности каждой БД и улучшить гибкость и масштабируемость приложений.

Было бы очень удобно, если взаимодействие с разными БД было представленное в виде веб-сервера.

Во-первых, при использовании веб-сервера в качестве промежуточного слоя между приложением и базой данных, можно создать унифицированный интерфейс для взаимодействия с разными базами данных, что позволит разработчикам использовать общие методы и функции для работы с данными в разных базах данных, что упростит разработку и поддержку приложения.

Во-вторых, можно легко добавлять или изменять базы данных без необходимости изменения или перекомпиляции основного приложения, благодаря чему можно будет легко масштабировать систему и добавлять новые базы данных по мере необходимости.

В-третьих, веб-сервер может обеспечить дополнительный уровень безопасности путем реализации аутентификации и авторизации на уровне сервера из-за чего будет контроль доступа к данным в базах данных и обеспечение дополнительной защиты от несанкционированного доступа. В-четвёртых, можно централизованно управлять доступом к данным, анализировать и мониторить запросы и производительность баз данных. В итоге это позволит упростить процесс управления базами данных и обеспечит более эффективное использование ресурсов.

Всё, перечисленное выше, обосновывает актуальность разработки.

Веб-сервер для взаимодействия с разнородными базами данных будет использовать Python для взаимодействия с базами данных. Для подключения баз данных к Python необходимо использовать сторонние библиотеки. Для подключения к объектно-реляционной базой данных PostgreSQL используется библиотека psycopg2, к MongoDB pymongo, к Neo4j neo4j. После подключения идёт проверка операций CRUD и вывод данных в общей для всех БД форме.

# РЕАЛИЗАЦИЯ

Для большинства операционных систем самым быстрым способом установки Psycopg является использование пакета wheel, доступного в PyPI:

$ pip install psycopg2-binary

При этом будет установлена предварительно скомпилированная двоичная версия модуля, которая не требует описанных ниже предварительных условий сборки или среды выполнения.

Импорт установленной библиотеки:

import psycopg2 #Объектно-реляционная БД

Перед подключением к PostgreSQL необходимо, чтобы уже существовала база данных. Подключение к существующей базе данных PostgreSQL "OrderingDishes":

connection = psycopg2.connect(dbname="OrderingDishes", user="postgres", password="12345", host="127.0.0.1") # подключение

Для работы с базой данных необходимо создать курсор, через который будет проходить взаимодействие. Создание курсора:

cursor = connection.cursor()# курсор

Заранее созданная база данных "OrderingDishes" является пустой и её необходимо заполнить. Для этого создаём таблицы:

create\_users\_query = '''CREATE TABLE users (   
 IdUser INTEGER PRIMARY KEY,  
 Name VARCHAR(20),  
 Mail VARCHAR(20),  
 Surname VARCHAR(20),  
 Login VARCHAR(20),  
 Password VARCHAR(20),  
 Phone BIGINT  
 );'''  
cursor.execute(create\_users\_query)  
connection.commit()  
create\_orders\_query = '''CREATE TABLE orders (   
 IdOrder INTEGER PRIMARY KEY,  
 IdUserA INTEGER UNIQUE,  
 Price INTEGER,  
 Condition VARCHAR(20)х,  
 FOREIGN KEY (IdUserA) REFERENCES users (IdUser) ON UPDATE CASCADE  
 );'''  
cursor.execute(create\_orders\_query)  
connection.commit()

Далее необходимо заполнить таблицы. Для этого производит запрос CREATE, для которого необходимо создать переменную со структурой запроса и переменную с данными для запроса, после чего обе переменные используются в курсоре:

postgres\_insert\_users\_query = """INSERT INTO users (IdUser, Name, Mail, Surname, Login, Password, Phone)   
VALUES (%s, %s, %s, %s, %s, %s, %s)""" # выполнение запроса CREATE  
record\_users\_to\_insert1 = (1, 'Vasiliy', 'vasya@gmail.com', 'Kotov', 'user1', '39757123', 89017482454)  
record\_users\_to\_insert2 = (2, 'Alexandr', 'alex@mail.ru', 'Frolov', 'user2', '48725075', 89038127430)  
record\_users\_to\_insert3 = (3, 'Andrey', 'andruha@gmail.com', 'Uralov', 'andruha2', '63289136', 89048623806)  
cursor.execute(postgres\_insert\_users\_query, record\_users\_to\_insert1)  
connection.commit()  
cursor.execute(postgres\_insert\_users\_query, record\_users\_to\_insert2)  
connection.commit()  
cursor.execute(postgres\_insert\_users\_query, record\_users\_to\_insert3)  
connection.commit()

postgres\_insert\_orders\_query = """INSERT INTO orders (IdOrder, IdUserA, Price, Condition)   
VALUES (%s, %s, %s, %s)"""  
record\_orders\_to\_insert1 = (1, 1, 450, 'in road')  
record\_orders\_to\_insert2 = (2, 2, 560, 'in the assembly')  
cursor.execute(postgres\_insert\_orders\_query, record\_orders\_to\_insert1)  
connection.commit()  
cursor.execute(postgres\_insert\_orders\_query, record\_orders\_to\_insert2)  
connection.commit()

Для запроса READ и вывода данных в Python используется только переменная с запросом:

cursor.execute("SELECT \* FROM users ORDER BY Login DESC;") # выполнение запроса READ  
results = cursor.fetchall()  
print(results)

Для изменения данных в таблицах используется запрос UPDATE, для которого необходимо создать переменную со структурой запроса и переменную с данными для запроса, после чего обе переменные используются в курсоре:

postgres\_update\_users\_query = """UPDATE users SET Login = %s WHERE Login = %s""" # выполнение запроса UPDATE  
record\_users\_to\_update1 = ('vk111', 'user1')  
record\_users\_to\_update2 = ('Alex1', 'user2')  
cursor.execute(postgres\_update\_users\_query, record\_users\_to\_update1)  
connection.commit()  
cursor.execute(postgres\_update\_users\_query, record\_users\_to\_update2)  
connection.commit()  
cursor.execute("SELECT \* FROM users ORDER BY Login DESC;") # выполнение запроса READ  
results = cursor.fetchall()  
print(results)

Для удаления данных в таблицах используется запрос DELETE, для которого необходимо создать переменную со структурой запроса и переменную с данными для запроса, после чего обе переменные используются в курсоре:

postgres\_delete\_users\_query = """DELETE FROM users WHERE Login = %s""" # выполнение запроса DELETE  
cursor.execute(postgres\_delete\_users\_query, ('andruha2',))  
connection.commit()  
cursor.execute("SELECT \* FROM users ORDER BY Login DESC;") # выполнение запроса READ  
results = cursor.fetchall()  
print(results)

Получение и запись таблиц:

cursor.execute("SELECT \* FROM users;") # выполнение запроса READ  
users = cursor.fetchall()  
cursor.execute("SELECT \* FROM orders;") # выполнение запроса READ  
orders = cursor.fetchall() # порядок чтения результатов запроса  
return users, orders

Очистка курсора и отключение от базы данных:

cursor.close() # очистка курсора cursor  
connection.close() # отключение

Для работы с документной базой данных MongoDB на языке Python используется библиотека pymongo.

Установка библиотеки:

python3 -m pip install pymongo

После установки необходимо, чтобы экземпляр MongoDB был запущен на хосте и порту. Для запуска необходимо сделать следующее:

from pymongo import MongoClient #Документная БД

В этом руководстве также предполагается, что экземпляр MongoDB запущен на хосте и порту по умолчанию. Предполагая, что вы скачали и установили MongoDB, вы можете запустить его следующим образом:

$ mongod

Установление соединения с MongoClient:

Первым шагом при работе с PyMongo является создание MongoClient для запущенного экземпляра mongod. Приведенный ниже код подключится к хосту и порту по умолчанию. Также можно использовать формат URI MongoDB:

client = MongoClient("mongodb://localhost:27017/") # подключение

Если имя базы данных таково, что использование доступа в стиле атрибута не сработает (например, test-database), можно вместо этого использовать доступ в стиле словаря:

db1 = client['OrderingDishes1']

Коллекция - это группа документов, хранящихся в MongoDB, и ее можно рассматривать примерно как эквивалент таблицы в реляционной базе данных. Создание коллекции пользователей:

users = db["users"]  
user1 = {  
 "Name": 'Vasiliy',  
 "Mail": 'vasya1@gmail.com',  
 "Surname": 'Kotov',  
 "Login": 'user1',  
 "Password": '39757123',  
 "Phone": 89017482454  
}  
user2 = {  
 "Name": 'Alexandr',  
 "Mail": 'alex1@mail.ru',  
 "Surname": 'Frolov',  
 "Login": 'user2',  
 "Password": '48725075',  
 "Phone": 89038127430  
}  
user3 = {  
 "Name": 'Andrey',  
 "Mail": 'andruha@gmail.com',  
 "Surname": 'Uralov',  
 "Login": 'andruha2',  
 "Password": '63289136',  
 "Phone": 89048623806  
}

Документы могут содержать собственные типы Python (например, экземпляры datetime.datetime), которые будут автоматически преобразованы в соответствующие типы BSON и из них. Чтобы вставить документ в коллекцию можно использовать метод insert\_one()(Запрос CREATE):

user\_insert1 = users.insert\_one(user1) # выполнение запроса CREATE  
user\_insert2 = users.insert\_one(user2)  
user\_insert3 = users.insert\_one(user3)

Создание коллекции заказов и её заполнение:

ID = []  
cursor1 = db.users.find() # курсор READ  
for doc in cursor1:  
 ID.append(doc["\_id"]) # порядок чтения результатов запроса  
orders = db["orders"]  
order1 = {  
 "IdUserA": str(ID[0]),  
 "Price": 450,  
 "Condition": 'in road'  
}  
order2 = {  
 "IdUserA": str(ID[1]),  
 "Price": 560,  
 "Condition": 'in the assembly'  
}  
order\_insert1 = orders.insert\_one(order1) # выполнение запроса CREATE  
order\_insert2 = orders.insert\_one(order2)

Самый простой тип запроса, который может быть выполнен в MongoDB, - это find(). Этот метод возвращает единственный документ, соответствующий запросу (или None, если совпадений нет). Это полезно, когда известно, что есть только один соответствующий документ или только первое совпадение. Запрос READ и вывод данных в Python:

cursor1 = db.users.find() # курсор, запрос READ  
for doc in cursor1:  
 print(doc) # порядок чтения результатов запроса

Для изменения данных в коллекциях используется метод update\_one(), который изменяет данные только в одном документе. Запрос UPDATE:

myquery1 = {"Login": "user1"}  
newvalues1 = {"$set": {"Login": "vk111"}}  
myquery2 = {"Login": "user2"}  
newvalues2 = {"$set": {"Login": "Alex1"}}  
user\_update1 = users.update\_one(myquery1, newvalues1) # выполнение запроса UPDATE  
user\_update2 = users.update\_one(myquery2, newvalues2)  
cursor1 = db.users.find() # курсор READ  
for doc in cursor1:  
 print(doc) # порядок чтения результатов запроса

Для удаления данных в коллекциях используется метод delete\_many(), который удаляет данные только в одном документе. Запрос DELETE:

deluser = {"Login": "andruha2"}  
result = users.delete\_many(deluser) # выполнение запроса DELETE  
cursor1 = db.users.find() # курсор READ  
for doc in cursor1:  
 print(doc) # порядок чтения результатов запроса

Создание второй базы данных:

db2 = client['OrderingDishes2']

Создание коллекции пользователей, в которой будут храниться данные о заказах пользователей, а не в отдельной коллекции:

users = db["users"]  
user1 = {  
 "Name": 'Vasiliy',  
 "Mail": 'vasya2@gmail.com',  
 "Surname": 'Kotov',  
 "Login": 'user1',  
 "Password": '39757123',  
 "Phone": 89017482454,  
 "Orders": [  
 {  
 "IdOrder": 1,  
 "IdUserA": 1,  
 "Price": 450,  
 "Condition": 'in road'  
 }  
 ]  
}  
user2 = {  
 "Name": 'Alexandr',  
 "Mail": 'alex2@mail.ru',  
 "Surname": 'Frolov',  
 "Login": 'user2',  
 "Password": '48725075',  
 "Phone": 89038127430,  
 "Orders": [  
 {  
 "IdOrder": 2,  
 "IdUserA": 2,  
 "Price": 560,  
 "Condition": 'in the assembly'  
 }  
 ]  
}  
user3 = {  
 "IdUser": 3,  
 "Name": 'Andrey',  
 "Mail": 'andruha@gmail.com',  
 "Surname": 'Uralov',  
 "Login": 'andruha2',  
 "Password": '63289136',  
 "Phone": 89048623806,  
 "Orders": [  
 {  
 "IdOrder": 3,  
 "IdUserA": 3,  
 "Price": 100000,  
 "Condition": 'What?'  
 }  
 ]  
}

Запрос CREATE:

user\_insert1 = users.insert\_one(user1) # выполнение запроса CREATE  
user\_insert2 = users.insert\_one(user2)  
user\_insert3 = users.insert\_one(user3)  
cursor1 = db.users.find() # курсор READ  
ids = []  
for doc in cursor1:  
 print(doc) # порядок чтения результатов запроса  
 ids.append(str(doc['\_id']))

Запрос UPDATE:

myquery1 = {"Login": 'user1'}  
newvalues1 = {"$set": {"Login": 'vk111'}}  
myquery2 = {"Login": 'user2'}  
newvalues2 = {"$set": {"Login": 'Alex1'}}  
user\_update1 = users.update\_one(myquery1, newvalues1) # выполнение запроса UPDATE  
user\_update2 = users.update\_one(myquery2, newvalues2)  
cursor1 = db.users.find() # курсор READ  
for doc in cursor1:  
 print(doc) # порядок чтения результатов запроса

Запрос DELETE:

deluser = {"Login": "andruha2"}  
result = users.delete\_many(deluser) # выполнение запроса DELETE  
cursor1 = db.users.find() # курсор READ  
for doc in cursor1:  
 print(doc) # порядок чтения результатов запроса

Получение и запись таблиц:

cursor1 = db1.users.find() # курсор READ  
cursor2 = db1.orders.find() # курсор READ  
cursor3 = db2.users.find() # курсор READ  
for doc in cursor1:  
 user1.append(doc) # порядок чтения результатов запроса  
for doc in cursor2:  
 order1.append(doc) # порядок чтения результатов запроса  
for doc in cursor3:  
 user2.append(doc) # порядок чтения результатов запроса  
return user1, order1, user2

Очистка курсора и отключение от базы данных:

del cursor1 # очистка курсора cursor  
del cursor2 # очистка курсора cursor  
del cursor3 # очистка курсора cursor  
client.close() # отключение

Для работы с графовой базой данных Neo4j на языке Python используется библиотека neo4j.

Установка:

Установите драйвер Neo4j Python с помощью pip:

$ pip install neo4j

Импорт библиотеки для взаимодействия с Neo4j в Python:

from neo4j import GraphDatabase #Графовая БД

Подключение к базе данных:

Для подключения к базе данных нужно создать объект Driver и указать URL-адрес и токен аутентификации. Подключение к базе данных Neo4j:

URI = "neo4j+s://b66822b1.databases.neo4j.io"  
AUTH = ("neo4j", "7V5beDyG9TAytJMkwHLCSP2wugLfjpYyARAShHHJhN4")  
with GraphDatabase.driver(URI, auth=AUTH) as driver:  
 driver.verify\_connectivity() # подключение

Запрос к базе данных:

Выполнить инструкцию Cypher с помощью драйвера метода.execute\_query(). Использовать заполнители и указывать параметры в качестве аргументов ключевого слова.

Для запроса CREATE в Neo4j используется ключевые слова CREATE и MERGE(чтобы не создавать дубликаты):

summary = driver.execute\_query(  
 "MERGE (u:User {IdUser: $IdUser, Name: $Name, Mail: $Mail, Surname: $Surname, Login: $Login, Password: $Password, Phone: $Phone})",  
 IdUser=3,  
 Name='Vasiliy',  
 Mail='vasya3@gmail.com',  
 Surname='Kotov',  
 Login='user1',  
 Password='39757123',  
 Phone=89017482454,  
 database\_="neo4j",  
).summary # выполнение запроса CREATE  
summary = driver.execute\_query(  
 "MERGE (u:User {IdUser: $IdUser, Name: $Name, Mail: $Mail, Surname: $Surname, Login: $Login, Password: $Password, Phone: $Phone})",  
 IdUser=4,  
 Name='Alexandr',  
 Mail='alex3@mail.ru',  
 Surname='Frolov',  
 Login='user2',  
 Password='48725075',  
 Phone=89038127430,  
 database\_="neo4j",  
).summary  
summary = driver.execute\_query(  
 "MERGE (u:User {IdUser: $IdUser, Name: $Name, Mail: $Mail, Surname: $Surname, Login: $Login, Password: $Password, Phone: $Phone})",  
 IdUser=5,  
 Name='Andrey',  
 Mail='andruha@gmail.com',  
 Surname='Uralov',  
 Login='andruha2',  
 Password='63289136',  
 Phone=89048623806,  
 database\_="neo4j",  
).summary  
summary = driver.execute\_query(  
 "MERGE (o:Order {Idorder: $Idorder, IdUserA: $IdUserA, Price: $Price, Condition: $Condition})",  
 Idorder=1,  
 IdUserA=3,  
 Price=450,  
 Condition='in road',  
 database\_="neo4j",  
).summary  
summary = driver.execute\_query(  
 "MERGE (o:Order {Idorder: $Idorder, IdUserA: $IdUserA, Price: $Price, Condition: $Condition})",  
 Idorder=2,  
 IdUserA=4,  
 Price=560,  
 Condition='in the assembly',  
 database\_="neo4j",  
).summary

Чтобы создать отношение между уже существующими узлами, выбираются эти узлы с помощью MATCH и условия WHERE и, с помощью MERGE, создаётся связь:

summary = driver.execute\_query("""  
 MATCH (u:User), (o:Order)  
 WHERE u.Name=$Name AND o.Idorder=$Idorder MERGE (u)-[h:HAVE {Account\_number: 1}]->(o)  
 """, Name="Vasiliy", Idorder=1,  
 database\_="neo4j",  
)  
summary = driver.execute\_query("""  
 MATCH (u:User), (o:Order)  
 WHERE u.Name=$Name AND o.Idorder=$Idorder MERGE (u)-[h:HAVE {Account\_number: 1}]->(o)  
 """, Name="Alexandr", Idorder=2,  
 database\_="neo4j",  
)

Для запроса READ и вывода данных в Python также необходимо использовать MATCH, после чего записанные в переменную records узлы выводятся:

records, summary, keys = driver.execute\_query(  
 "MATCH (u:User)-[h:HAVE]->(o:Order) RETURN u,h,o",  
 database\_="neo4j",  
) # READ  
for record in records:  
 print(record.data()) # порядок чтения результатов запроса

Для запроса UPDATE также необходимо использовать MATCH на нужные узлы, после чего, с помощью SET, пишутся данные для обновления:

summary = driver.execute\_query("""  
 MATCH (u:User {Login: $Login})  
 SET u.Login='vk111'  
 """, Login="user1",  
 database\_="neo4j",  
) # UPDATE  
summary = driver.execute\_query("""  
 MATCH (u:User {Login: $Login})  
 SET u.Login='Alex1'  
 """, Login="user2",  
 database\_="neo4j",  
)  
records2, summary2, keys2 = driver.execute\_query(  
 "MATCH (u:User) RETURN u",  
 database\_="neo4j",  
) # READ  
for record in records2:  
 print(record.data()) # порядок чтения результатов запроса

Для запроса DELETE также необходимо использовать MATCH на нужные узлы, после чего, с помощью DELETE, пишутся данные, которые будут удалены:

summary = driver.execute\_query("""  
 MATCH (u:User {Login: $Login})  
 DELETE u  
 """, Login="andruha2",  
 database\_="neo4j",  
)  
records2, summary2, keys2 = driver.execute\_query(  
 "MATCH (u:User) RETURN u",  
 database\_="neo4j",  
) # READ  
for record in records2:  
 print(record.data()) # порядок чтения результатов запроса

Получение и запись таблиц:

users, summary, keys = driver.execute\_query(  
 "MATCH (u:User) RETURN u.IdUser, u.Name, u.Mail, u.Surname, u.Login, u.Password, u.Phone",  
 database\_="neo4j",  
) # READ  
orders, summary, keys = driver.execute\_query(  
 "MATCH (o:Order) RETURN o.Idorder, o.IdUserA, o.Price, o.Condition",  
 database\_="neo4j",  
) # READ  
return users, orders

Закрыть соединения:

Если они не были созданы с помощью инструкции with, вызывается метод .close() для всех экземпляров драйвера, чтобы освободить все ресурсы, которые все еще удерживаются ими.

driver.close() # отключение

Так как каждая из этих баз данных выводит возвращает данные в виде своих структур, необходимо все эти структуры привести в общую форму. Преобразование данных их разнородных баз данных к единой форме и их вывод:

PSGusers, PSGorders = PosgreSQL()  
MGusers, MGorders, MGusers2 = MongoDB()  
Neousers, Neoorders = Neo4j()  
Allusers = []  
Allorders = []  
user = []  
order = []  
for i in PSGusers:  
 for j in i:  
 user.append(j)  
 Allusers.append(user)  
 user = []  
for i in PSGorders:  
 for j in i:  
 order.append(j)  
 Allorders.append(order)  
 order = []  
for i in MGusers:  
 for j in i:  
 if j == '\_id':  
 user.append(str(i[j]))  
 else:  
 user.append(i[j])  
 Allusers.append(user)  
 user = []  
for i in MGorders:  
 for j in i:  
 if j == '\_id' or j == '':  
 order.append(str(i[j]))  
 else:  
 order.append(i[j])  
 Allorders.append(order)  
 order = []  
for i in MGusers2:  
 for j in i:  
 if j == '\_id':  
 n = str(i[j])  
 if isinstance(i[j],list):  
 for k in list(i[j][0]):  
 if k == 'IdUserA':  
 order.append(n)  
 #order.append(i[j][0][k])  
 else:  
 order.append(i[j][0][k])  
 else:  
 if j == '\_id':  
 user.append(str(i[j]))  
 else:  
 user.append(i[j])  
 Allusers.append(user)  
 Allorders.append(order)  
 user = []  
 order = []  
for i in Neousers:  
 for j in i:  
 user.append(j)  
 Allusers.append(user)  
 user = []  
for i in Neoorders:  
 for j in i:  
 order.append(j)  
 Allorders.append(order)  
 order = []  
print(Allusers)  
print(Allorders)

# ЗАКЛЮЧЕНИЕ

Было сделано теоретическое и практическое исследование технологий взаимодействия с мультимодельной базой данных на примере СУБД PosgreSQL, Mongo, SQLite, Neo4j. В результате анализа был разработан и реализован алгоритм для взаимодействия с разными БД и сделаны выводы, которые в дальнейшем планируется использовать в ВКРМ.

# СПИСОК ЛИТЕРАТУРЫ

1. Виноградова М. В. «Постреляционные базы данных» Подготовительный материал. МГТУ им. Н.Э. Баумана, 2023.
2. Виноградова М.В., Крутов Т.Ю., Макаров Д.А., Волков А.С. Методические указания к лабораторным работам по курсу «Постреляционные базы данных». Москва: Под редакцией к.т.н. доц. Виноградовой М.В., 2022.
3. Официальный сайт библиотеки pymongo. URL: <https://pymongo.readthedocs.io/en/stable/> (дата обращения: 14.05.2024)
4. Официальный сайт библиотеки Neo4j. URL: <https://neo4j.com/docs/python-manual/current/> (дата обращения: 25.12.2022)
5. Официальный сайт библиотеки psycopg. URL: <https://www.psycopg.org/docs/> (дата обращения: 25.12.2022)