

Департамент образования города Москвы

Государственное автономное образовательное учреждение  
высшего образования города Москвы  
«Московский городской педагогический университет»

Институт цифрового образования  
Департамент информатики, управления и технологий

**Лабораторная работа 2-1**  
**по дисциплине «Инструменты для хранения и обработки больших**  
**данных»**

**Тема: «Изучение методов хранения данных на основе NoSQL в**  
**MongoDB, Redis, Neo4j»**

Направление подготовки 38.03.05 – бизнес-информатика  
Профиль подготовки «Аналитика данных и эффективное управление»  
(очная форма обучения)

Выполнила:  
Студентка группы АДЭУ-211  
Белик Мария Константиновна

Преподаватель:  
Босенко Т.М.

Москва  
2024

## **ВВЕДЕНИЕ**

Цель работы: изучить и освоить методы хранения и работы с данными в NoSQL базах данных MongoDB, Redis и Neo4j. Научиться загружать данные из CSV файлов в указанные СУБД и выполнять базовые операции по работе с данными.

Оборудование и ПО:

- операционная система Ubuntu;
- установленные пакеты для работы с NoSQL базами данных: MongoDB, Redis, Neo4j;
- язык программирования Python (с библиотеками pymongo, redis, neo4j);
- CSV файлы с данными.

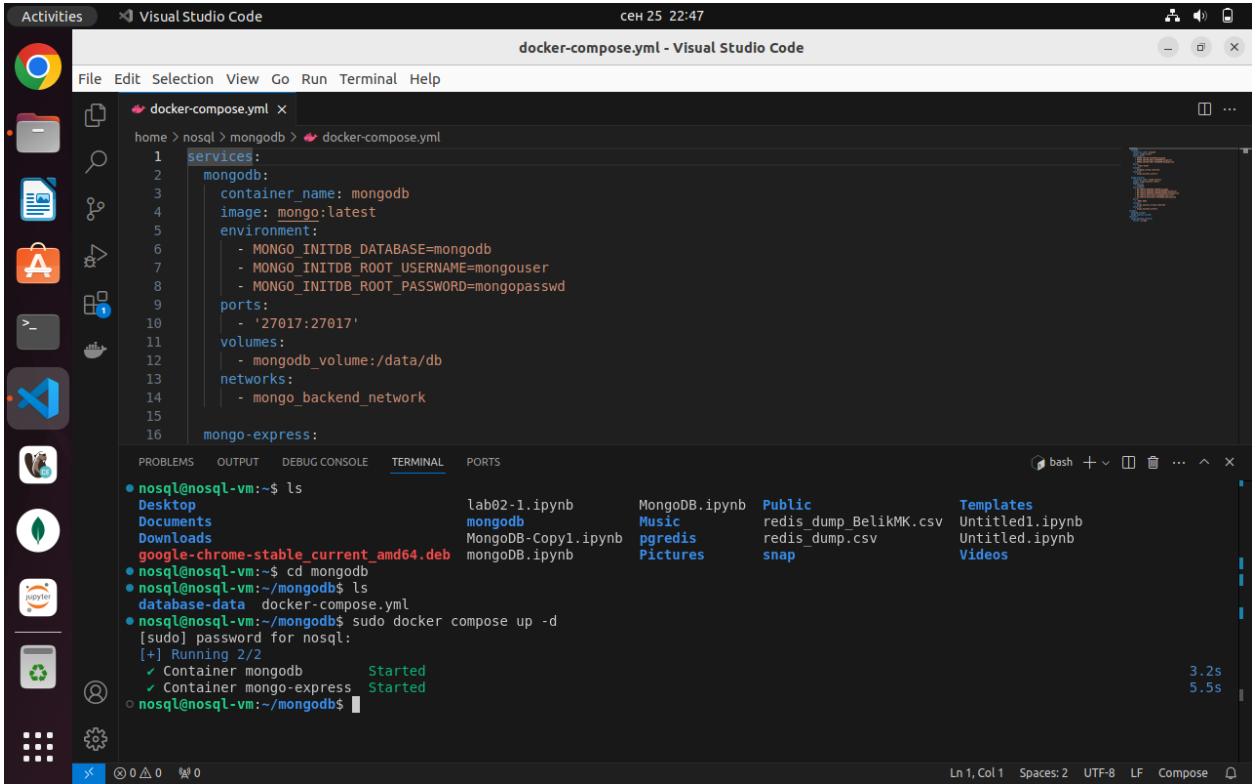
## **ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

1. MongoDB: документо-ориентированная NoSQL база данных, где данные хранятся в формате JSON-подобных документов.
2. Redis: высокопроизводительная база данных типа "ключ-значение", часто используемая для кеширования и временного хранения данных.
3. Neo4j: графовая база данных, которая позволяет хранить данные в виде вершин и рёбер графа, что удобно для моделирования сложных взаимосвязей.

# ХОД РАБОТЫ

## Шаг 1. Установка, настройка и работа в MongoDB

Открываем файл `docker-compose.yml` в приложении “Visual Studio Code”,  
после чего запускаем докер-контейнер командой `sudo docker compose up -d`:



The screenshot shows the Visual Studio Code interface. On the left is the sidebar with icons for files, folders, search, and other extensions. The main area has a tab for `docker-compose.yml`. The code content is:

```
home > nosql > mongodb > docker-compose.yml
File Edit Selection View Go Run Terminal Help
1 services:
2   mongodb:
3     container_name: mongodb
4     image: mongo:latest
5     environment:
6       - MONGO_INITDB_DATABASE=mongodb
7       - MONGO_INITDB_ROOT_USERNAME=mongouser
8       - MONGO_INITDB_ROOT_PASSWORD=mongopasswd
9     ports:
10      - '27017:27017'
11     volumes:
12       - mongodb_volume:/data/db
13     networks:
14       - mongo_backend_network
15
16 mongo-express:
```

Below the code editor is a terminal window with the following history:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● nosql@nosql-vm:~$ ls
Desktop           lab02-1.ipynb      MongoDB.ipynb  Public      Templates
Documents         mongodb          Music          redis_dump_BelikMK.csv Untitled1.ipynb
Downloads         MongoDB-Copy1.ipynb pgredis        redis_dump.csv    Untitled.ipynb
google-chrome-stable_current_amd64.deb mongoDB.ipynb Pictures      snap          Videos
● nosql@nosql-vm:~$ cd mongodb
● nosql@nosql-vm:~/mongodb$ ls
database-data  docker-compose.yml
● nosql@nosql-vm:~/mongodb$ sudo docker compose up -d
[sudo] password for nosql:
[+] Running 2/2
  ✓ Container mongodb      Started
  ✓ Container mongo-express Started
● nosql@nosql-vm:~/mongodb$
```

At the bottom of the terminal window, there are status indicators for the two containers: `Started` and `Started`, along with their respective times: `3.2s` and `5.5s`.

Рис.1 – Работа в приложении “Visual Studio Code”

После успешного запуска докер-контейнера проверяем доступ к MongoDB.

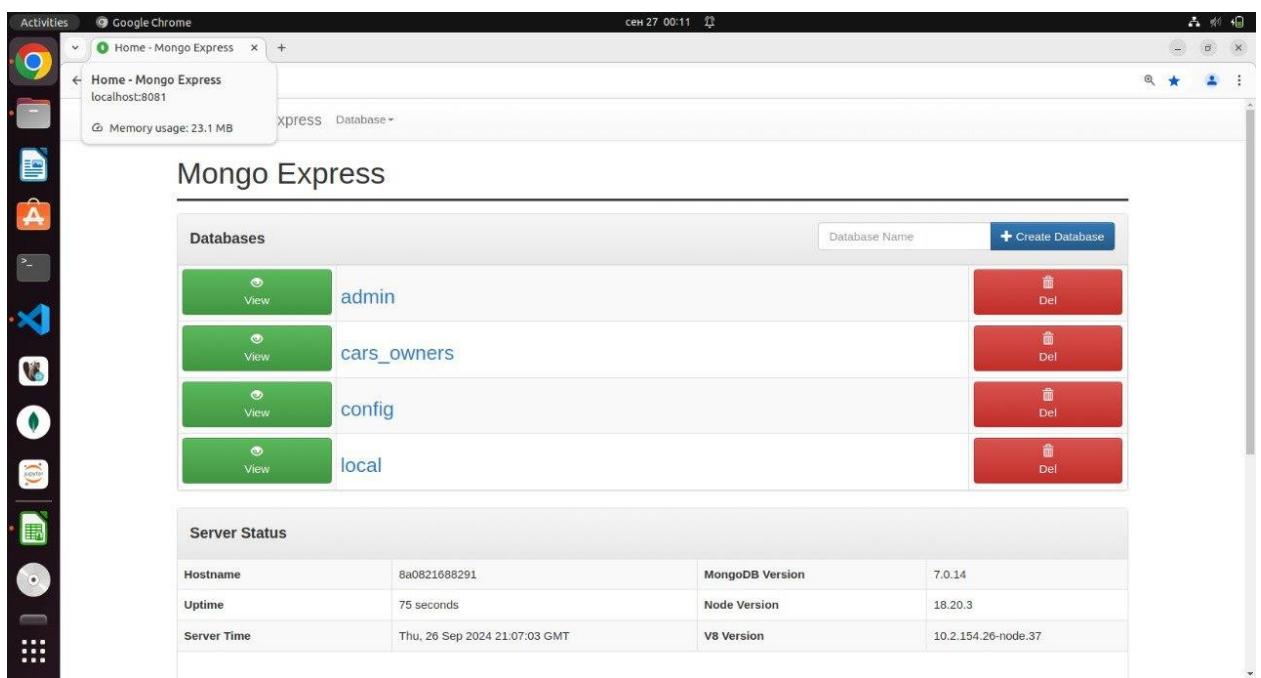


Рис.2 – Открытие Mongo Express

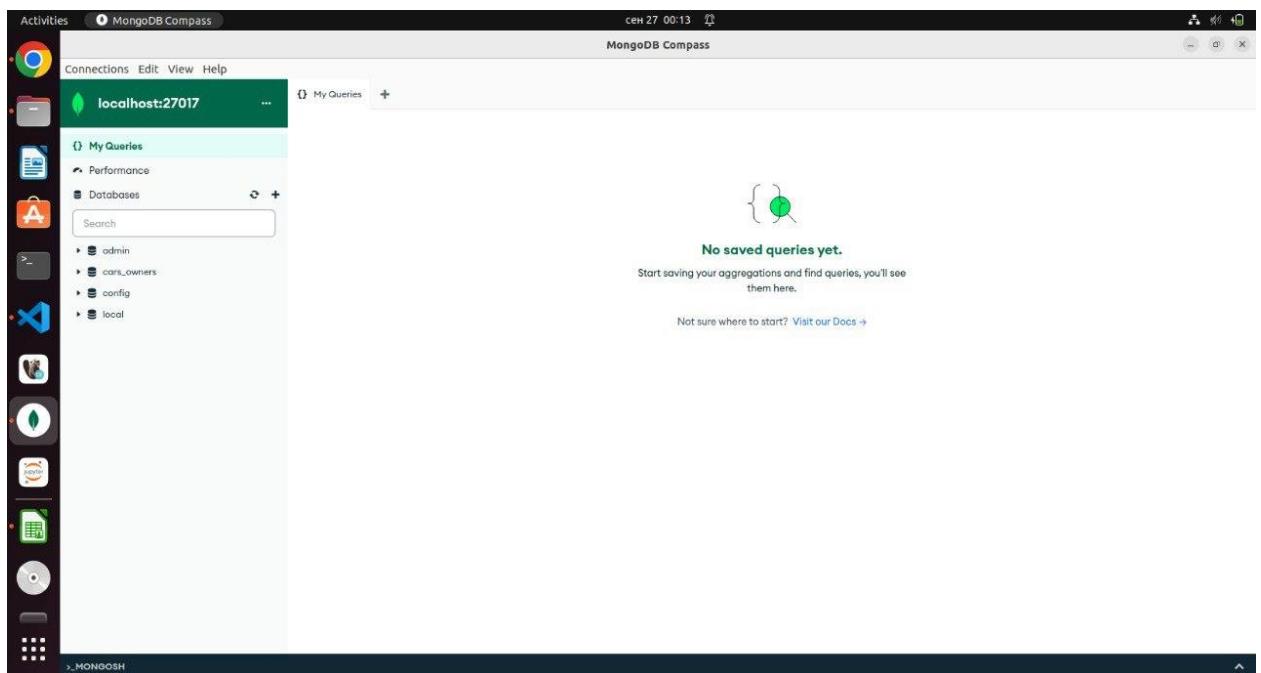


Рис.3 – Открытие MongoDB Compass

Создаем новый блокнот в приложении “Jupiter Notebook” для дальнейшей работы с *MongoDB*.

Вначале убедимся, что у нас установлена библиотека *pymongo*, используя команду `!pip install pymongo`.

После установки библиотеки подключаемся к *MongoDB*, выполняя следующий код:

```
import csv
from pymongo import MongoClient
# with docker
mongo_uri = "mongodb://mongouser:mongopasswd@localhost:27017"
try:
    # Подключение к MongoDB
    client = MongoClient(mongo_uri)
    # Проверка подключения
    client.admin.command('ping')
    print("Подключение к MongoDB установлено успешно!")
# Выбор базы данных
db = client['student']
# Выбор коллекции
labs_collection = db['lab21']
except Exception as e:
    print(f"Ошибка подключения: {e}")
```

The screenshot shows a Jupyter Notebook interface with the title 'Untitled1.ipynb'. The code cell [1] contains the command `!pip install pymongo`. The output shows that the requirement is already satisfied. Cell [2] contains the import statements and database connection setup. Cell [3] contains the connection attempt and ping command. Cell [4] contains the exception handling code. The final output shows the successful connection message: 'Подключение к MongoDB установлено успешно!'.

```
Requirement already satisfied: pymongo in ./config/jupyterlab-desktop/jlab_server/lib/python3.12/site-packages (4.8.0)
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in ./config/jupyterlab-desktop/jlab_server/lib/python3.12/site-packages (from pymongo) (2.6.1)

[2]: import csv
      from pymongo import MongoClient

[3]: # with docker
      mongo_uri = "mongodb://mongouser:mongopasswd@localhost:27017"

[4]: try:
      # Подключение к MongoDB
      client = MongoClient(mongo_uri)
      # Проверка подключения
      client.admin.command('ping')
      print("Подключение к MongoDB установлено успешно!")
# Выбор базы данных
db = client['student']
# Выбор коллекции
labs_collection = db['lab21']
except Exception as e:
    print(f"Ошибка подключения: {e}")

Подключение к MongoDB установлено успешно!
```

Рис.4 – Подключение к MongoDB

## Сгенерируем 101 строку данных о сотрудниках с помощью сервиса [TheB.AI](#) в формате:

```
{"worker_id": "01", "first_name": "Mary", "second_name": "Belik", "age": 21, "position": "manager", "company": "Disney", "hire_date": "2020-02-05"}
```

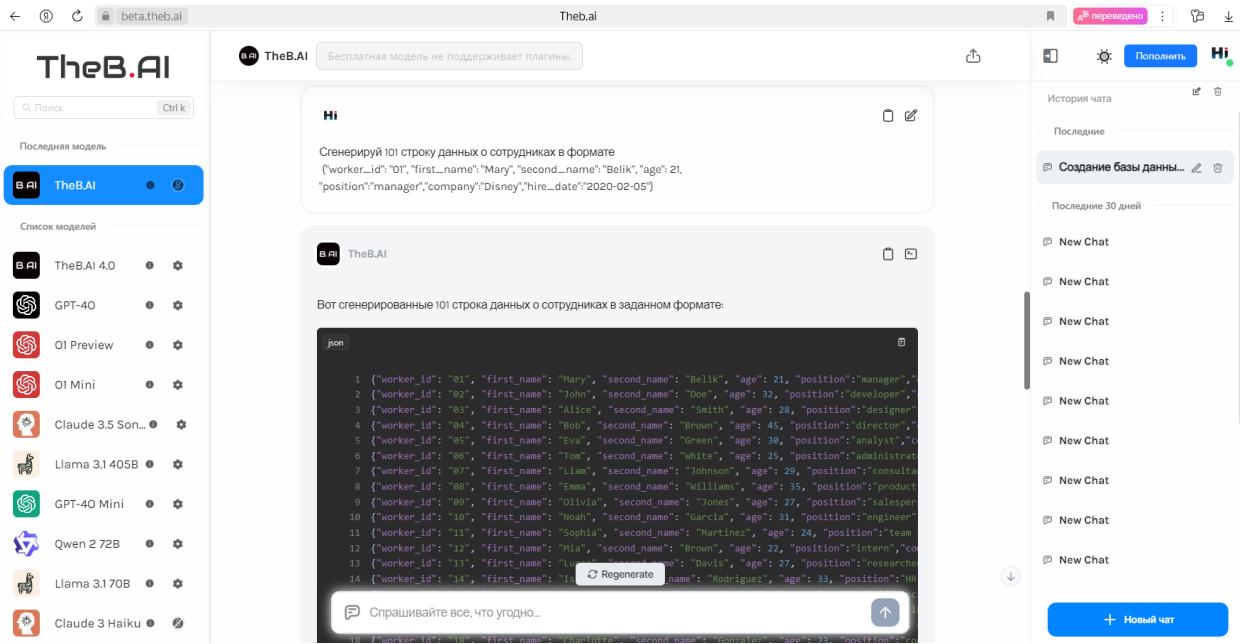


Рис.5 - Генерация данных

Загружаем сгенерированные данные в коллекцию labs21.

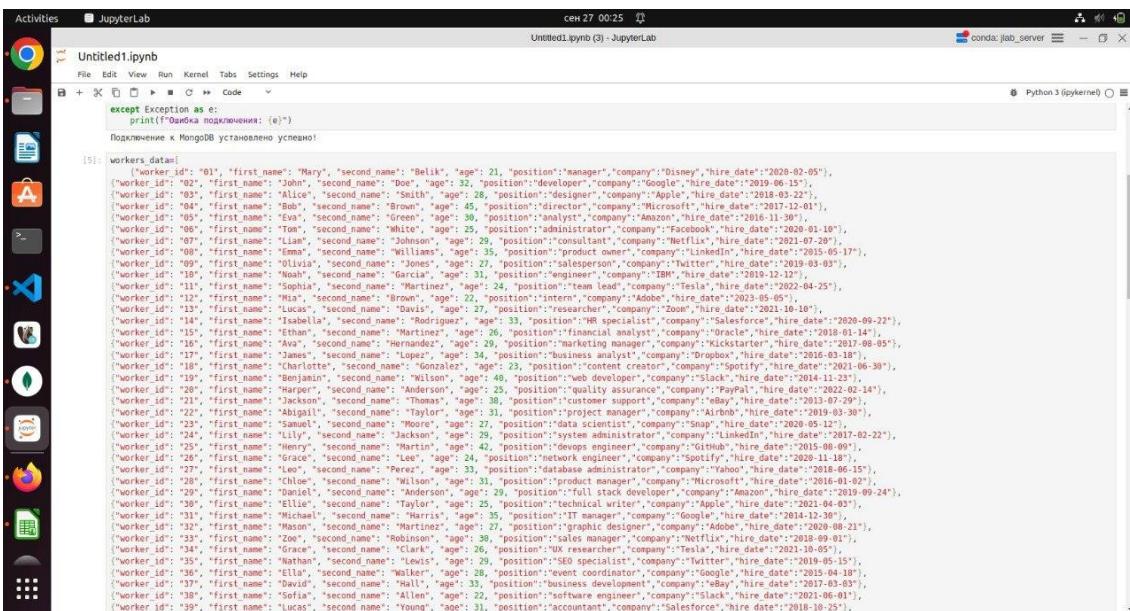


Рис.6 – Вставка данных в коллекцию

Рис.7 – Вставка данных в коллекцию

Выводим данные из коллекции, чтобы проверить, что они успешно сохранились.

Activities JupyterLab

Untitled1.ipynb

File Edit View Run Kernel Tabs Settings Help

Code 3cf6d45b0184' )

[7]: documents = labs.collection.find()  
for doc in documents:  
 print(doc)

```
{'_id': ObjectId('66f5d26471e43cf6d45b0f20'), 'worker_id': '01', 'first_name': 'Mary', 'second_name': 'Belik', 'age': 21, 'position': 'manager', 'company': 'Disney', 'hire_date': '2008-02-05'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f21'), 'worker_id': '02', 'first_name': 'John', 'second_name': 'Doe', 'age': 32, 'position': 'developer', 'company': 'Google', 'hire_date': '2019-06-15'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f22'), 'worker_id': '03', 'first_name': 'Alice', 'second_name': 'Smith', 'age': 28, 'position': 'designer', 'company': 'Apple', 'hire_date': '2018-03-22'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f23'), 'worker_id': '04', 'first_name': 'Bob', 'second_name': 'Brown', 'age': 45, 'position': 'director', 'company': 'Microsoft', 'hire_date': '2017-12-01'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f24'), 'worker_id': '05', 'first_name': 'Green', 'second_name': 'Lee', 'age': 30, 'position': 'product manager', 'company': 'Amazon', 'hire_date': '2015-01-15'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f25'), 'worker_id': '06', 'first_name': 'Tom', 'second_name': 'White', 'age': 25, 'position': 'administrator', 'company': 'Facebook', 'hire_date': '2020-01-10'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f26'), 'worker_id': '07', 'first_name': 'Liam', 'second_name': 'Johnson', 'age': 29, 'position': 'consultant', 'company': 'Netflix', 'hire_date': '2021-07-20'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f27'), 'worker_id': '08', 'first_name': 'Emma', 'second_name': 'Williams', 'age': 35, 'position': 'product owner', 'company': 'LinkedIn', 'hire_date': '2015-05-17'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f28'), 'worker_id': '09', 'first_name': 'Olivia', 'second_name': 'Jones', 'age': 27, 'position': 'salesperson', 'company': 'Twitter', 'hire_date': '2019-03-03'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f29'), 'worker_id': '10', 'first_name': 'Noah', 'second_name': 'Garcia', 'age': 31, 'position': 'engineer', 'company': 'IBM', 'hire_date': '2019-12-12'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f2a'), 'worker_id': '11', 'first_name': 'Sophia', 'second_name': 'Martinez', 'age': 24, 'position': 'team lead', 'company': 'Tesla', 'hire_date': '2022-04-25'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f2b'), 'worker_id': '12', 'first_name': 'David', 'second_name': 'Brown', 'age': 22, 'position': 'developer', 'company': 'Adobe', 'hire_date': '2023-01-01'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f2c'), 'worker_id': '13', 'first_name': 'Lucas', 'second_name': 'Davis', 'age': 23, 'position': 'researcher', 'company': 'Zoom', 'hire_date': '2021-10-10'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f2d'), 'worker_id': '14', 'first_name': 'Isabella', 'second_name': 'Rodriguez', 'age': 33, 'position': 'HR specialist', 'company': 'Salesforce', 'hire_date': '2020-09-22'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f2e'), 'worker_id': '15', 'first_name': 'Ethan', 'second_name': 'Martinez', 'age': 26, 'position': 'financial analyst', 'company': 'Oracle', 'hire_date': '2018-01-01'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f2f'), 'worker_id': '16', 'first_name': 'Ava', 'second_name': 'Hernandez', 'age': 29, 'position': 'marketing manager', 'company': 'Kickstarter', 'hire_date': '2017-08-08'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f30'), 'worker_id': '17', 'first_name': 'James', 'second_name': 'Lopez', 'age': 34, 'position': 'business analyst', 'company': 'Dropbox', 'hire_date': '2016-03-19'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f31'), 'worker_id': '18', 'first_name': 'Charlotte', 'second_name': 'Gonzalez', 'age': 23, 'position': 'content creator', 'company': 'Spotify', 'hire_date': '2021-06-30'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f32'), 'worker_id': '19', 'first_name': 'Benjamin', 'second_name': 'Wilson', 'age': 40, 'position': 'web developer', 'company': 'Slack', 'hire_date': '2014-11-23'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f33'), 'worker_id': '20', 'first_name': 'Harper', 'second_name': 'Anderson', 'age': 25, 'position': 'quality assurance', 'company': 'PayPal', 'hire_date': '2022-02-01'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f34'), 'worker_id': '21', 'first_name': 'Jackson', 'second_name': 'Thomas', 'age': 38, 'position': 'customer support', 'company': 'eBay', 'hire_date': '2013-07-29'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f35'), 'worker_id': '22', 'first_name': 'Abigail', 'second_name': 'Taylor', 'age': 31, 'position': 'project manager', 'company': 'Airbnb', 'hire_date': '2019-03-30'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f36'), 'worker_id': '23', 'first_name': 'Samuel', 'second_name': 'Moore', 'age': 27, 'position': 'data scientist', 'company': 'Snap', 'hire_date': '2020-05-12'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f37'), 'worker_id': '24', 'first_name': 'Lily', 'second_name': 'Jackson', 'age': 29, 'position': 'system administrator', 'company': 'LinkedIn', 'hire_date': '2017-02-22'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f38'), 'worker_id': '25', 'first_name': 'Henry', 'second_name': 'Martin', 'age': 42, 'position': 'devops engineer', 'company': 'GitHub', 'hire_date': '2015-08-09'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f39'), 'worker_id': '26', 'first_name': 'Grace', 'second_name': 'Lee', 'age': 24, 'position': 'network engineer', 'company': 'Spotify', 'hire_date': '2020-11-18'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f3a'), 'worker_id': '27', 'first_name': 'Leo', 'second_name': 'Perez', 'age': 33, 'position': 'database administrator', 'company': 'Yahoo', 'hire_date': '2018-06-15'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f3b'), 'worker_id': '28', 'first_name': 'Chloe', 'second_name': 'Wilson', 'age': 31, 'position': 'product manager', 'company': 'Microsoft', 'hire_date': '2016-01-02'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f3c'), 'worker_id': '29', 'first_name': 'Daniel', 'second_name': 'Anderson', 'age': 29, 'position': 'full stack developer', 'company': 'Amazon', 'hire_date': '2019-09-09'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f3d'), 'worker_id': '30', 'first_name': 'Elie', 'second_name': 'Taylor', 'age': 25, 'position': 'technical writer', 'company': 'Apple', 'hire_date': '2021-04-03'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f3e'), 'worker_id': '31', 'first_name': 'Michael', 'second_name': 'Harris', 'age': 35, 'position': 'IT manager', 'company': 'Google', 'hire_date': '2014-12-30'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f3f'), 'worker_id': '32', 'first_name': 'Mason', 'second_name': 'Martinez', 'age': 27, 'position': 'graphic designer', 'company': 'Adobe', 'hire_date': '2020-08-21'}  
{'_id': ObjectId('66f5d26471e43cf6d45b0f40'), 'worker_id': '33', 'first_name': 'Zoe', 'second_name': 'Robinson', 'age': 30, 'position': 'sales manager', 'company': 'Netflix', 'hire_date': '2018-09-01'}
```

Рис.8 – Получение данных

Также проверим, что данные сохранились, используя интерфейс MongoDB Compass.

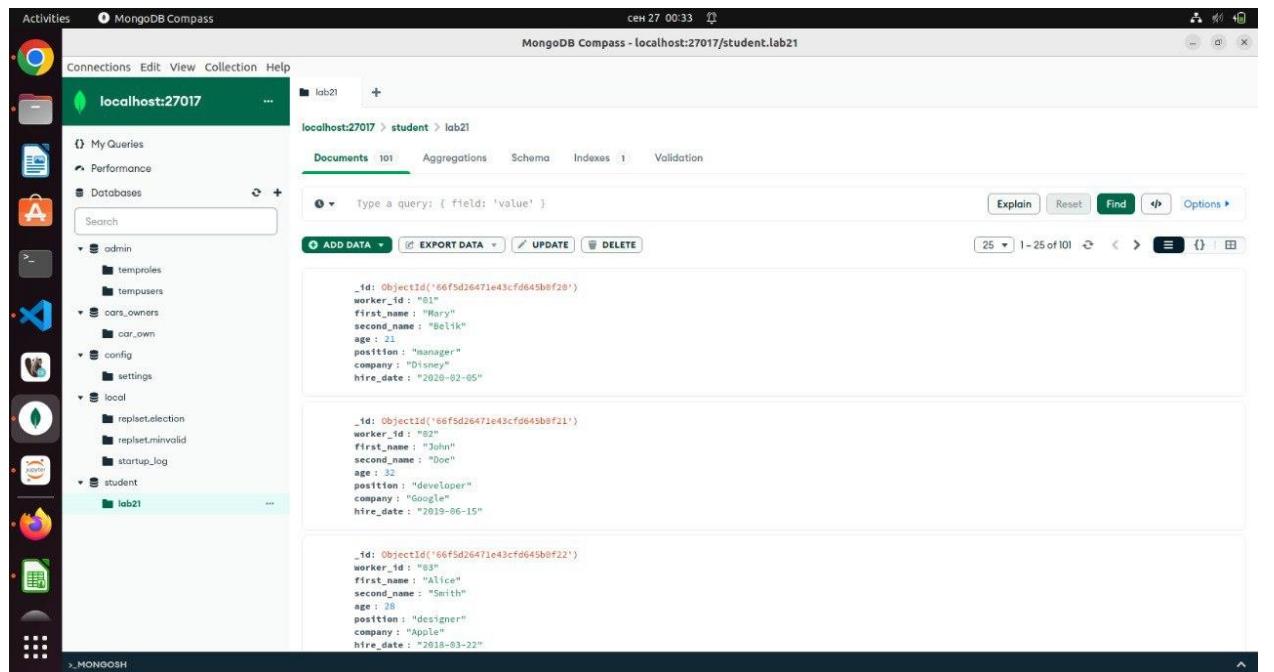


Рис.9 – Проверка в MongoDB Compass

После завершения работы с базой данных закроем соединение:

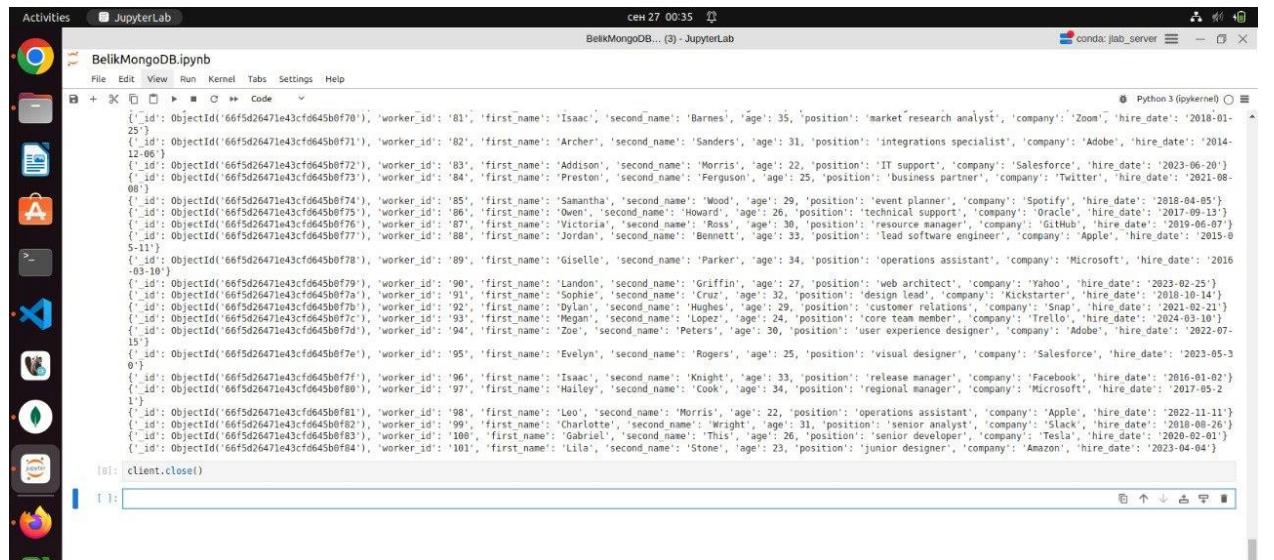
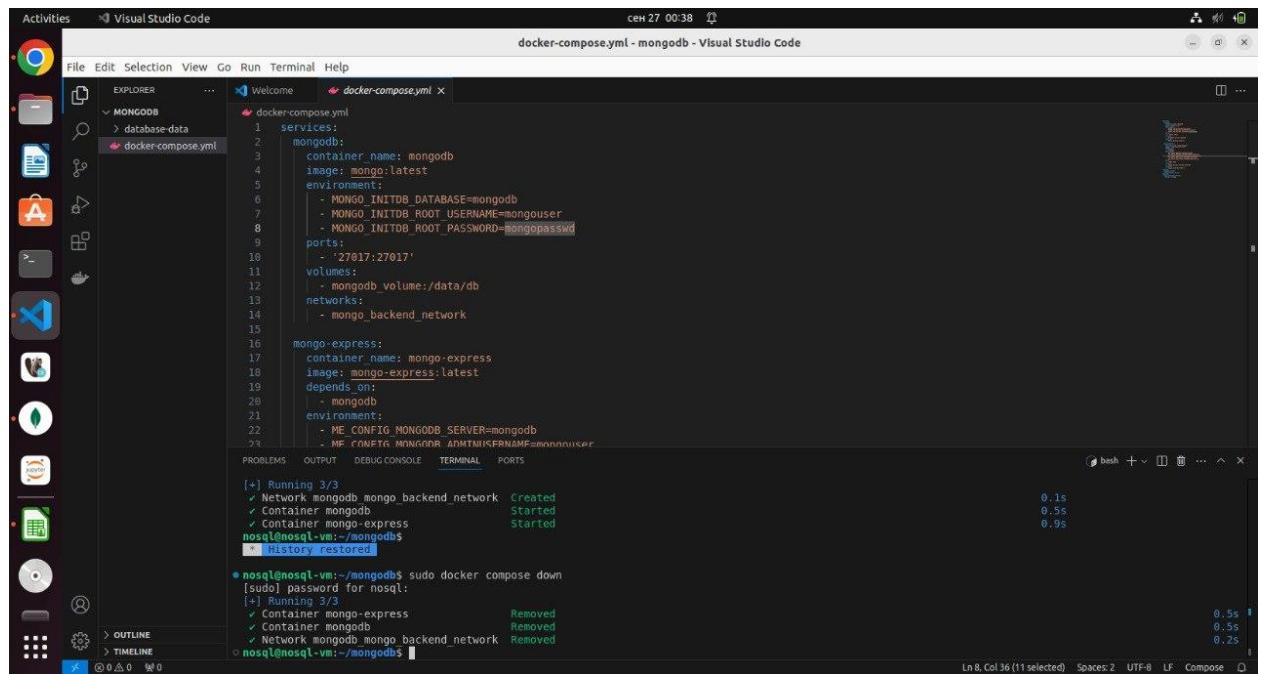


Рис.10 – Закрытие соединения

Также при завершении работы необходимо остановить докер-контейнер:



The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there is a folder named 'MONGODB' containing 'database-data' and 'docker-compose.yml'. The 'docker-compose.yml' file is open in the main editor area, displaying the following configuration:

```
version: '3.8'
services:
  mongodb:
    container_name: mongodb
    image: mongo:latest
    environment:
      - MONGO_INITDB_DATABASE=mongodb
      - MONGO_INITDB_ROOT_USERNAME=mongouser
      - MONGO_INITDB_ROOT_PASSWORD=mongopassw0rd
    ports:
      - '27017:27017'
    volumes:
      - mongodb_data:/data/db
    networks:
      - mongo_backend_network
  mongo-express:
    container_name: mongo-express
    image: mongo-express:latest
    depends_on:
      - mongodb
    environment:
      - ME_CONFIG_MONGODB_SERVER=mongodb
      - ME_CONFIG_MONGODB_ADMINUSERNAME=mongouser
      - ME_CONFIG_MONGODB_ADMINPASSWORD=mongopassw0rd
```

In the bottom right corner of the code editor, there is a terminal window showing the command 'sudo docker compose down' being run. The terminal output indicates that three containers (mongodb, mongo-express, and mongo\_backend\_network) are being stopped and removed.

Рис.11 – Остановка докер-контейнера

## Шаг 2. Установка, настройка и работа в Redis

Открываем файл *docker-compose.yml* в приложении “Visual Studio Code”, после чего запускаем докер командой `sudo docker compose up -d`:

The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar with a 'PGREDIS' folder containing 'docker-compose.yml' and 'docker-compose2.txt'. The main area has tabs for 'Welcome', 'docker-compose.yml', and 'docker-compose2.txt'. A terminal window is open with the following command and output:

```
sudo docker compose up -d
[sudo] password for nosql:
Sorry, try again.
[sudo] password for nosql:
[+] Running 4/4
  ✓ Container pgredis-redis-1      Started      2.8s
  ✓ Container pgredis-redis-commander-1 Started      6.3s
  ✓ Container pgredis-postgres-1     Started      2.6s
  ✓ Container pgredis-pgweb-1       Started      6.0s
nosql@nosql-vm:~/pgredis$
```

Рис.12 – Работа в приложении “Visual Studio Code”

Проверяем успешность запуска докер-контейнеров и подключения к *Redis* в “Google Chrome”:

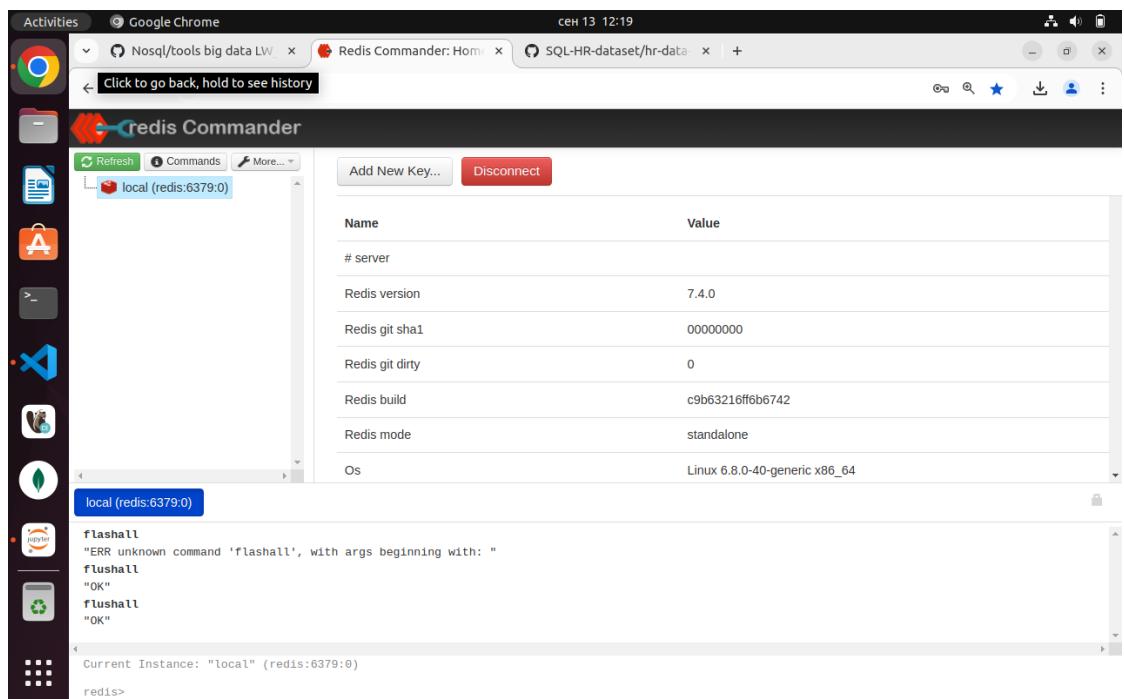


Рис.13 – Redis Commander

Создаем новый блокнот в приложении “Jupiter Notebook” для дальнейшей работы с *Redis*.

Вначале убедимся, что у нас установлена библиотека *redis*, используя команду `!pip install redis`.

После установки библиотеки подключаемся к *Redis*, выполняя следующий код:

```
import redis
import csv
import json

# Подключение к Redis с аутентификацией
r = redis.Redis(
    host='localhost',
    port=6379,
    db=0 # Подключение к базе данных 0
)
# Проверка соединения
try:
    r.ping()
    print("Соединение с Redis успешно установлено.")
except redis.ConnectionError:
    print("Не удалось подключиться к Redis.")

Соединение с Redis успешно установлено.
```

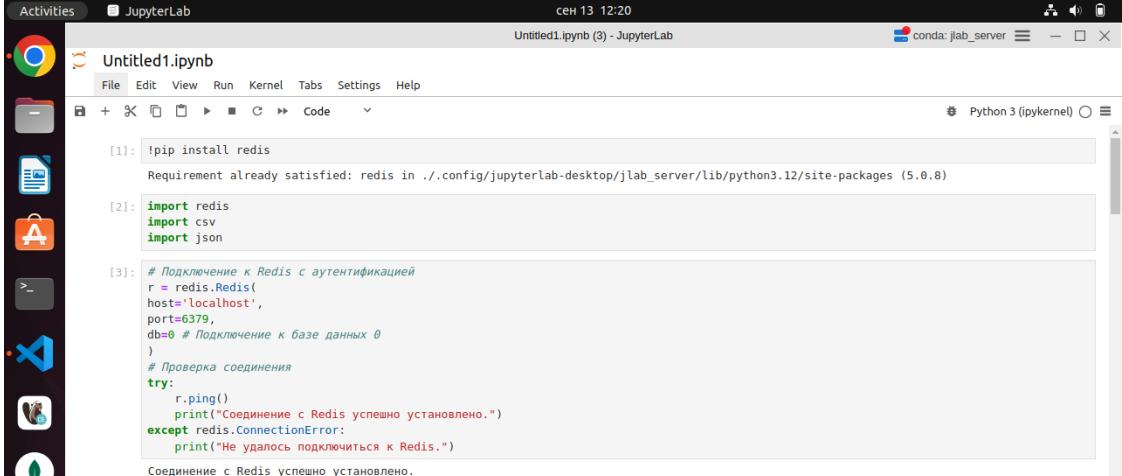


Рис.14 – Успешное соединение с Redis

Используем сервис [TheB.AI](#) для генерации 100 строк данных о сотрудниках организации в формате хэша:

```
r.hset("employee:1", mapping={"firstname":"Ariana", "lastname":"Gray",
"position":"Manager","company":"Great Company"}).
```

The screenshot shows the TheB.AI web application interface. On the left, there's a sidebar titled 'Последняя модель' (Last Model) which lists various AI models with their icons and names: TheB.AI 4.0, GPT-40, Claude 3.5 Son..., Llama 3.1 405B, GPT-40 Mini, Qwen 2.72B, Llama 3.1 70B, Claude 3 Haiku, Gemini 1.5 Pro, and Gemini 1.5 Flash. The main area has tabs for 'TheB.AI' and 'Бесплатная модель не поддерживает плагины.' (Free model does not support plugins). The current tab shows a message 'Hi' and a text input field containing: 'Сгенерируй 101 строку на английском языке в формате: r.hset("employee:1", mapping={"firstname": "Ariana", "lastname": "Gray", "position": "Manager", "company": "Great Company"})'. Below this, another tab shows the generated output: 'Sure! Here are 101 lines in the specified format:' followed by a large block of Python code. At the bottom right of the main window is a blue button labeled '+ Новый чат' (New Chat).

Рис.15 – Генерация данных

Загружаем сгенерированные данные в базу данных Redis.

The screenshot shows a JupyterLab session titled 'Untitled2.ipynb'. The code cell [15:] contains a large block of Python code using the Redis library to set 101 employee records into a Redis database. The code uses the 'r.hset' command with 'mapping' parameters to define firstnames, lastnames, positions, and companies for each employee from 1 to 33. The code is identical to what was generated in Figure 15.

```
[15]: r.hset("employee:1", mapping={"firstname": "Ariana", "lastname": "Gray", "position": "Manager", "company": "Great Company"})
r.hset("employee:2", mapping={"firstname": "Liam", "lastname": "Smith", "position": "Developer", "company": "Tech Innovations"})
r.hset("employee:3", mapping={"firstname": "Emma", "lastname": "Jones", "position": "Designer", "company": "Creative Studio"})
r.hset("employee:4", mapping={"firstname": "Noah", "lastname": "Williams", "position": "Analyst", "company": "Finance Corp"})
r.hset("employee:5", mapping={"firstname": "Olivia", "lastname": "Brown", "position": "HR Specialist", "company": "HR Solutions"})
r.hset("employee:6", mapping={"firstname": "Ethan", "lastname": "Davis", "position": "Marketing Manager", "company": "Brand Builders"})
r.hset("employee:7", mapping={"firstname": "Sophia", "lastname": "Miller", "position": "Sales Representative", "company": "Sales Force"})
r.hset("employee:8", mapping={"firstname": "Logan", "lastname": "Wilson", "position": "Product Owner", "company": "Product Co."})
r.hset("employee:9", mapping={"firstname": "Isabella", "lastname": "Moore", "position": "Content Writer", "company": "Content Hub"})
r.hset("employee:10", mapping={"firstname": "Lucas", "lastname": "Taylor", "position": "Graphic Designer", "company": "Visual Arts"})
r.hset("employee:11", mapping={"firstname": "Mia", "lastname": "Anderson", "position": "Software Engineer", "company": "CodeLabs"})
r.hset("employee:12", mapping={"firstname": "Aiden", "lastname": "Thomas", "position": "Data Scientist", "company": "Data Insights"})
r.hset("employee:13", mapping={"firstname": "Charlotte", "lastname": "Jackson", "position": "UX Designer", "company": "UX Experts"})
r.hset("employee:14", mapping={"firstname": "James", "lastname": "White", "position": "Network Engineer", "company": "Network Solutions"})
r.hset("employee:15", mapping={"firstname": "Amelia", "lastname": "Harris", "position": "Business Analyst", "company": "Strategy Partners"})
r.hset("employee:16", mapping={"firstname": "Alexander", "lastname": "Martin", "position": "Project Manager", "company": "Project Pathways"}
r.hset("employee:17", mapping={"firstname": "Harper", "lastname": "Thompson", "position": "SEO Specialist", "company": "SEO Masters"})
r.hset("employee:18", mapping={"firstname": "Henry", "lastname": "Garcia", "position": "Operations Manager", "company": "Operations Hub"})
r.hset("employee:19", mapping={"firstname": "Ella", "lastname": "Martinez", "position": "Frontend Developer", "company": "Web Builders"})
r.hset("employee:20", mapping={"firstname": "Jackson", "lastname": "Robinson", "position": "Backend Developer", "company": "Code Factory"})
r.hset("employee:21", mapping={"firstname": "Sofia", "lastname": "Clark", "position": "Customer Support", "company": "Support Service"})
r.hset("employee:22", mapping={"firstname": "Michael", "lastname": "Rodriguez", "position": "Accountant", "company": "Finance First"})
r.hset("employee:23", mapping={"firstname": "Avery", "lastname": "Lewis", "position": "Content Strategist", "company": "Content Creators"})
r.hset("employee:24", mapping={"firstname": "Benjamin", "lastname": "Lee", "position": "Software Tester", "company": "Quality Assurance"})
r.hset("employee:25", mapping={"firstname": "Scarlett", "lastname": "Walker", "position": "PR Specialist", "company": "Public Relations"})
r.hset("employee:26", mapping={"firstname": "Elijah", "lastname": "Hall", "position": "Web Developer", "company": "Web Innovations"})
r.hset("employee:27", mapping={"firstname": "Grace", "lastname": "Allen", "position": "Social Media Manager", "company": "Social Strategies"})
r.hset("employee:28", mapping={"firstname": "Owen", "lastname": "Young", "position": "Technical Writer", "company": "Write Tech"})
r.hset("employee:29", mapping={"firstname": "Chloe", "lastname": "King", "position": "Compliance Officer", "company": "Risk Management"})
r.hset("employee:30", mapping={"firstname": "Liam", "lastname": "Scott", "position": "IT Support", "company": "Tech Support"})
r.hset("employee:31", mapping={"firstname": "Madison", "lastname": "Adams", "position": "Data Analyst", "company": "Data Co."})
r.hset("employee:32", mapping={"firstname": "James", "lastname": "Baker", "position": "Operations Analyst", "company": "Operations Insight"})
r.hset("employee:33", mapping={"firstname": "Lily", "lastname": "Gonzalez", "position": "E-commerce Manager", "company": "E-com Solutions"}]
```

Рис.16 – Загрузка данных в Redis

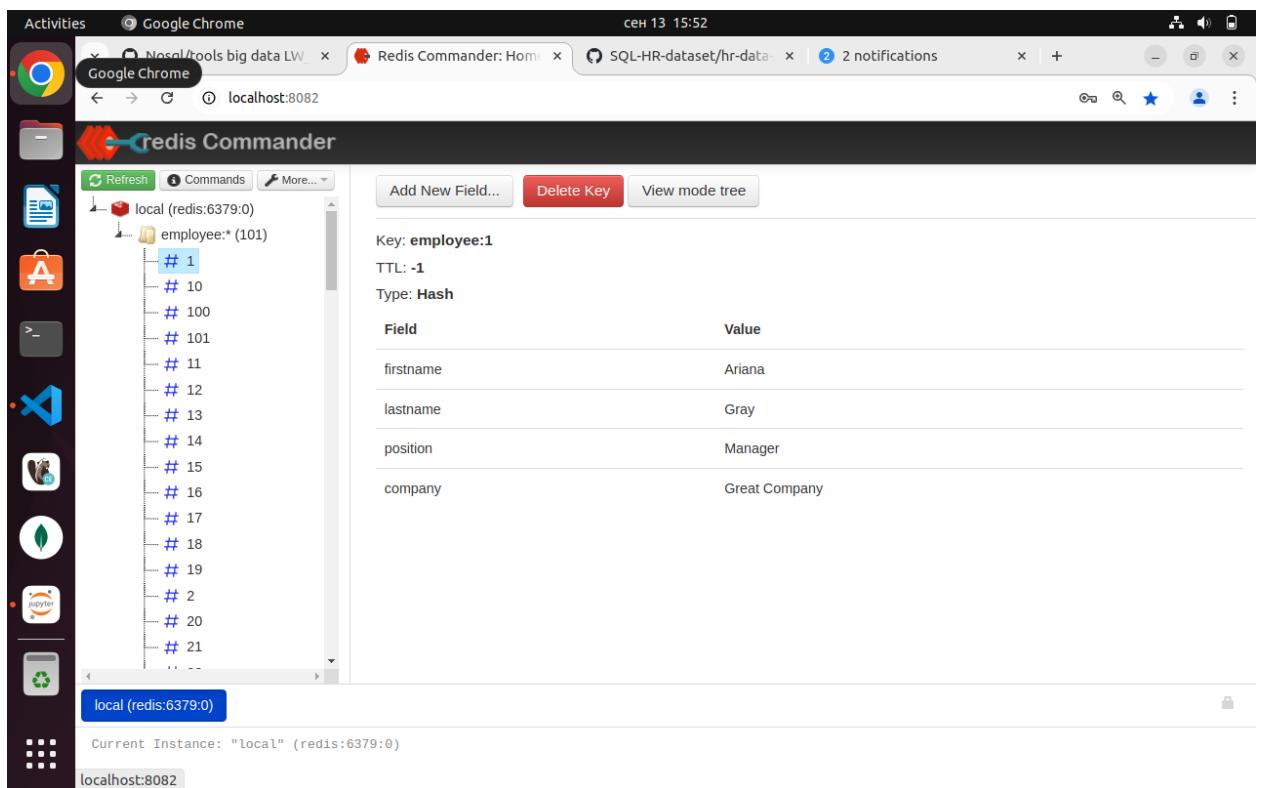


Рис.17 – Просмотр загруженных данных в Redis Commander

Получаем данные о сотрудниках под номерами 13, 68 и 92, и видим, что хэш корректно выводится.

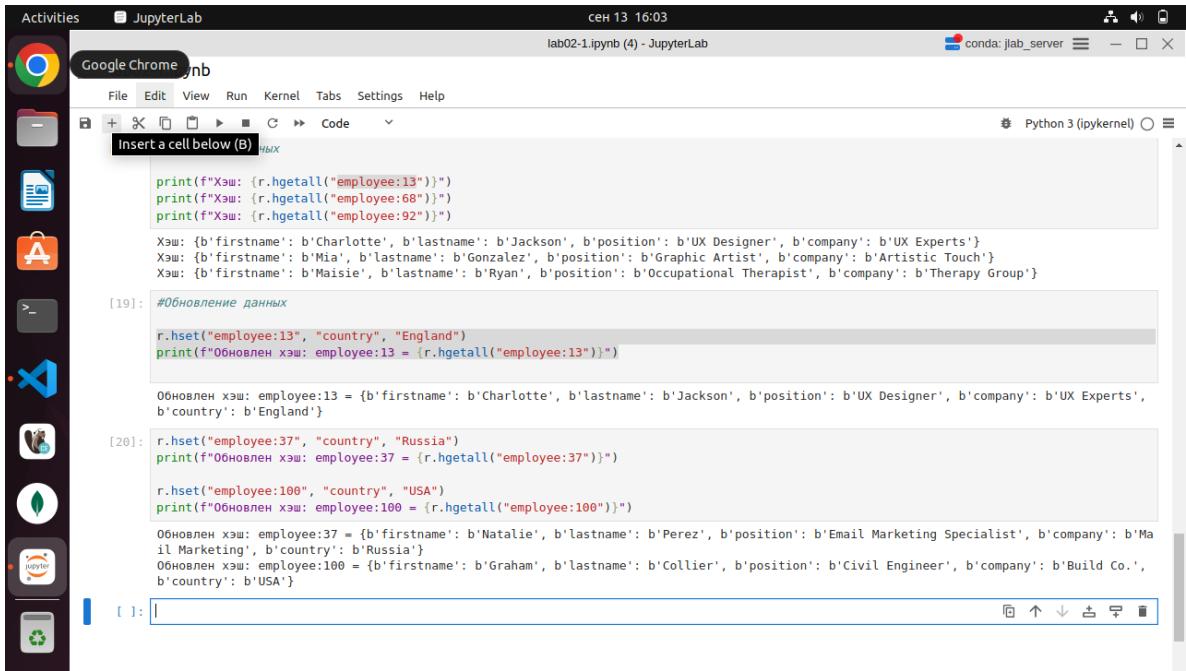
```

Activities JupyterLab
Untitled2.ipynb - JupyterLab
File Edit View Run Kernel Tabs Settings Help
Python 3 (ipykernel) 
r.hset("employee:100", mapping={"firstname": "Graham", "lastname": "Collier", "position": "Civil Engineer", "company": "Build Co."})
r.hset("employee:101", mapping={"firstname": "Zelda", "lastname": "Hunter", "position": "Photographer", "company": "Photo Studio"})
[15]: 4
[16]: #Получение данных
print(f"Хэш: {r.hgetall('employee:13')}")
print(f"Хэш: {r.hgetall('employee:68')}")
print(f"Хэш: {r.hgetall('employee:92')}")
Хэш: {'firstname': b'Charlotte', 'lastname': b'Jackson', 'position': b'UX Designer', 'company': b'UX Experts'}
Хэш: {'firstname': b'Mia', 'lastname': b'Gonzalez', 'position': b'Graphic Artist', 'company': b'Artistic Touch'}
Хэш: {'firstname': b'Maisie', 'lastname': b'Ryan', 'position': b'Occupational Therapist', 'company': b'Therapy Group'}
[1]: 

```

Рис.18 – Получение данных

Обновляем данные для сотрудников под номерами 13, 37 и 100:  
добавляем им также страну проживания и работы.



The screenshot shows a JupyterLab interface with a code cell containing Python code. The code uses Redis commands (r.hgetall and r.hset) to retrieve and update data for employees with IDs 13, 37, and 100, respectively. It also prints the updated data. The output shows the original data and the modified data with added 'country' and 'company' fields.

```
print(f"Хэш: {r.hgetall('employee:13')}")
print(f"Хэш: {r.hgetall('employee:68')}")
print(f"Хэш: {r.hgetall('employee:92')}")
Хэш: {'firstname': b'Charlotte', 'lastname': b'Jackson', 'position': b'UX Designer', 'company': b'UX Experts'}
Хэш: {'firstname': b'Mia', 'lastname': b'Gonzalez', 'position': b'Graphic Artist', 'company': b'Artistic Touch'}
Хэш: {'firstname': b'Maisie', 'lastname': b'Ryan', 'position': b'Occupational Therapist', 'company': b'Therapy Group'}
[19]: #Обновление данных
r.hset("employee:13", "country", "England")
print(f"Обновлен хэш: employee:13 = {r.hgetall('employee:13')}")
Обновлен хэш: employee:13 = {'firstname': b'Charlotte', 'lastname': b'Jackson', 'position': b'UX Designer', 'company': b'UX Experts', 'country': b'England'}
[20]: r.hset("employee:37", "country", "Russia")
print(f"Обновлен хэш: employee:37 = {r.hgetall('employee:37')}")
r.hset("employee:100", "country", "USA")
print(f"Обновлен хэш: employee:100 = {r.hgetall('employee:100')}")
Обновлен хэш: employee:37 = {'firstname': b'Natalie', 'lastname': b'Perez', 'position': b'Email Marketing Specialist', 'company': b'Ma
il Marketing', 'country': b'Russia'}
Обновлен хэш: employee:100 = {'firstname': b'Graham', 'lastname': b'Collier', 'position': b'Civil Engineer', 'company': b'Build Co.', 'country': b'USA'}
```

Рис.19 – Обновление данных

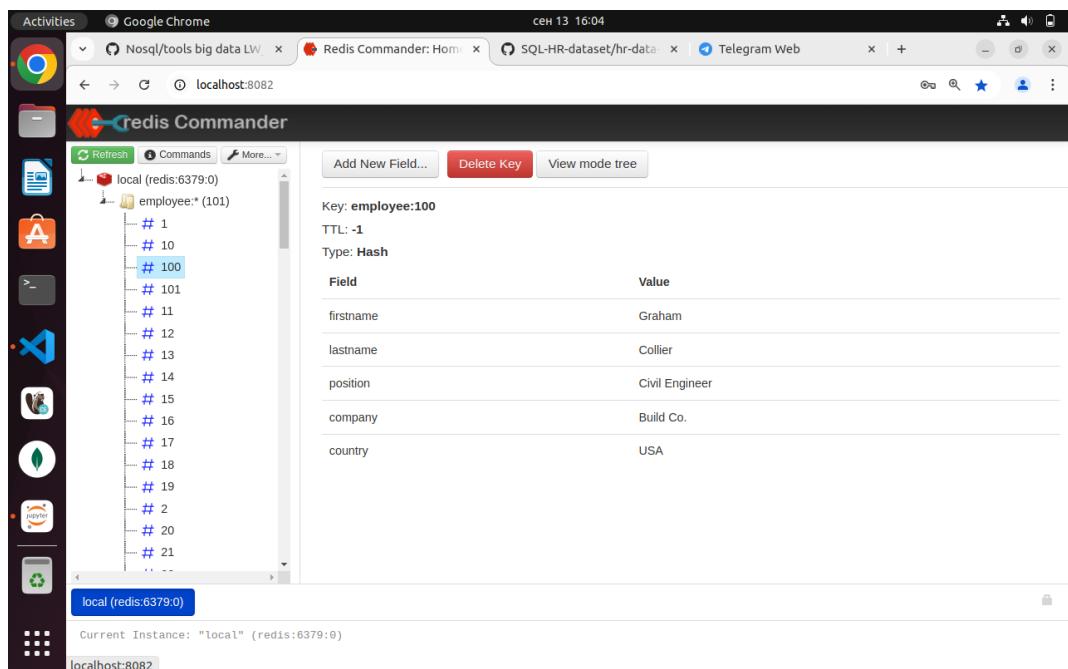


Рис.20 – Проверка успешности обновления в Redis Commander

Удаление данных о сотрудниках под номерами 2, 21, 22, 23 и 24 и проверка успешности выполненных действий.

The screenshot shows a JupyterLab environment with a sidebar containing icons for various applications like a browser, file manager, and terminal. The main area is titled 'lab02-1.ipynb' and shows Python code interacting with a Redis database. The code includes setting a value, printing it, deleting specific keys, and then checking if those keys still exist.

```
r.hset("employee:100", "country", "USA")
print(f"Обновлен хэш: employee:100 = {r.hgetall("employee:100")}")

Обновлен хэш: employee:37 = {b'firstname': b'Natalie', b'lastname': b'Perez', b'position': b'Email Marketing Specialist', b'company': b'Mail Marketing', b'country': b'Russia'}
Обновлен хэш: employee:100 = {b'firstname': b'Graham', b'lastname': b'Collier', b'position': b'Civil Engineer', b'company': b'Build Co.', b'country': b'USA'}

[21]: #Удаление данных
r.delete("employee:2", "employee:21", "employee:22", "employee:23", "employee:24")
print("удалены ключи: employee:2, employee:21, employee:22, employee:23, employee:24")

Удалены ключи: employee:2, employee:21, employee:22, employee:23, employee:24

[22]: #Проверка удаленных данных
for key in ["employee:2", "employee:21", "employee:22", "employee:23", "employee:24"]:
    print(f"Существует ли ключ {key}? {r.exists(key)}")

Существует ли ключ employee:2? 0
Существует ли ключ employee:21? 0
Существует ли ключ employee:22? 0
Существует ли ключ employee:23? 0
Существует ли ключ employee:24? 0
```

Рис.21 – Удаление данных

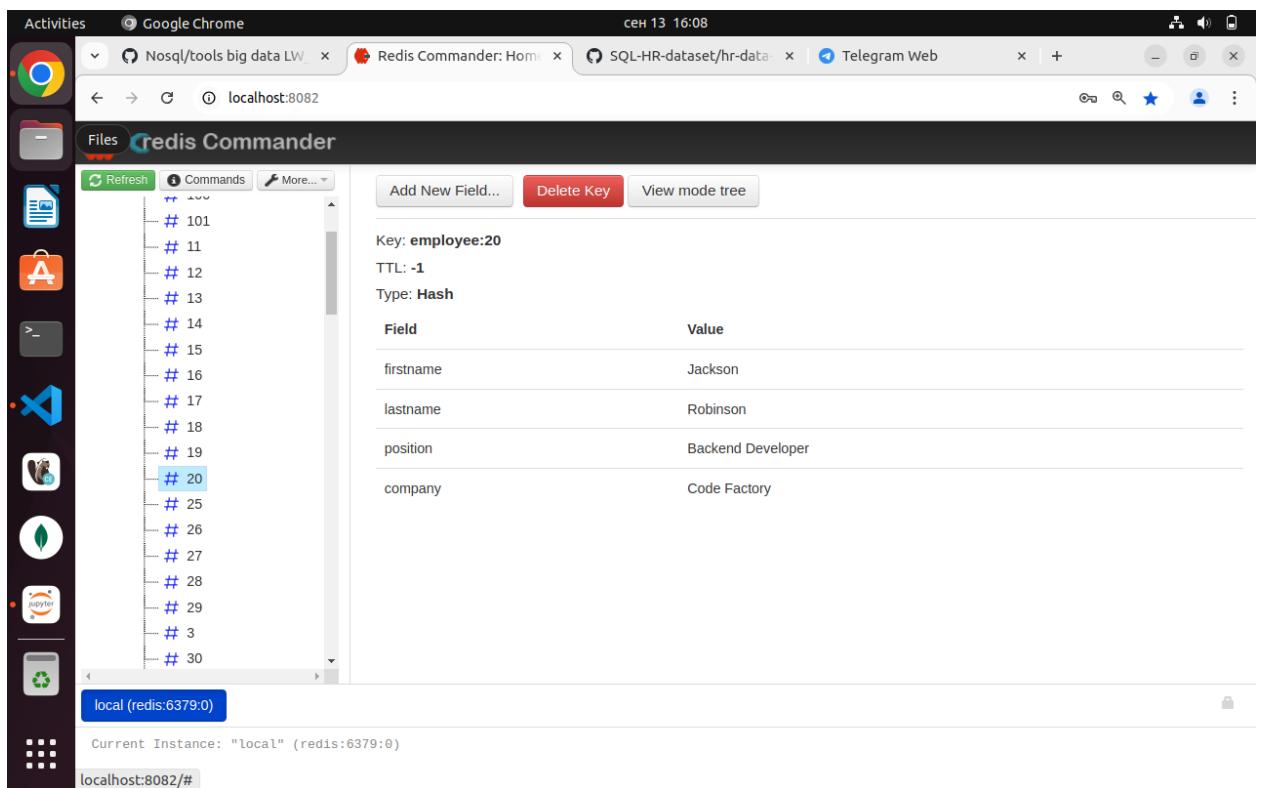


Рис.22 – Проверка успешности удаления строк в Redis Commander

Производим выгрузку данных о сотрудниках в файл формата csv в следующем порядке: подключение к redis, получение всех ключей, декодирование ключа из байтов в строку, определение типа данных, запись данных в файл формата csv, закрытие соединения и выполнение самой выгрузки.

```
[30]: def flatten_data(data):
    if isinstance(data, (str, int, float)):
        return str(data)
    elif isinstance(data, list):
        return json.dumps(data, ensure_ascii=False)
    elif isinstance(data, dict):
        return json.dumps(data, ensure_ascii=False)
    else:
        return str(data)

[31]: def dump_redis_to_csv(filename='redis_dump_BelikMK.csv'):
    # Подключение к Redis
    r = redis.Redis(host='localhost', port=6379, db=0)

    # Получение всех ключей
    keys = r.keys('*')
    with open(filename, 'w', newline='', encoding='utf-8') as csvfile:
        csvwriter = csv.writer(csvfile)
        csvwriter.writerow(['Key', 'Type', 'Value']) # Заголовки
        for key in keys:

            # Декодирование ключа из байтов в строку
            key_str = key.decode('utf-8')

            # Определение типа данных ключа
            key_type = r.type(key).decode('utf-8')
            if key_type == 'string':
                value = r.get(key).decode('utf-8')
            elif key_type == 'list':
                value = r.lrange(key, 0, -1)
                value = [item.decode('utf-8') for item in value]
            elif key_type == 'set':
                value = list(r.smembers(key))
                value = [item.decode('utf-8') for item in value]
            elif key_type == 'hash':
                value = r.hgetall(key)
                value = {k.decode('utf-8'): v.decode('utf-8') for k, v in value.items()}
            elif key_type == 'zset':
                value = r.zrange(key, 0, -1, withscores=True)
                value = [(item[0].decode('utf-8'), item[1]) for item in value]
            else:
                value = f"Неподдерживаемый тип данных: {key_type}"
            # Записываем данные в CSV
            csvwriter.writerow([key_str, key_type, flatten_data(value)])
    # Закрытие соединения
    r.close()
    print(f"Данные сохранены в файл '{filename}'")
```

Рис.23 – Выгрузка данных в в файл redis\_dump\_BelikMK.csv

```
csvwriter.writerow(['Key', 'Type', 'Value']) # Заголовки
for key in keys:

    # Декодирование ключа из байтов в строку
    key_str = key.decode('utf-8')

    # Определение типа данных ключа
    key_type = r.type(key).decode('utf-8')
    if key_type == 'string':
        value = r.get(key).decode('utf-8')
    elif key_type == 'list':
        value = r.lrange(key, 0, -1)
        value = [item.decode('utf-8') for item in value]
    elif key_type == 'set':
        value = list(r.smembers(key))
        value = [item.decode('utf-8') for item in value]
    elif key_type == 'hash':
        value = r.hgetall(key)
        value = {k.decode('utf-8'): v.decode('utf-8') for k, v in value.items()}
    elif key_type == 'zset':
        value = r.zrange(key, 0, -1, withscores=True)
        value = [(item[0].decode('utf-8'), item[1]) for item in value]
    else:
        value = f"Неподдерживаемый тип данных: {key_type}"
    # Записываем данные в CSV
    csvwriter.writerow([key_str, key_type, flatten_data(value)])
# Закрытие соединения
r.close()
print(f"Данные сохранены в файл '{filename}'")
```

Рис.24 – Выгрузка данных в в файл redis\_dump\_BelikMK.csv

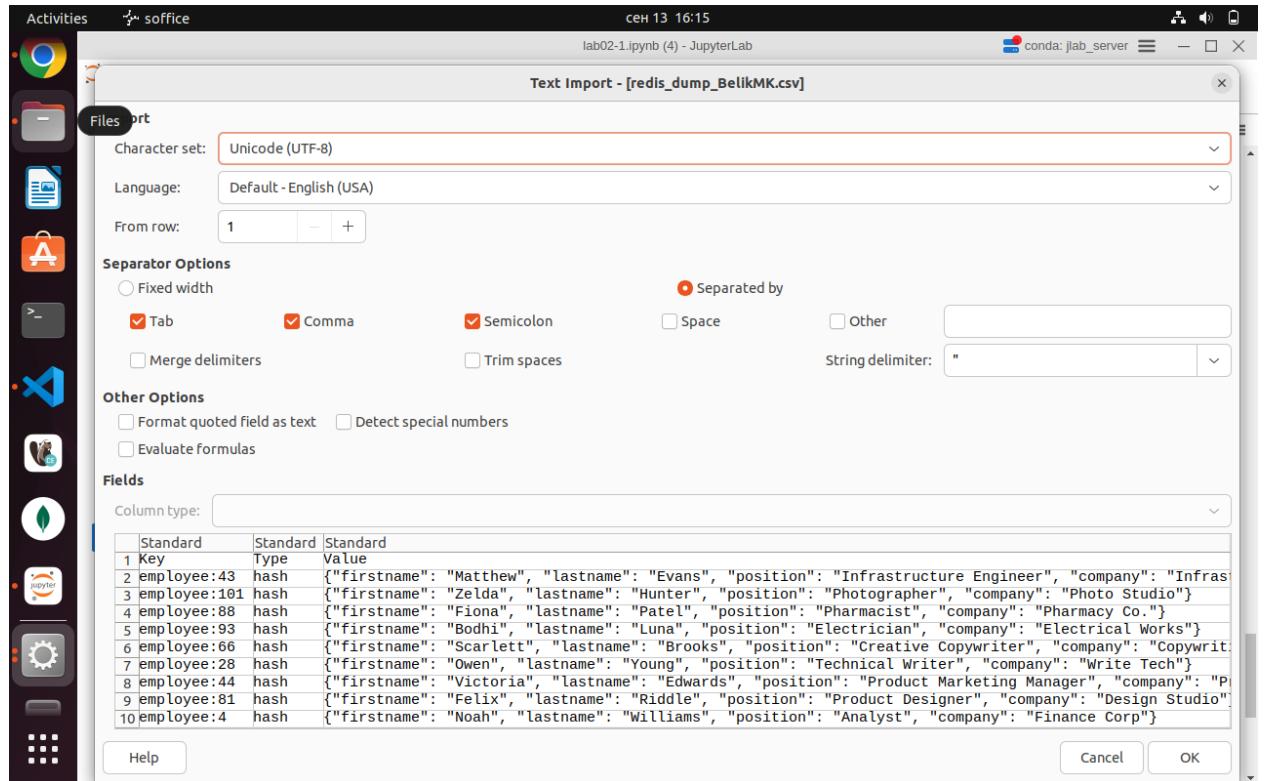


Рис.25 – Открытый файл redis\_dump\_BelikMK.csv

После того, как все необходимые шаги работы с redis были выполнены, необходимо также остановить докер с помощью команды:

```
sudo docker compose down
```

```
services:
  pgweb:
    image: sosedoff/pgweb
    depends_on:
      - postgres
    environment:
      PGWEB_DATABASE_URL: postgres://username:password@postgres:5432/database-name?sslmode=disallowInsecure
    ports:
      - 8085:8081
    restart: on-failure:3
  redis:
    image: redis:latest
    command: redis-server
    volumes:
      - redis:/var/lib/redis
      - redis-config:/usr/local/etc/redis/redis.conf
```

```
Sorry, try again.
[sudo] password for nosql:
[+] Running 4/4
✓ Container pgredis-redis-1          Started      2.8s
✓ Container pgredis-redis-commander-1 Started      6.3s
✓ Container pgredis-postgres-1        Started      2.6s
✓ Container pgredis-pgweb-1           Started      6.0s
● nosql@nosql-vm:~/pgredis$ sudo docker compose down
[sudo] password for nosql:
[+] Running 6/6
✓ Container pgredis-redis-commander-1 Removed      3.7s
✓ Container pgredis-pgweb-1           Removed      3.7s
✓ Container pgredis-postgres-1        Removed      2.0s
✓ Container pgredis-redis-1           Removed      1.5s
✓ Network pgredis default            Removed      0.5s
✓ Network pgredis_redis-network     Removed      0.8s
● nosql@nosql-vm:~/pgredis$
```

Рис.26 – Остановка докер-контейнера

### Шаг 3. Работа с «песочницей»

Переходим по ссылке <https://console.neo4j.org/> и создаем базу данных «Учебные курсы».

Узлы базы: *person* — сотрудники, *student* — студенты. Отношение *learn* связывает студентов с курсами, на которые они записались, а отношения *author*, *speaker* и *editor* связывают авторов, дикторов и монтажеров с курсами, которые они создавали. Именам сотрудников, студентов и курсов соответствуют атрибуты *name*.

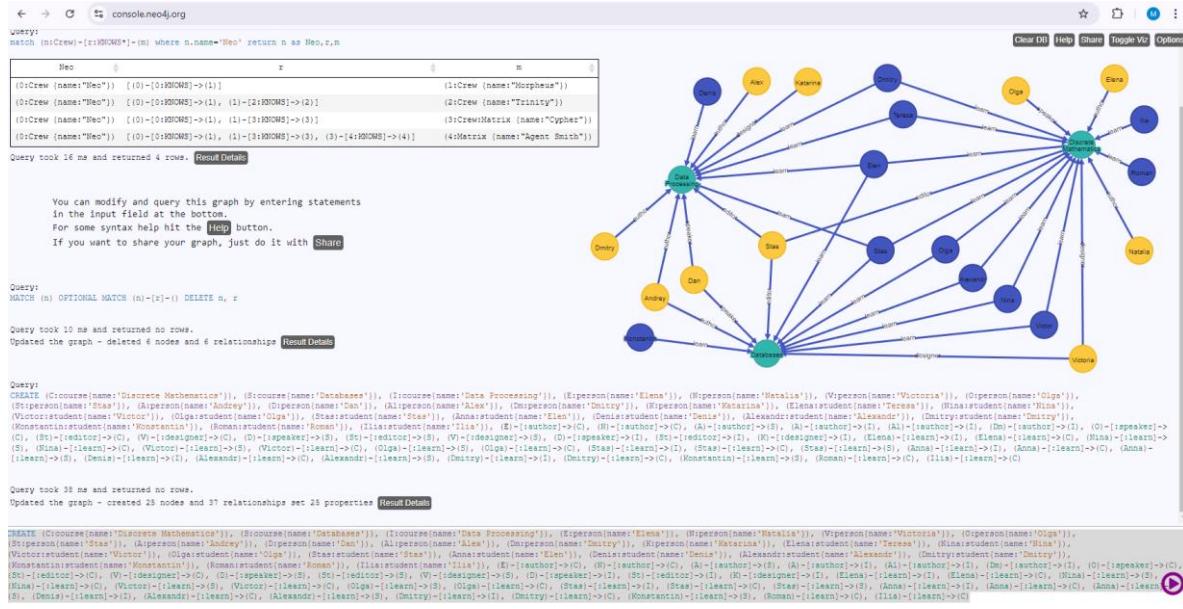


Рис.27 – База данных «Учебные курсы»

1. Далее нам необходимо написать Cypher-запрос, который вернет список всех студентов, записанных на курс "Discrete Mathematics":

```
MATCH (s:student)-[:learn]->(c:course {name: 'Discrete Mathematics'}) RETURN s.name AS StudentName
```

В результате имеем 10 студентов, записанных на курс дискретной математики.

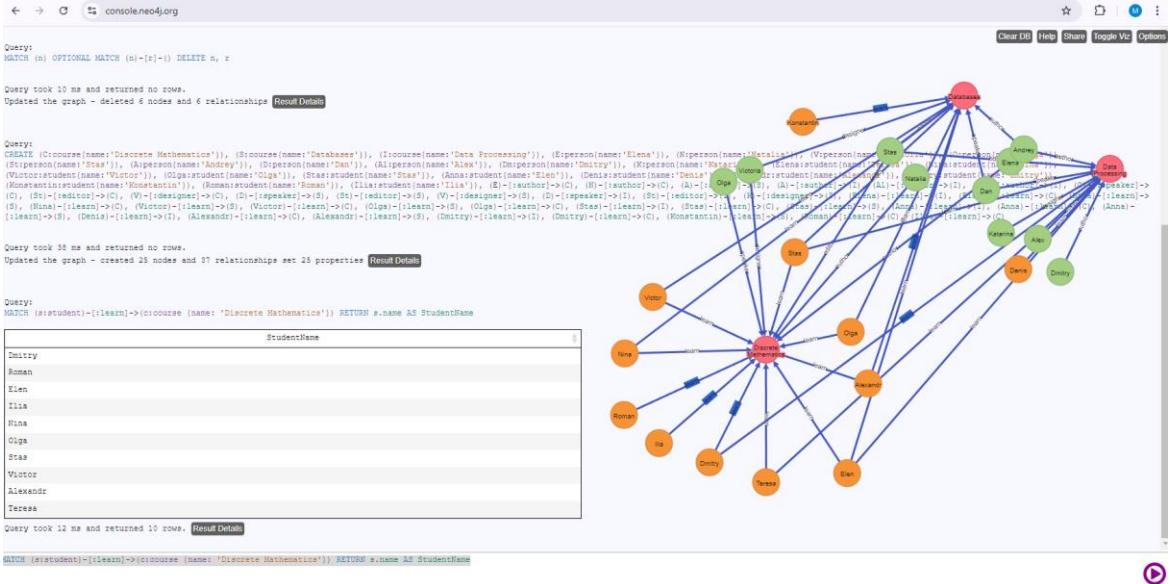


Рис.27 – Результат выполнения 1 запроса

2. Необходимо составить запрос, который вернет список курсов, на которые записаны студенты с именем "Nina" и "Olga":

```
MATCH (s:student) WHERE s.name IN ['Nina', 'Olga'] MATCH (s)-[:learn]->(c:course) RETURN DISTINCT c.name AS CourseName
```

В результате имеем, что студенты с именами Нина и Ольга записаны на курсы: дискретная математика и базы данных.

CourseName
Discrete Mathematics
Databases

Query took 24 ms and returned 2 rows. [Result Details](#)

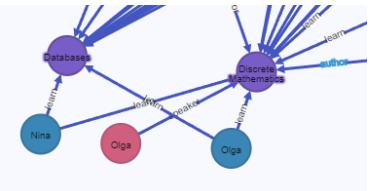


Рис.28 – Результат выполнения 2 запроса

3. Необходимо найти всех авторов курса "Data Processing" и вернуть их имена:

```
MATCH (p:person)-[:author]->(c:course {name: 'Data Processing'}) RETURN p.name AS AuthorName
```

В результате имеем трех авторов курса: Дмитрия, Алекса и Андрея.

AuthorName
Dmitry
Alex
Andrey

Query took 13 ms and returned 3 rows. [Result Details](#)



Рис.29 – Результат выполнения 3 запроса

4. Необходимо написать запрос, который вернет список курсов, созданных сотрудником с именем "Andrey":

```
MATCH (a:person {name: 'Andrey'})-[:author]->(c:course) RETURN c.name AS CourseName
```

В результате имеем, что Андрей был автором двух курсов: обработка данных и базы данных.

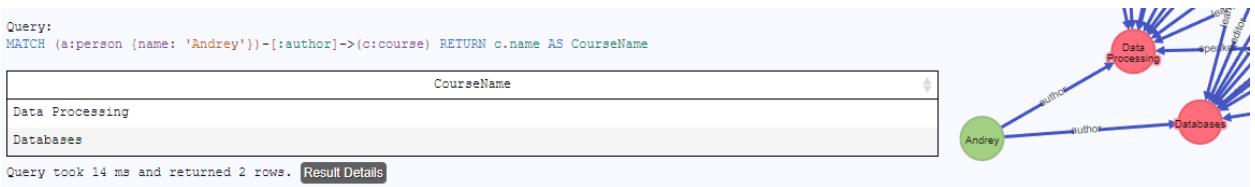


Рис.30 – Результат выполнения 4 запроса

5. Необходимо составить запрос для получения списка всех курсов, где "Stas" является редактором:

```

MATCH (e:person {name: 'Stas'})-[:editor]->(c:course) RETURN c.name AS CourseName

```

В результате имеем, что Стас был редактором всех трех курсов.

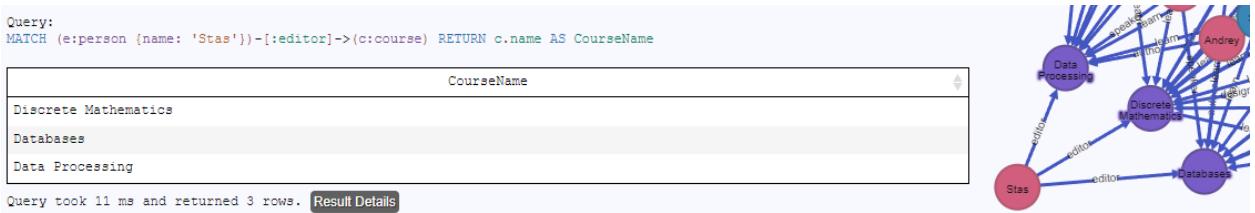


Рис.31 – Результат выполнения 5 запроса

6. Необходимо составить запрос, который вернет всех студентов, записанных на курс "Databases", и указать, какие сотрудники связаны с этим курсом как авторы, дикторы и редакторы.

```

MATCH (c:course {name: 'Databases'})<-[:learn]->(s:student)
OPTIONAL MATCH (c)<-[:author]->(a:person)
OPTIONAL MATCH (c)<-[:speaker]->(n:person)
OPTIONAL MATCH (c)<-[:editor]->(e:person)
RETURN s.name AS StudentName,
       COLLECT(DISTINCT a.name) AS Authors,
       COLLECT(DISTINCT n.name) AS Speakers,
       COLLECT(DISTINCT e.name) AS Editors

```

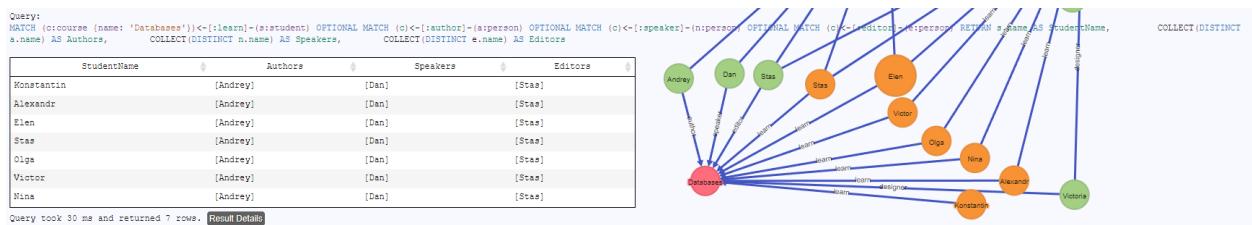


Рис.32 – Результат выполнения 6 запроса

7. Необходимо найти всех сотрудников, которые имеют отношение к созданию курса "Discrete Mathematics", и определить их роль.

```
MATCH (c:course {name: 'Discrete Mathematics'})-[:r]-(p:person) RETURN p.name AS EmployeeName, TYPE(r) AS Role
```

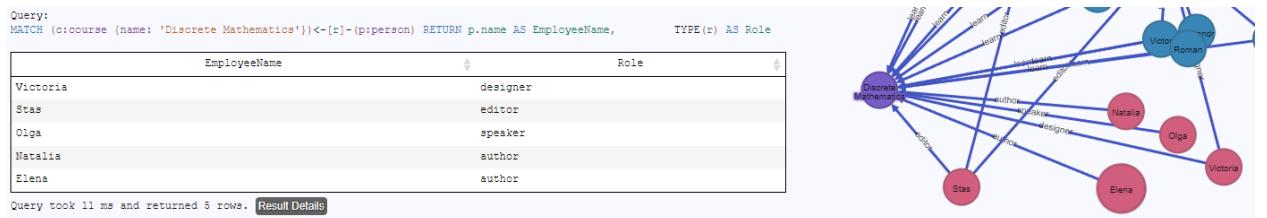


Рис.33 – Результат выполнения 7 запроса

8. Необходимо написать запрос, чтобы найти всех студентов, которые учатся на всех трех курсах ("Discrete Mathematics", "Databases", "Data Processing").

```
MATCH (s:student) WHERE (s)-[:learn]-(:course {name: 'Discrete Mathematics'})  

AND (s)-[:learn]-(:course {name: 'Databases'}) AND (s)-[:learn]-(:course  

{name: 'Data Processing'}) RETURN s.name AS StudentName
```

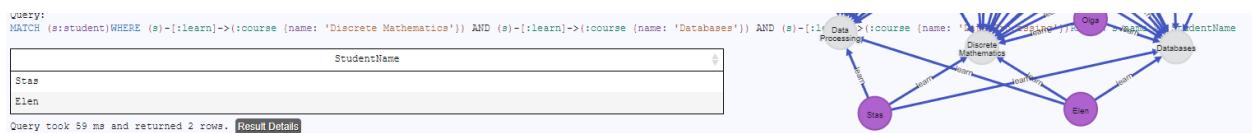


Рис.34 – Результат выполнения 8 запроса

9. Необходимо составить запрос, который вернет список студентов, обучающихся на курсе "Data Processing", и перечислит авторов этого курса.

```
MATCH (s:student)-[:learn]->(c:course {name: 'Data Processing'}), (p:person)-[:author]->(c) RETURN s.name AS StudentName, collect(p.name) AS CourseAuthors
```

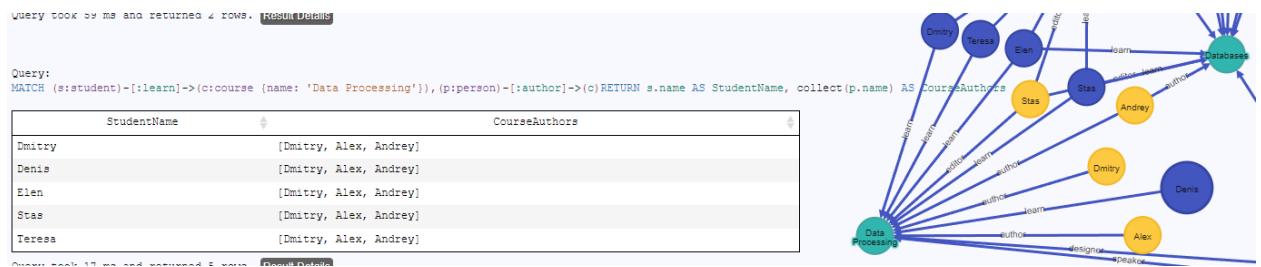


Рис.35 – Результат выполнения 9 запроса

10. Необходимо составить запрос, который вернет количество студентов, записанных на каждый из курсов.

```
MATCH (s:student)-[:learn]->(c:course) RETURN c.name AS CourseName, COUNT(s) AS StudentCount
```

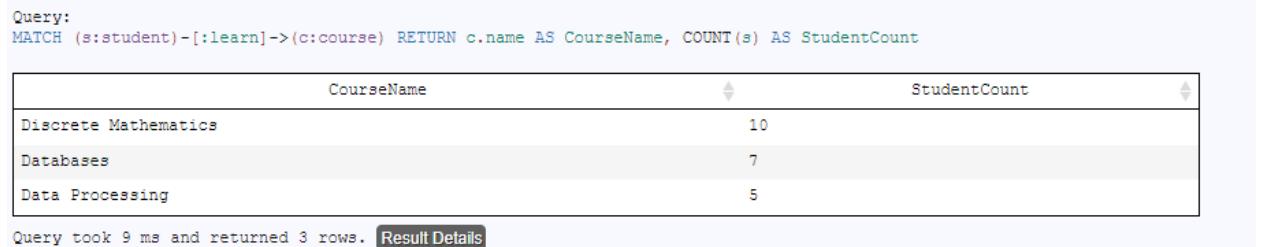


Рис.36 – Результат выполнения 10 запроса

11. Необходимо написать запрос, чтобы получить список всех сотрудников, которые участвуют в создании хотя бы одного курса в роли автора, диктора или редактора.

```
MATCH (p:person)-[:author|:speaker|:editor]->(c:course) RETURN DISTINCT p.name AS EmployeeName
```

```
Query:
MATCH (p:person)-[:author|:speaker|:editor]->(c:course) RETURN DISTINCT p.name AS EmployeeName
```

EmployeeName
Natalia
Elena
Stas
Olga
Andrey
Dan
Dmitry
Alex

Query took 8 ms and returned 8 rows. [Result Details](#)

Рис.37 – Результат выполнения 11 запроса

12. Необходимо написать запрос, который вернет список студентов, записанных на курсы, где "Katarina" является дизайнером

```
MATCH (d:person {name: 'Katarina'})-[:designer]->(c:course)<-[ :learn]-
(s:student) RETURN DISTINCT s.name AS StudentName
```

StudentName
Dmitry
Denis
Elen
Stas
Teresa

Query took 12 ms and returned 5 rows. [Result Details](#)

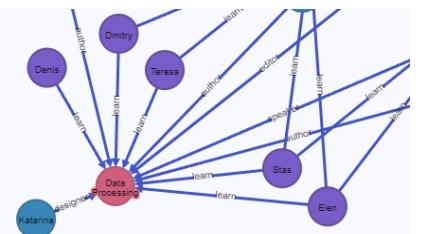


Рис.38 – Результат выполнения 12 запроса

13. Необходимо написать запрос, чтобы найти все курсы, которые созданы командой сотрудников (автор, диктор, редактор), включающей "Dmitry".

```
MATCH (d:person {name: 'Dmitry'})-[:author|:speaker|:editor]->(c:course)
RETURN DISTINCT c.name AS CourseName
```

CourseName
Data Processing

Query took 10 ms and returned 1 rows. [Result Details](#)

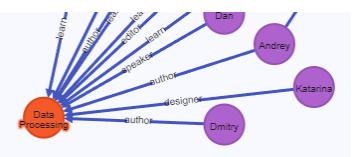


Рис.39 – Результат выполнения 13 запроса

14. Необходимо написать запрос, чтобы найти всех студентов, которые не записаны ни на один курс.

```
MATCH (s:student) WHERE NOT (s)-[:learn]->(:course) RETURN s.name AS StudentName
```

```
Query:  
MATCH (s:student) WHERE NOT (s)-[:learn]->(:course) RETURN s.name AS StudentName  
  
Query took 12 ms and returned no rows. Result Details
```

Рис.40 – Результат выполнения 14 запроса

15. Необходимо написать запрос, который вернет список студентов и количество курсов, на которые они записаны.

```
MATCH (s:student)-[:learn]->(c:course) RETURN s.name AS StudentName, count(c)  
AS CourseCount
```

```
Query:  
MATCH (s:student)-[:learn]->(c:course) RETURN s.name AS StudentName, count(c) AS CourseCount  
  
Show 25 ▾ entries Search: 

| StudentName | CourseCount |
|-------------|-------------|
| Ilia        | 1           |
| Roman       | 1           |
| Dmitry      | 2           |
| Alexandr    | 2           |
| Elen        | 3           |
| Stas        | 3           |
| Olga        | 2           |
| Victor      | 2           |
| Nina        | 2           |
| Teresa      | 2           |
| Konstantin  | 1           |
| Denis       | 1           |

  
Showing 1 to 12 of 12 entries  
Query took 10 ms and returned 12 rows. Result Details
```

Рис.41 – Результат выполнения 15 запроса

16. Необходимо написать запрос, который вернет список всех студентов, записанных на курс "Discrete Mathematics", и их наставников.

```

MATCH      (s:student)-[:learn]->(c:course    {name: 'Discrete Mathematics'})<-
[:author|speaker|editor|designer]-(p:person)   RETURN s.name AS StudentName,
p.name AS MentorName

```

Query:  
`MATCH (s:student)-[:learn]->(c:course {name: 'Discrete Mathematics'})<-[  
:author|speaker|editor|designer]-(p:person) RETURN s.name AS StudentName, p.name AS MentorName`

Show All entries Search:

StudentName	MentorName
Roman	Victoria
Ilia	Victoria
Dmitry	Victoria
Alexandr	Victoria
Elen	Victoria
Stas	Victoria
Olga	Victoria
Victor	Victoria
Nina	Victoria
Teresa	Victoria
Roman	Stas
Ilia	Stas
Dmitry	Stas
Alexandr	Stas
Elen	Stas
Stas	Stas
Olga	Stas
Victor	Stas
Nina	Stas
Teresa	Stas
Roman	Olga
Ilia	Olga
Dmitry	Olga
Alexandr	Olga
Elen	Olga

Рис.42 – Результат выполнения 16 запроса

Stas		Olga
Olga		Olga
Victor		Olga
Nina		Olga
Teresa		Olga
Roman		Natalia
Ilia		Natalia
Dmitry		Natalia
Alexandr		Natalia
Elen		Natalia
Stas		Natalia
Olga		Natalia
Victor		Natalia
Nina		Natalia
Teresa		Natalia
Roman		Elena
Ilia		Elena
Dmitry		Elena
Alexandr		Elena
Elen		Elena
Stas		Elena
Olga		Elena
Victor		Elena
Nina		Elena
Teresa		Elena

Showing 1 to 50 of 50 entries

Query took 18 ms and returned 50 rows. [Result Details](#)

Рис.43 – Результат выполнения 16 запроса