

Департамент образования и науки города Москвы  
Государственное автономное образовательное учреждение высшего  
образования города Москвы  
«Московский городской педагогический университет»  
Институт цифрового образования  
Департамент информатики, управления и технологий

**ДИСЦИПЛИНА:**

Распределенные системы

**Лабораторная работа 4.1**

**Сравнение подходов хранения больших данных**

Выполнил(а): Ванярина Ю. А., группа: АДЭУ-211

Преподаватель: Босенко Т.М.

Москва

2024

**Цель работы:** сравнить производительность и эффективность различных подходов к хранению и обработке больших данных на примере реляционной базы данных PostgreSQL и документо- ориентированной базы данных MongoDB.

### **Оборудование и программное обеспечение**

- Компьютер с операционной системой Ubuntu.
- PostgreSQL.
- MongoDB.
- Python 3.x.
- Библиотеки: psycpg2, pymongo, pandas, matplotlib.

### **Теоретическая часть**

В современном мире объемы данных растут экспоненциально, что приводит к необходимости использования эффективных методов их хранения и обработки. Существует два основных подхода к хранению больших данных:

1. Реляционные базы данных (например, PostgreSQL)
2. NoSQL базы данных (например, MongoDB)

Каждый из этих подходов имеет свои преимущества и недостатки, которые мы рассмотрим в ходе выполнения лабораторной работы.

### **Ход работы.**

1. Подключились к виртуальной машине через удаленный доступ с помощью xrdp, предварительно узнав ip adress виртуальной машины и добавив порт 3389 к подключению
2. Следующим шагом были запущены контейнеры с помощью команды `sudo docker compose up -d`

The screenshot shows the Visual Studio Code editor with a file named `docker-compose.yml` open. The file contains the following configuration:

```
1 services:
17   mongo-express:
20     environment:
26   ports:
27     - "8081:8081"
28   depends_on:
29     - mongodb
30   volumes:
31     - mongo_express_volume:/data/db
32   networks:
33     - backend
34
35 # PostgreSQL Service Vanyarina
36 postgres-compose:
37   container_name: postgres_container
38   image: postgres
39   environment:
40     POSTGRES_USER: ${POSTGRES_USER:-postgres}
41     POSTGRES_PASSWORD: ${POSTGRES_PASSWORD:-changeme}
42   ports:
43     - "${POSTGRES_PORT:-5432}:5432"
44   volumes:
45     - ./postgresql:/var/lib/postgresql/data
46   networks:
47     - backend
48   restart: unless-stopped
```

The terminal window at the bottom shows the command `sudo docker ps -a` being executed, displaying a list of running and stopped containers:

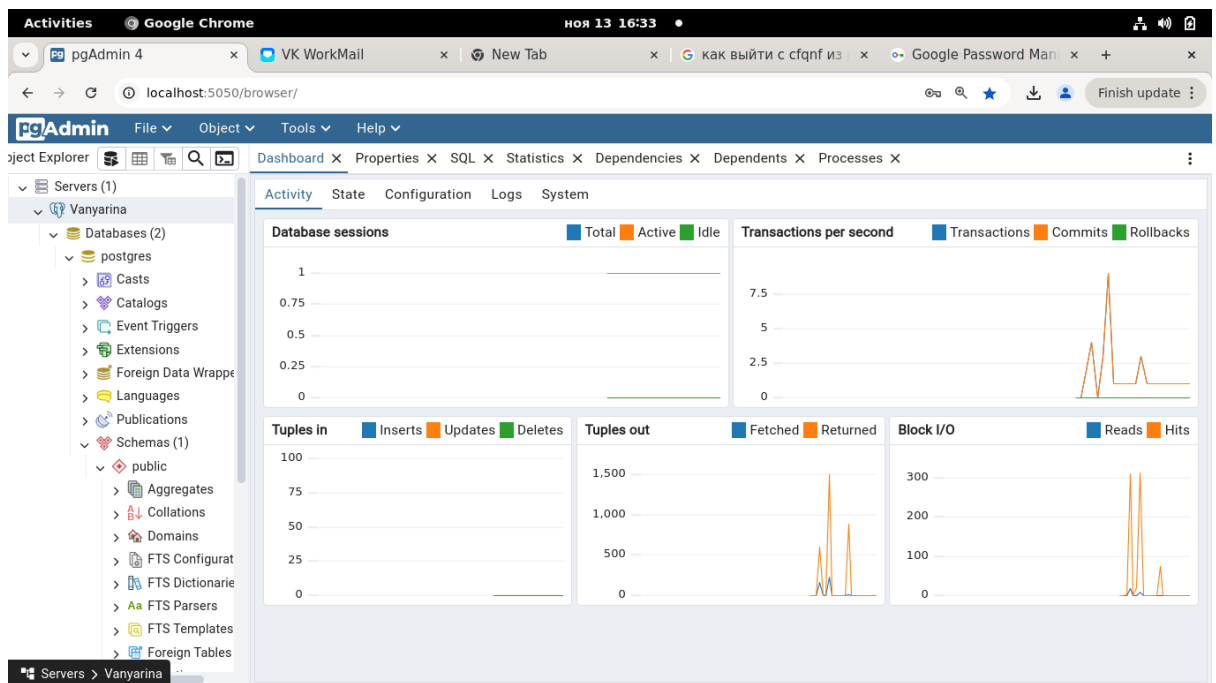
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
ATUS	PORTS		NAMES	
5be394b6aa29	dpape/pgadmin4	"/entrypoint.sh"	24 seconds ago	Up
20 seconds	443/tcp, 0.0.0.0:5050->80/tcp, [::]:5050->80/tcp	pgadmin_container		
9780dd518a5b	mongo-express:latest	"/sbin/tini -- /dock..."	24 seconds ago	Up
20 seconds	0.0.0.0:8081->8081/tcp, ::8081->8081/tcp	mongo-express		
98c919a21aba	postgres	"docker-entrypoint.s..."	24 seconds ago	Up
22 seconds	0.0.0.0:5432->5432/tcp, ::5432->5432/tcp	postgres_containe		
r				
b2c376747583	mongo:latest	"docker-entrypoint.s..."	24 seconds ago	Up

3. для подключения узнаем ip adress нужного нам контейнера с помощью команды `sudo docker inspect [id_container]`

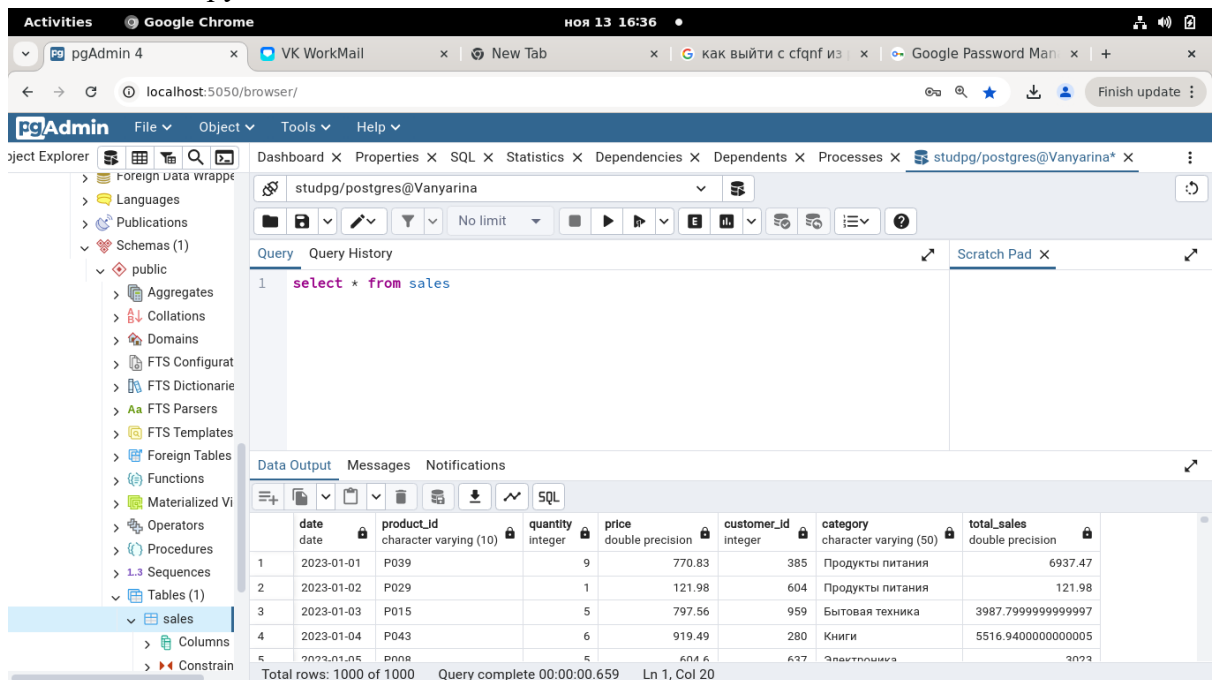
The terminal window shows the output of the command `sudo docker inspect [id_container]`, displaying network configuration details for a container:

```
"NetworkID": "acc96936b800a48fda5a82f8d311a06f268d1d511f8faf87d3d103e5013d35fd",
"EndpointID": "4bad16a33bb9ea3ff57149b04375e1e861bf81c983842f8d75d3b75bc7f45dfa",
"Gateway": "172.19.0.1",
"IPAddress": "172.19.0.4",
"IPPrefixLen": 16,
"IPv6Gateway": "",
"GlobalIPv6Address": "",
"GlobalIPv6PrefixLen": 0,
"DNSNames": [
```

4. было выполнено подключение к pgadmin и добавлен сервер Vanyarina



5. далее была загружена таблица sales



## Индивидуальное задание вариант 16

Вариант 16. Исследовать возможности и производительность при работе с JSON-данными на примере системы сбора и анализа отзывов клиентов.

1. В качестве тестовых данных были сгенерированы данные об отзывах клиентов

The screenshot shows a JupyterLab window with a file named 'lw\_4.ipynb'. The code in the cell is as follows:

```
# Подключение к MongoDB
mongo_client = MongoClient('mongodb://localhost:27017/')
mongo_db = mongo_client['studmongo']

[36]: # 16. Данные отзывов клиентов Vanyarina
def generate_customer_reviews(n_records):
    reviews = []
    for _ in range(n_records):
        review = {
            "review_id": fake.uuid4(),
            "product_id": fake.uuid4(),
            "user_id": fake.uuid4(),
            "rating": random.randint(1, 5),
            "comment": fake.text(),
            "date": fake.date_time_this_year(),
            "helpful_votes": random.randint(0, 100)
        }
        reviews.append(review)
    return reviews
var16_Vanyarina = generate_customer_reviews(1000)
print(var16_Vanyarina)
```

The output of the code is a large JSON array of 1000 review objects, each containing fields like 'review\_id', 'product\_id', 'user\_id', 'rating', 'comment', 'date', and 'helpful\_votes'.

2. Далее с данными было выполнено несколько операций, где записывалось время выполнения операции в postgresql и mongodb  
Подключение к postgresql и mongodb выполнено успешно

The screenshot shows a JupyterLab cell with the following code:

```
[38]: try:
    pg_conn = psycopg2.connect(
        dbname="postgres",
        user="postgres",
        password="changeme",
        host="localhost"
    )
    pg_cursor = pg_conn.cursor()
    print("Подключение к PostgreSQL выполнено успешно.")
except Exception as e:
    print("Ошибка подключения к PostgreSQL:", e)

# Подключение к MongoDB
try:
    mongo_client = MongoClient("mongodb://localhost:27017/")
    mongo_db = mongo_client['studmongo']
    mongo_collection = mongo_db['customer_reviews']
    print("Подключение к MongoDB выполнено успешно.")
except Exception as e:
    print("Ошибка подключения к MongoDB:", e)
```

The output of the code is two lines of text:

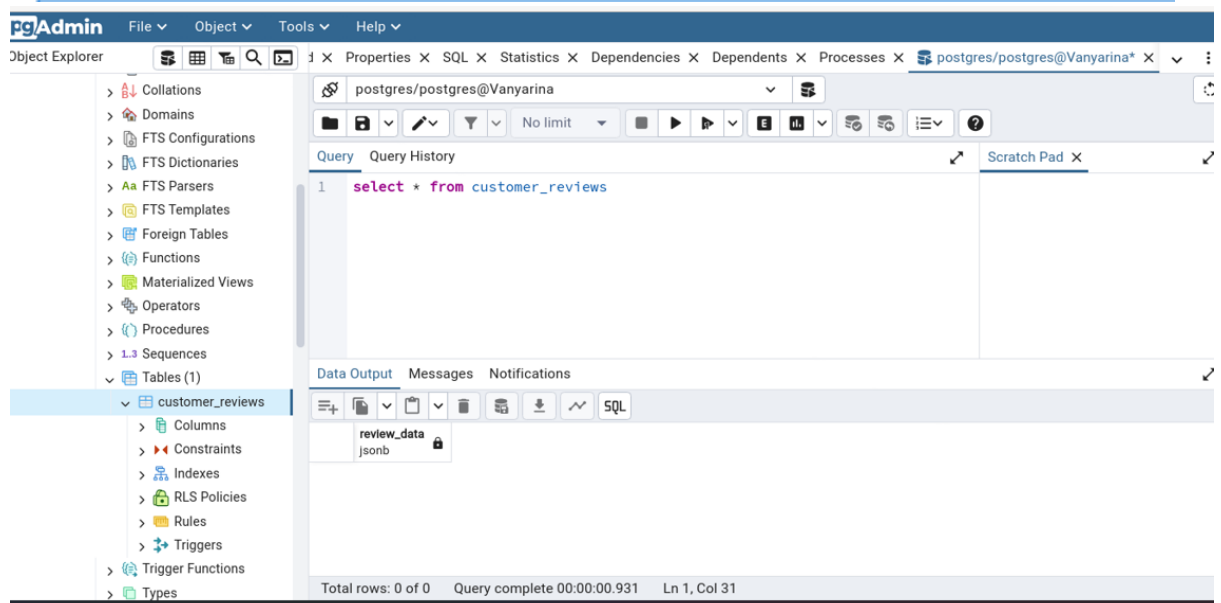
```
Подключение к PostgreSQL выполнено успешно.
Подключение к MongoDB выполнено успешно.
```

3. далее была создана таблица customer\_reviews (с указанием типа данных json)

```
# Создание таблицы
pg_cursor.execute("""
    CREATE TABLE IF NOT EXISTS customer_reviews (
        review_data JSONB
    );
""")
pg_conn.commit()
print("Таблица customer_reviews проверена или создана в PostgreSQL.")

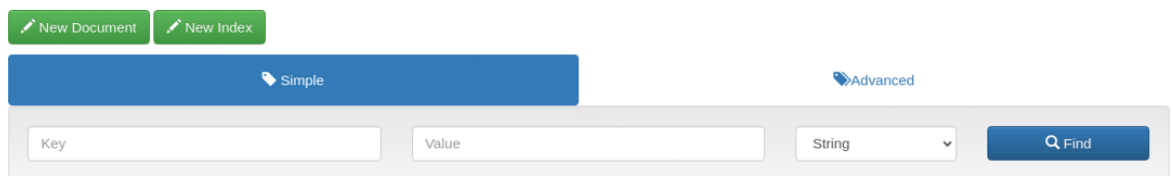
except Exception as e:
    print("Ошибка подключения к PostgreSQL:", e)
```

Подключение к PostgreSQL выполнено успешно.  
Таблица customer\_reviews проверена или создана в PostgreSQL.



также в монго создали коллекцию customer\_reviews

## Viewing Collection: customer\_reviews



- Далее были загружены данные в таблицу в postgresql  
также использована функция для сохранения времени выполнения операции  
(вставка данных)

```

5]: import json
import time

# Функция измерения времени вставки данных в PostgreSQL
def measure_pg_time():
    times = {}

    # Вставка данных в PostgreSQL
    start_time = time.time()
    try:
        for review in var16_Vanyarina:
            # Попробуем вставить данные и вывести результат вставки
            pg_cursor.execute(
                "INSERT INTO customer_reviews (review_data) VALUES (%s)",
                [json.dumps(review)]
            )
            pg_conn.commit()
            times['insert'] = time.time() - start_time
            print("Данные успешно загружены в PostgreSQL.")
    except Exception as e:
        print("Ошибка при загрузке данных в PostgreSQL:", e)
    return times

# Запуск вставки данных
measure_pg_time()

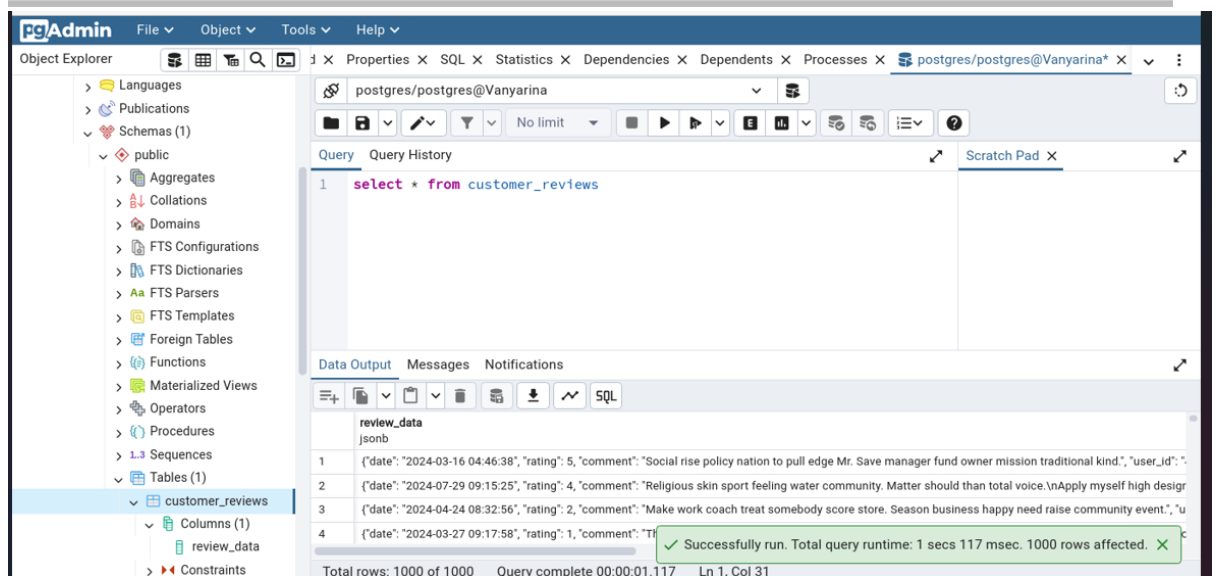
```

Данные успешно загружены в PostgreSQL.

```

5]: {'insert': 0.7614073753356934}

```



The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer displays the database structure, with the 'customer\_reviews' table selected under the 'public' schema. The main pane shows the SQL query editor with the query 'select \* from customer\_reviews'. Below the editor, the Data Output pane displays the results of the query, showing 1000 rows of data. The results are presented in a table with columns for 'review\_data' and 'jsonb'. The status bar at the bottom indicates that the query was successfully executed, with a total runtime of 1 second and 117 milliseconds, affecting 1000 rows.

Также было записано время выполнения других операций, а именно: чтение данных, подсчёт рейтинга выше 3 и среднего рейтинга

```

pg_conn.commit()
times['insert'] = time.time() - start_time
print("Данные успешно загружены в PostgreSQL.")
except Exception as e:
    print("Ошибка при загрузке данных в PostgreSQL:", e)
# Чтение данных из PostgreSQL
start_time = time.time()
pg_cursor.execute("SELECT * FROM customer_reviews")
pg_cursor.fetchall()
times['read'] = time.time() - start_time

# Подсчет отзывов с рейтингом выше 3
start_time = time.time()
pg_cursor.execute("SELECT COUNT(*) FROM customer_reviews WHERE (review_data->>'rating')::int > 3")
pg_cursor.fetchone()
times['count_high_rating'] = time.time() - start_time

# Вычисление среднего рейтинга
start_time = time.time()
pg_cursor.execute("SELECT AVG((review_data->>'rating')::int) FROM customer_reviews")
pg_cursor.fetchone()
times['average_rating'] = time.time() - start_time
return times

# Запуск вставки данных
measure_pg_time()

```

Данные успешно загружены в PostgreSQL.

```

[78]: {'insert': 0.9624035358428955,
      'read': 0.9884004592895508,
      'count_high_rating': 0.06484603881835938,
      'average_rating': 0.013821601867675781}

```

Проводим те же измерения в mongodb

```

# Функция измерения времени вставки данных в MongoDB
def measure_mongo_time():
    times = {}

    # Подключение к MongoDB
    mongo_client = MongoClient('mongodb://mongouser:mongopass@localhost:27017/')
    if check_mongo_connection(mongo_client):
        mongo_db = mongo_client['studmongo']
        collection = mongo_db['customer_reviews']

    start_time = time.time()
    try:
        records = var16_Vanyariana
        collection.insert_many(records)
        times['insert'] = time.time() - start_time
        print("Данные успешно загружены в MongoDB.")
    except Exception as e:
        print("Ошибка при загрузке данных в MongoDB:", e)

    # Чтение данных из MongoDB
    start_time = time.time()
    results_mongo = list(collection.find({}))
    # cursor = collection.find()
    # for document in cursor:
    #     pass
    times['read'] = time.time() - start_time
    return times

# Запуск вставки данных
measure_mongo_time()

```



```

# cursor = collection.find()
# for document in cursor:
#     pass
times['read'] = time.time() - start_time
# Подсчет отзывов с рейтингом выше 3
start_time = time.time()
high_rating_count = collection.count_documents({"rating": {"$gt": 3}})
times['count_high_rating'] = time.time() - start_time

# Вычисление среднего рейтинга
start_time = time.time()
avg_rating = collection.aggregate([
    {"$group": {"_id": None, "average_rating": {"$avg": "$rating"}}}]
])
list(avg_rating)
times['average_rating'] = time.time() - start_time
return times

# Запуск вставки данных
measure_mongo_time()

```

Успешное подключение к MongoDB  
Данные успешно загружены в MongoDB.

```

[124]: {'insert': 0.3009333610534668,
        'read': 0.3335292339324951,
        'count_high_rating': 0.10074496269226074,
        'average_rating': 0.009836196899414062}

```

## Viewing Collection: customer\_reviews

New Document
New Index

Simple
Advanced

Find

Delete all 1000 documents retrieved

1 2 3 4 5 > >>

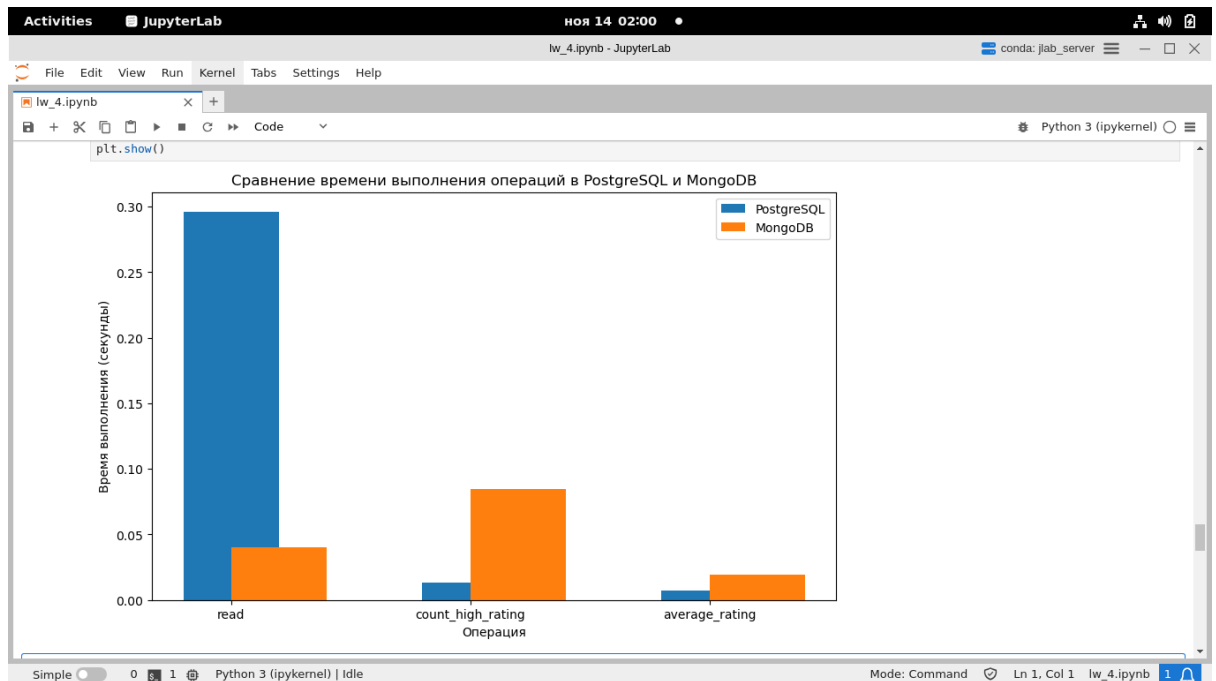
_id	review_id	product_id	user_id	rating	comment	date	helpful_votes
<a href="#">Link</a> <a href="#">Delete</a> 67351cc44ab3b1c305515714	60d2fe89-d7d5-4854-ac0c-e05ee4f14331	6208bcb7-cb0b-498f-a8a6-b561ceece8cf	47aca251-8a80-43d5-bfa9-3ce8842c672e	5	Social rise policy nation to pull edge Mr. Save m...	2024-03-16 04:46:38	20

5. Проводим измерения с полученными временными данными и строим график для наглядности

```

1]: # Проведение измерений
pg_times = measure_pg_time()
mongo_times = measure_mongo_time()

```



Выводы по работе:

1. Удобнее, проще и привычнее было работать с postgresql, на графике видно, что чтение данных было дольше в postgresql
2. Подсчёт данных с рейтингом выше трёх postgresql подсчитал за 0.013 с, что быстрее mongodb с результатом 0.1
3. Средний рейтинг postgresql так же подсчитал быстрее за 0.013 с, а mongodb за 0.09 с