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

Отчёт по лабораторной работе № 2.1

«Изучение методов хранения данных на основе NoSQL»

Выполнила:
студентка группы АДЭУ-211,
Макарова Екатерина Павловна

Преподаватель:
Босенко Тимур Муртазович,
Доцент, кандидат технических наук

Москва 2024

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.

Задание: сгенерировать данные (не менее 100 записей) по предметной области «Транспортные средства и их владельцы». Необходимо загрузить данные в MongoDB и Redis. Студент должен продемонстрировать выполнение базовых операций (вставка, выборка, обновление, удаление) для каждой из СУБД.

1. Запуск контейнера MongoDB

Был выбран каталог **mongodb**, в которой хранится контейнер MongoDB, проверено наличие файла конфигурации в директории с помощью команды ls (Рис. 1).

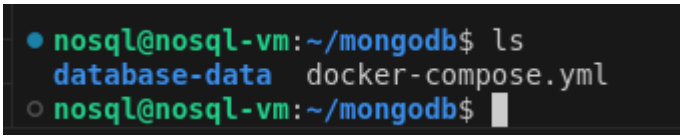


Рис. 1

Выполнена команда sudo docker compose up -d, которая запустила контейнер в конфигурационном файле docker-compose.yml (Рис. 2).

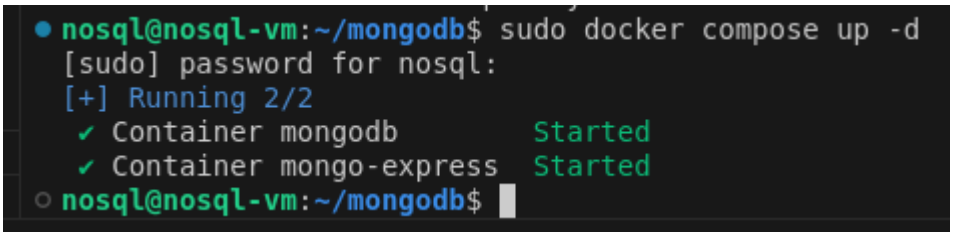


Рис. 2

Проверка доступа к хосту (Рис. 3, Рис. 4).

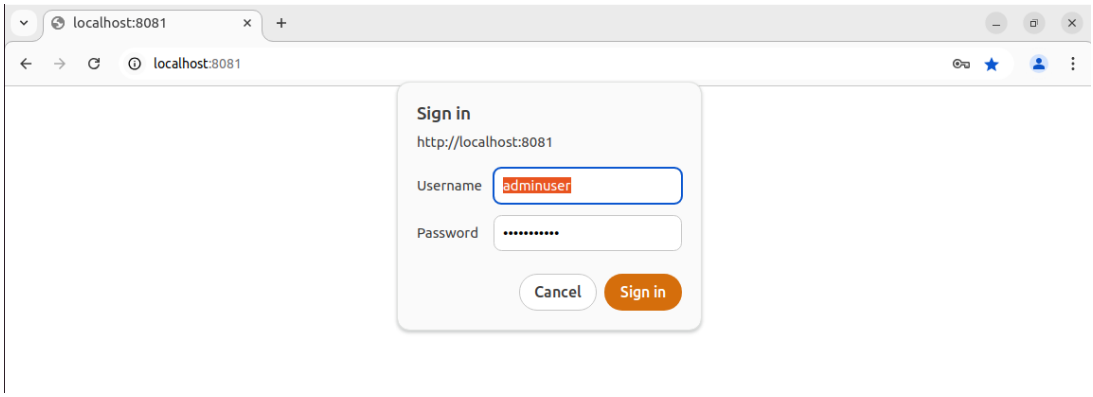


Рис. 3

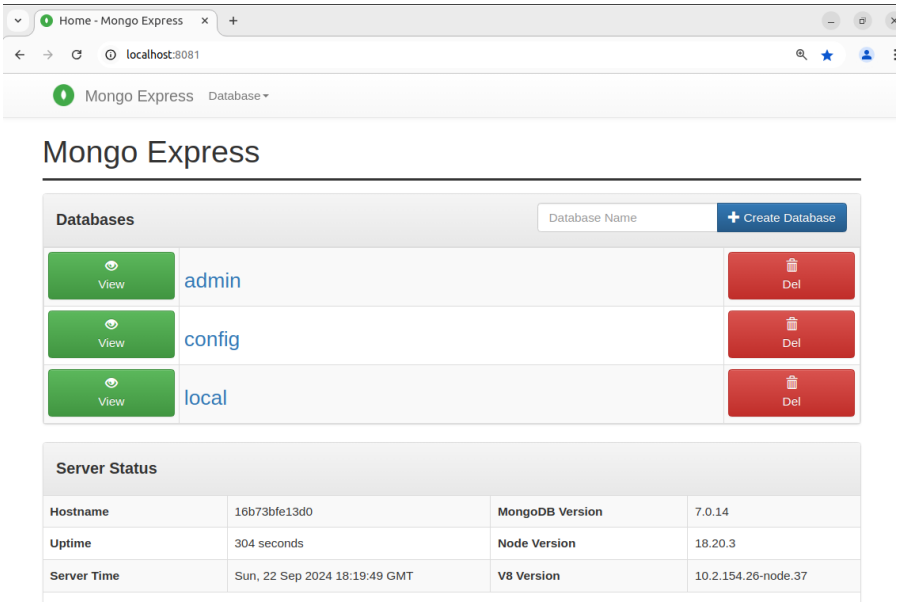


Рис. 4

Подключение к MongoDB с помощью Jupyter и загрузка данных

Установка библиотеки pymongo (Рис. 5)

```
!pip install pymongo
```

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

Рис. 5

Создание подключения и проверка успешности (Рис. 6)

```
from pymongo import MongoClient
```

```
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['cars_owners']
# Выбор коллекции
    labs_collection = db['car_own']
except Exception as e:
     print(f"Ошибка подключения: {e}")
```

```
Подключение к MongoDB установлено успешно!
```

Рис. 6

База данных – cars_owners

Таблица (коллекция) -car_own

Генерация данных о машинах и их владельцах (Рис. 7).

Для генерации данных о владельцах была использована библиотека Faker, для генерации данных о машинах были использованы списки с названием марок машин и их моделей

```python
#import libraries
import csv
import random
from faker import Faker

#create instance
fake = Faker()

# Generate data
def generate_car_data(num_records):
#Parametrs for data generation
    car_brands = ['Toyota', 'Ford', 'BMW', 'Honda', 'Reno', 'Chevrolet']
    car_models = ['Camry', 'Focus', '3 Series', 'Civic', 'Logan', 'Malibu']
    data = []
    for _ in range(num_records):
        car = {
            'brand': random.choice(car_brands),
            'model': random.choice(car_models),
            'year': random.randint(2000, 2023),
            'last_name': fake.last_name(),
            'first_name': fake.first_name(),
            'middle_name': fake.first_name(),
            'age': random.randint(18, 75)

            }

        data.append(car)
    return data
car_records = generate_car_data(150)
for record in car_records:
    print(record)
```

```
{'brand': 'BMW', 'model': 'Focus', 'year': 2007, 'last_name': 'Smith', 'first_name': 'Courtney', 'middle_name': 'Marissa', 'age': 66}
{'brand': 'Ford', 'model': '3 Series', 'year': 2014, 'last_name': 'Long', 'first_name': 'Carolyn', 'middle_name': 'Elijah', 'age': 26}
{'brand': 'Reno', 'model': '3 Series', 'year': 2018, 'last_name': 'Rivera', 'first_name': 'Jane', 'middle_name': 'Melissa', 'age': 27}
{'brand': 'Toyota', 'model': '3 Series', 'year': 2005, 'last_name': 'Andrews', 'first_name': 'Mary', 'middle_name': 'Anthony', 'age': 68}
{'brand': 'Reno', 'model': 'Focus', 'year': 2012, 'last_name': 'Wang', 'first_name': 'Bradley', 'middle_name': 'Adam', 'age': 64}
{'brand': 'Reno', 'model': 'Logan', 'year': 2008, 'last_name': 'Burke', 'first_name': 'Dylan', 'middle_name': 'Kimberly', 'age': 52}
{'brand': 'Ford', 'model': '3 Series', 'year': 2002, 'last_name': 'Taylor', 'first_name': 'Selena', 'middle_name': 'Rebecca', 'age': 30}
```

Рис. 7

Запрос к коллекции на данные (Рис. 8).

```python
documents = labs_collection.find()
for doc in documents:
    print(doc)
```

```
{'_id': ObjectId('66f06dd87829c4c1bd8ddc8d'), 'brand': 'BMW', 'model': 'Focus', 'year': 2007, 'last_name': 'Smith', 'first_name': 'Courtney', 'middle_nam
e': 'Marissa', 'age': 66}
{'_id': ObjectId('66f06dd87829c4c1bd8ddc8e'), 'brand': 'Ford', 'model': '3 Series', 'year': 2014, 'last_name': 'Long', 'first_name': 'Carolyn', 'middle_na
me': 'Elijah', 'age': 26}
{'_id': ObjectId('66f06dd87829c4c1bd8ddc8f'), 'brand': 'Reno', 'model': '3 Series', 'year': 2018, 'last_name': 'Rivera', 'first_name': 'Jane', 'middle_nam
e': 'Melissa', 'age': 27}
{'_id': ObjectId('66f06dd87829c4c1bd8ddc90'), 'brand': 'Toyota', 'model': '3 Series', 'year': 2005, 'last_name': 'Andrews', 'first_name': 'Mary', 'middle_
name': 'Anthony', 'age': 68}
{'_id': ObjectId('66f06dd87829c4c1bd8ddc91'), 'brand': 'Reno', 'model': 'Focus', 'year': 2012, 'last_name': 'Wang', 'first_name': 'Bradley', 'middle_nam
e': 'Adam', 'age': 64}
{'_id': ObjectId('66f06dd87829c4c1bd8ddc92'), 'brand': 'Reno', 'model': 'Logan', 'year': 2008, 'last_name': 'Burke', 'first_name': 'Dylan', 'middle_name':
'Kimberly', 'age': 52}
{'_id': ObjectId('66f06dd87829c4c1bd8ddc93'), 'brand': 'Ford', 'model': '3 Series', 'year': 2002, 'last_name': 'Taylor', 'first_name': 'Selena', 'middle_n
ame': 'Rebecca', 'age': 30}
{'_id': ObjectId('66f06dd87829c4c1bd8ddc94'), 'brand': 'Toyota', 'model': 'Focus', 'year': 2017, 'last_name': 'Wilson', 'first_name': 'Robert', 'middle_na
me': 'Jeremy', 'age': 22}
{'_id': ObjectId('66f06dd87829c4c1bd8ddc95'), 'brand': 'BMW', 'model': 'Focus', 'year': 2013, 'last_name': 'Cooper', 'first_name': 'Timothy', 'middle_nam
e': 'Kelsey', 'age': 59}
{'_id': ObjectId('66f06dd87829c4c1bd8ddc96'), 'brand': 'Chevrolet', 'model': 'Logan', 'year': 2021, 'last_name': 'Franklin', 'first_name': 'Jessica', 'mid
dle_name': 'Megan', 'age': 18}
{'_id': ObjectId('66f06dd87829c4c1bd8ddc97'), 'brand': 'Chevrolet', 'model': 'Camry', 'year': 2003, 'last_name': 'Flowers', 'first_name': 'David', 'middle
_name': 'Sean', 'age': 50}
{'_id': ObjectId('66f06dd87829c4c1bd8ddc98'), 'brand': 'BMW', 'model': 'Malibu', 'year': 2006, 'last_name': 'Wilson', 'first_name': 'Joyce', 'middle_nam
e': 'Barbara', 'age': 73}
{'_id': ObjectId('66f06dd87829c4c1bd8ddc99'), 'brand': 'Ford', 'model': 'Malibu', 'year': 2022, 'last_name': 'Hampton', 'first_name': 'Adriana', 'middle_n
ame': 'Edgar', 'age': 65}
{'_id': ObjectId('66f06dd87829c4c1bd8ddc9a'), 'brand': 'Honda', 'model': 'Focus', 'year': 2010, 'last_name': 'Donovan', 'first_name': 'Nathaniel', 'middle
_name': 'Kevin', 'age': 61}
{'_id': ObjectId('66f06dd87829c4c1bd8ddc9b'), 'brand': 'Ford', 'model': 'Camry', 'year': 2018, 'last_name': 'Black', 'first_name': 'Tammy', 'middle_name':
'Angela', 'age': 42}
{'_id': ObjectId('66f06dd87829c4c1bd8ddc9c'), 'brand': 'Reno', 'model': '3 Series', 'year': 2000, 'last_name': 'Murphy', 'first_name': 'Tim', 'middle_nam
```

Рис. 8

В Mongo Express была создана база данных cars_owners с данными о машинах и их владельцах (Рис. 9).

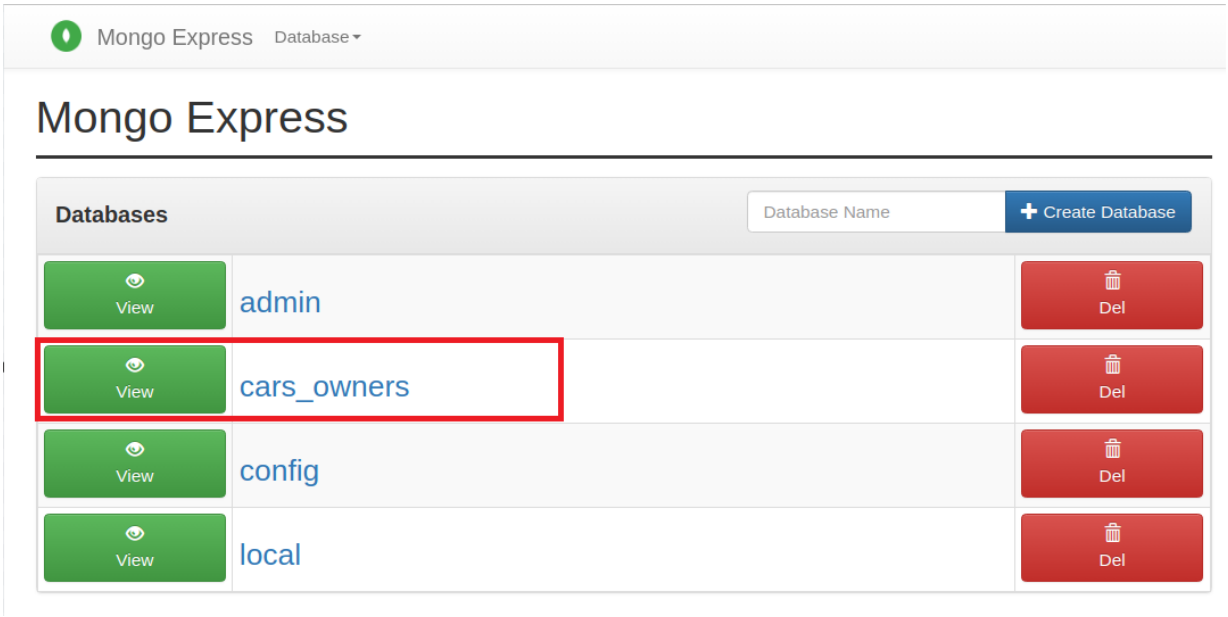


Рис. 9

Проверка данных в Mongo compass (Рис. 10).

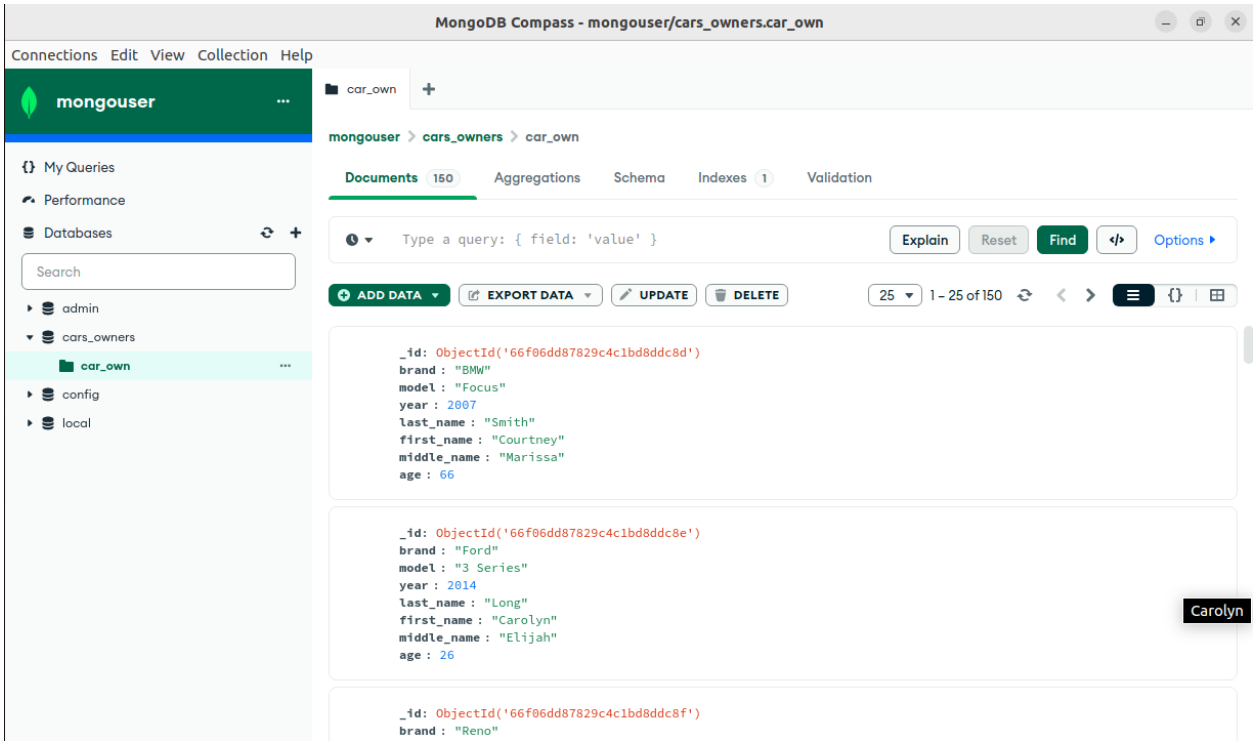


Рис. 10

Выборка данных с помощью Mongo Compass (Рис. 11)

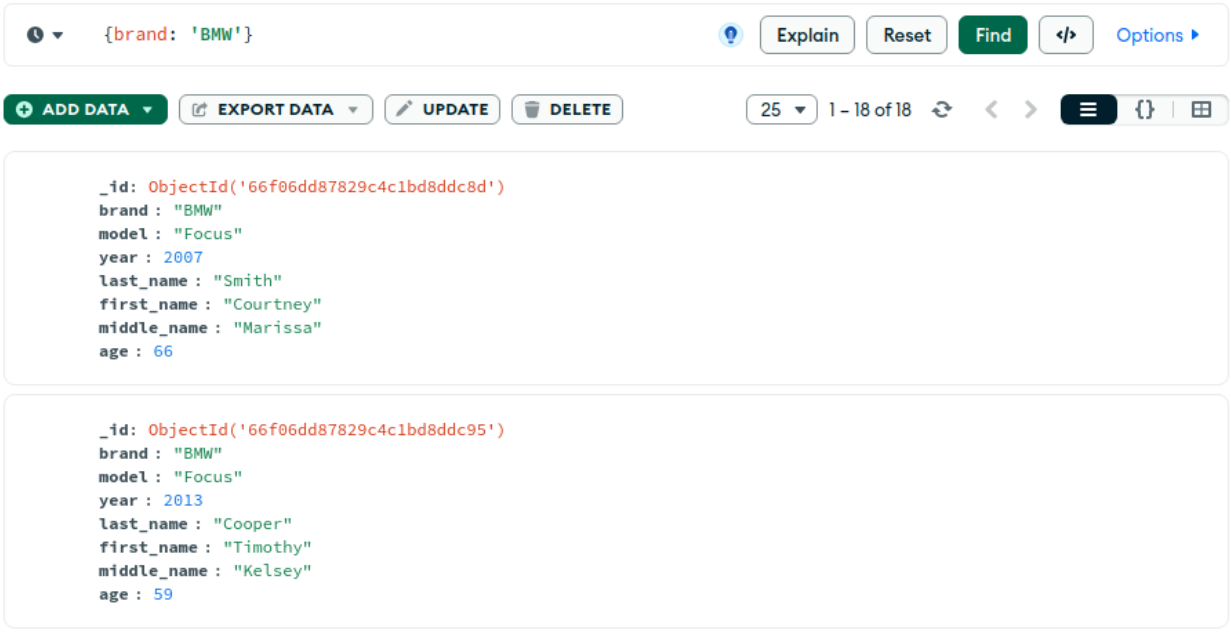
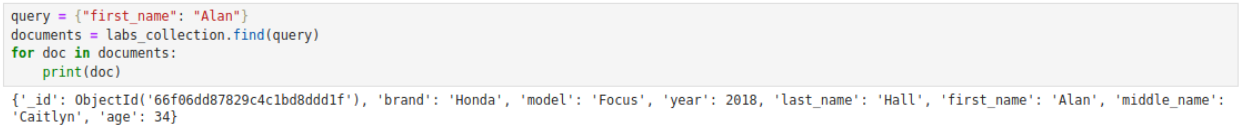


Рис. 11

Выборка данных с помощью кода python в Jupyter (Рис. 12)



```
query = {"first_name": "Alan"}
documents = labs_collection.find(query)
for doc in documents:
    print(doc)
```

```
{'_id': ObjectId('66f06dd87829c4c1bd8ddd1f'), 'brand': 'Honda', 'model': 'Focus', 'year': 2018, 'last_name': 'Hall', 'first_name': 'Alan', 'middle_name': 'Caitlyn', 'age': 34}
```

Рис. 12

Удаление записей в Mongo Compass (Рис. 13)

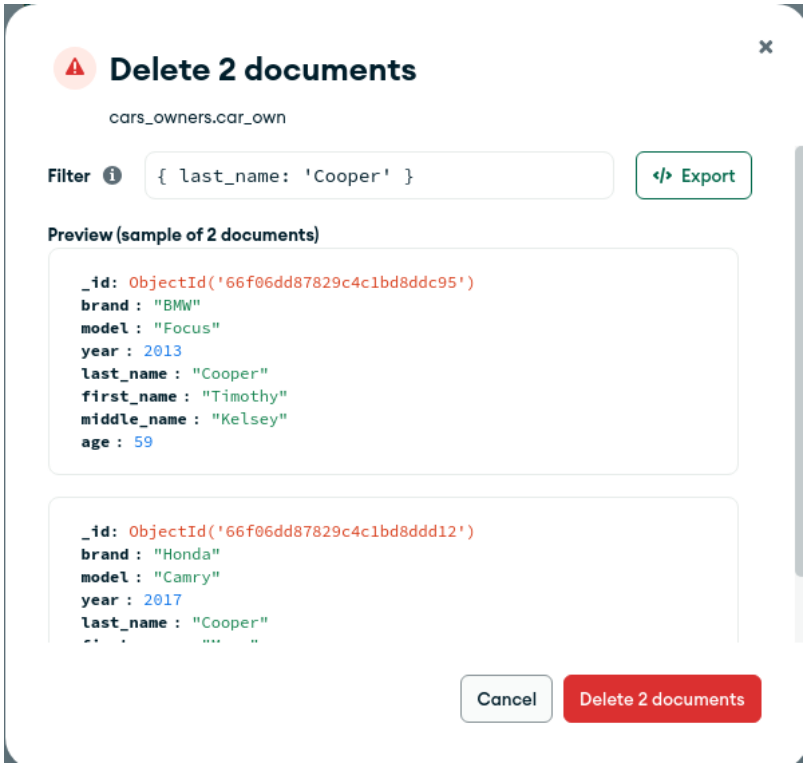


Рис. 13

Удаление с помощью кода (Рис. 15)

Запрос на выборку данных, где возраст владельца машины больше 70 (Рис. 14).

```
query_drop = {"age":  {"$gt": 70}}
documents = labs_collection.find(query_drop)
for doc in documents:
    print(doc)
```

```
{'_id': ObjectId('66f06dd87829c4c1bd8ddc98'), 'brand': 'BMW', 'model': 'Malibu', 'year': 2006, 'last_name': 'Wilson', 'first_name': 'Joyce', 'middle_nam
e': 'Barbara', 'age': 73}
{'_id': ObjectId('66f06dd87829c4c1bd8ddc9c'), 'brand': 'Reno', 'model': '3 Series', 'year': 2000, 'last_name': 'Murphy', 'first_name': 'Tim', 'middle_nam
e': 'Victoria', 'age': 75}
{'_id': ObjectId('66f06dd87829c4c1bd8ddca1'), 'brand': 'Chevrolet', 'model': 'Logan', 'year': 2012, 'last_name': 'Rice', 'first_name': 'Noah', 'middle_nam
e': 'Jose', 'age': 71}
{'_id': ObjectId('66f06dd87829c4c1bd8ddcaa'), 'brand': 'Honda', 'model': '3 Series', 'year': 2023, 'last_name': 'Jones', 'first_name': 'Cheryl', 'middle_n
ame': 'David', 'age': 74}
{'_id': ObjectId('66f06dd87829c4c1bd8ddcbe'), 'brand': 'Honda', 'model': 'Malibu', 'year': 2022, 'last_name': 'Mullins', 'first_name': 'Cheryl', 'middle_n
ame': 'Joseph', 'age': 74}
{'_id': ObjectId('66f06dd87829c4c1bd8ddcd9'), 'brand': 'BMW', 'model': 'Focus', 'year': 2008, 'last_name': 'Smith', 'first_name': 'Matthew', 'middle_nam
e': 'Richard', 'age': 74}
{'_id': ObjectId('66f06dd87829c4c1bd8ddcee'), 'brand': 'Chevrolet', 'model': 'Camry', 'year': 2018, 'last_name': 'Sanders', 'first_name': 'Kristen', 'midd
le_name': 'Brandon', 'age': 72}
{'_id': ObjectId('66f06dd87829c4c1bd8ddcef'), 'brand': 'BMW', 'model': 'Logan', 'year': 2021, 'last_name': 'Wright', 'first_name': 'Misty', 'middle_name':
'Tracey', 'age': 73}
{'_id': ObjectId('66f06dd87829c4c1bd8ddd01'), 'brand': 'Toyota', 'model': '3 Series', 'year': 2000, 'last_name': 'Green', 'first_name': 'Douglas', 'middle
_name': 'Jamie', 'age': 71}
{'_id': ObjectId('66f06dd87829c4c1bd8ddd10'), 'brand': 'Reno', 'model': 'Civic', 'year': 2006, 'last_name': 'Gonzalez', 'first_name': 'Brittany', 'middle_
name': 'Thomas', 'age': 74}
{'_id': ObjectId('66f06dd87829c4c1bd8ddd1c'), 'brand': 'Ford', 'model': 'Camry', 'year': 2007, 'last_name': 'Coleman', 'first_name': 'Andrew', 'middle_nam
e': 'Kevin', 'age': 72}
```

Рис. 14

```
drop = labs_collection.delete_many(query_drop)
```

Рис. 15

Проверка удаления значений (**Ошибка! Источник ссылки не найден.**).

```
query_drop = {"age":  {"$gt": 70}}
documents = labs_collection.find(query_drop)
for doc in documents:
    print(doc)
```

2. Запуск контейнера Redis

В каталоге pgredis был запущен конфигурационный файл с помощью команды sudo docker compose up -d (Рис. 16).

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

nosql@nosql-vm:~/pgredis$ ls
docker-compose2.txt  docker-compose.yml
nosql@nosql-vm:~/pgredis$ sudo docker compose up -d
[sudo] password for nosql:
[+] Running 4/4
 ✔ Container pgredis-redis-1            Started                    1.4s
 ✔ Container pgredis-postgres-1         Started                    1.4s
 ✔ Container pgredis-redis-commander-1  Started                    2.6s
 ✔ Container pgredis-pgweb-1            Started                    2.6s
nosql@nosql-vm:~/pgredis$
```

Рис. 16

Проверка подключения в Redis Commander (Рис. 17).

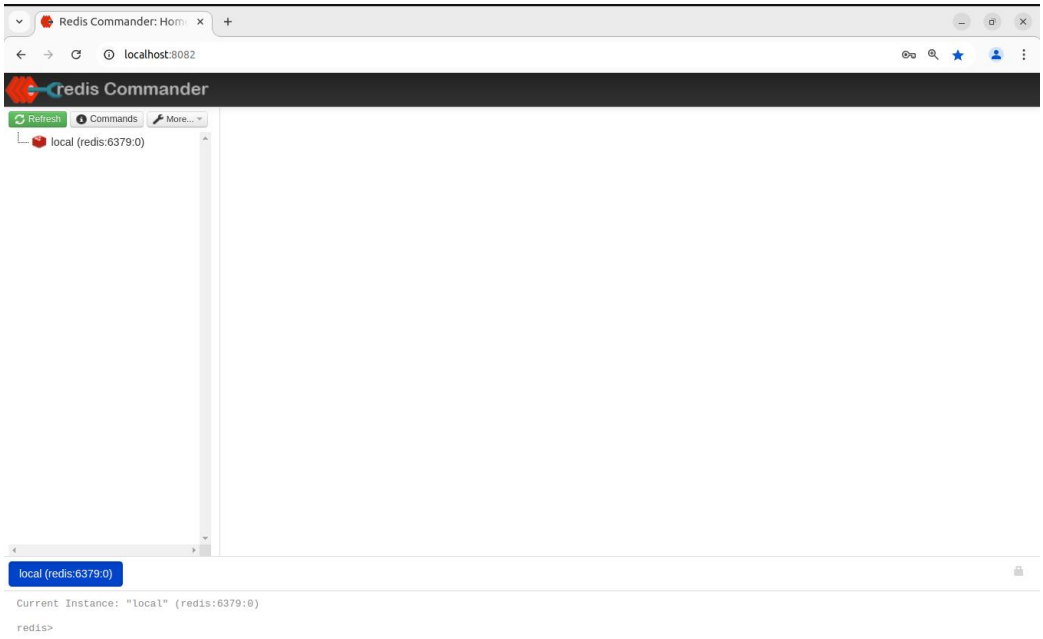


Рис. 17

Установка библиотеки в Jupyter (Рис. 18).

```
!pip install redis
```

```
Requirement already satisfied: redis in /home/nosql/.config/jupyterlab-desktop/jlab_server/lib/python3.12/site-packages (5.0.8)
```

Рис. 18

Импорт библиотек (Рис. 19).

```
import redis
import csv
import json
```

Рис. 19

Подключение к Redis с аутентификацией (Рис. 20).

```
# 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. 20

Импорт библиотек faker – для генерации данных о владельце (ФИО) и random – для выбора рандомных значений из констант, создание функции для печати разделителя (Рис. 21).

```
import random
from faker import Faker

# Создаем экземпляр Faker для генерации случайных данных
fake = Faker()
```

```
# Функция для печати разделителя
def print_separator(message):
    print(f"\n{'='*20} {message} {'='*20}")
```

Рис. 21

Генерация 150 записей о машинах и их владельцах в формате - ключ – значение (Рис. 22).

```
# Генерация данных
print_separator("Создание данных о машинах и владельцах")

# Словарь для хранения данных
car_registry = {}

# Генерация 150 автомобилей с владельцами
for i in range(1, 151):
    owner_name = fake.name()
    owner_age = random.randint(18, 80)
    owner_gender = random.choice(['Male', 'Female'])

    car_brand = random.choice(['Toyota', 'Ford', 'Honda', 'Chevrolet', 'BMW'])
    car_model = random.choice(['Model A', 'Model B', 'Model C'])
    engine_size = round(random.uniform(1.0, 5.0), 1)  # объем от 1.0 до 5.0 литров

    # Строка
    r.set(f"car_{i}", f"{car_brand} {car_model}")
    print(f"Создана строка: car_{i} = {car_brand} {car_model}")

    # Список владельцев
    r.rpush(f"owners_{i}", *[f"{owner_name}, {owner_age}, {owner_gender}"])
    print(f"Создан список: owners_{i} = {r.lrange(f'owners_{i}', 0, -1)}")

    # Множество услуг
    r.sadd(f"services_{i}", *[f"service_{j}" for j in range(1, 4)])
    print(f"Создано множество: services_{i} = {r.smembers(f'services_{i}')}")

    # Хэш с деталями автомобиля
    r.hset(f"details_{i}", mapping={
        'brand': car_brand,
        'model': car_model,
        'engine_size': engine_size,
        'owner': f"{owner_name}, {owner_age}, {owner_gender}"
    })
    print(f"Создан хэш: details_{i} = {r.hgetall(f'details_{i}')}")

    # Упорядоченное множество по объему двигателя
    r.zadd(f"engine_sizes", {f"{car_brand} {car_model}": engine_size})
    print(f"Создано упорядоченное множество: engine_sizes = {r.zrange(f'engine_sizes', 0, -1, withscores=True)}")

print_separator("Генерация завершена")
```

Рис. 22

Вывод сгенерированных данных (Рис. 23).

```
==================== Создание данных о машинах и владельцах ====================
Создана строка: car_1 = Honda Model A
Создан список: owners_1 = [b'Christian Jones, 29, Female', b'Christopher Vargas, 46, Female']
Создано множество: services_1 = {b'service_1', b'service_2', b'service_3'}
Создан хэш: details_1 = {b'brand': b'Honda', b'model': b'Model A', b'engine_size': b'3.4', b'ownre': b'Christian Jones, 29, Female', b'owner': b'Christopher Vargas, 46, Fema
le'}
Создано упорядоченное множество: engine_sizes = [(b'Toyota Model A', 1.4), (b'Ford Model C', 1.5), (b'Honda Model C', 1.8), (b'Toyota Model B', 1.9), (b'Ford Model A', 2.0),
(b'Chevrolet Model A', 2.3), (b'Honda Model B', 2.5), (b'Chevrolet Model C', 2.6), (b'BMW Model B', 2.7), (b'BMW Model A', 3.1), (b'Honda Model A', 3.4), (b'Chevrolet Model
B', 3.8), (b'Ford Model B', 4.1), (b'Toyota Model C', 4.5), (b'BMW Model C', 4.6)]
Создана строка: car_2 = BMW Model B
Создан список: owners_2 = [b'Jerome Barnes, 80, Female', b'Matthew Williams, 74, Female']
Создано множество: services_2 = {b'service_1', b'service_2', b'service_3'}
Создан хэш: details_2 = {b'brand': b'BMW', b'model': b'Model B', b'engine_size': b'3.3', b'ownre': b'Jerome Barnes, 80, Female', b'owner': b'Matthew Williams, 74, Female'}
Создано упорядоченное множество: engine_sizes = [(b'Toyota Model A', 1.4), (b'Ford Model C', 1.5), (b'Honda Model C', 1.8), (b'Toyota Model B', 1.9), (b'Ford Model A', 2.0),
(b'Chevrolet Model A', 2.3), (b'Honda Model B', 2.5), (b'Chevrolet Model C', 2.6), (b'BMW Model A', 3.1), (b'BMW Model B', 3.3), (b'Honda Model A', 3.4), (b'Chevrolet Model
B', 3.8), (b'Ford Model B', 4.1), (b'Toyota Model C', 4.5), (b'BMW Model C', 4.6)]
Создана строка: car_3 = Toyota Model C
Создан список: owners_3 = [b'Donald Hudson, 51, Male', b'Donna Morse, 55, Female']
Создано множество: services_3 = {b'service_1', b'service_2', b'service_3'}
Создан хэш: details_3 = {b'brand': b'Toyota', b'model': b'Model C', b'engine_size': b'3.7', b'ownre': b'Donald Hudson, 51, Male', b'owner': b'Donna Morse, 55, Female'}
Создано упорядоченное множество: engine_sizes = [(b'Toyota Model A', 1.4), (b'Ford Model C', 1.5), (b'Honda Model C', 1.8), (b'Toyota Model B', 1.9), (b'Ford Model A', 2.0),
(b'Chevrolet Model A', 2.3), (b'Honda Model B', 2.5), (b'Chevrolet Model C', 2.6), (b'BMW Model A', 3.1), (b'BMW Model B', 3.3), (b'Honda Model A', 3.4), (b'Toyota Model C',
3.7), (b'Chevrolet Model B', 3.8), (b'Ford Model B', 4.1), (b'BMW Model C', 4.6)]
Создана строка: car_4 = Ford Model A
Создан список: owners_4 = [b'Eric Costa, 66, Male', b'Gail Hughes, 42, Female']
Создано множество: services_4 = {b'service_1', b'service_2', b'service_3'}
Создан хэш: details_4 = {b'brand': b'Ford', b'model': b'Model A', b'engine_size': b'1.8', b'ownre': b'Eric Costa, 66, Male', b'owner': b'Gail Hughes, 42, Female'}
Создано упорядоченное множество: engine_sizes = [(b'Toyota Model A', 1.4), (b'Ford Model C', 1.5), (b'Ford Model A', 1.8), (b'Honda Model C', 1.8), (b'Toyota Model B', 1.9),
(b'Chevrolet Model A', 2.3), (b'Honda Model B', 2.5), (b'Chevrolet Model C', 2.6), (b'BMW Model A', 3.1), (b'BMW Model B', 3.3), (b'Honda Model A', 3.4), (b'Toyota Model C',
3.7), (b'Chevrolet Model B', 3.8), (b'Ford Model B', 4.1), (b'BMW Model C', 4.6)]
Создана строка: car_5 = Honda Model B
Создан список: owners_5 = [b'Brian Mitchell, 75, Male', b'Sharon Carter, 30, Female']
Создано множество: services_5 = {b'service_1', b'service_2', b'service_3'}
Создан хэш: details_5 = {b'brand': b'Honda', b'model': b'Model B', b'engine_size': b'4.5', b'ownre': b'Brian Mitchell, 75, Male', b'owner': b'Sharon Carter, 30, Female'}
Создано упорядоченное множество: engine_sizes = [(b'Toyota Model A', 1.4), (b'Ford Model C', 1.5), (b'Ford Model A', 1.8), (b'Honda Model C', 1.8), (b'Toyota Model B', 1.9),
(b'Chevrolet Model A', 2.3), (b'Chevrolet Model C', 2.6), (b'BMW Model A', 3.1), (b'BMW Model B', 3.3), (b'Honda Model A', 3.4), (b'Toyota Model C', 3.7), (b'Chevrolet Model
B', 3.8), (b'Ford Model B', 4.1), (b'Honda Model B', 4.5), (b'BMW Model C', 4.6)]
Создана строка: car_6 = Ford Model B
Создан список: owners_6 = [b'Dr. Gavin Green Jr., 47, Female', b'Michael Williams, 37, Female']
Создано множество: services_6 = {b'service_1', b'service_2', b'service_3'}
Создан хэш: details_6 = {b'brand': b'Ford', b'model': b'Model B', b'engine_size': b'2.7', b'ownre': b'Dr. Gavin Green Jr., 47, Female', b'owner': b'Michael Williams, 37, Fem
ale'}
Создано упорядоченное множество: engine_sizes = [(b'Toyota Model A', 1.4), (b'Ford Model C', 1.5), (b'Ford Model A', 1.8), (b'Honda Model C', 1.8), (b'Toyota Model B', 1.9),
(b'Chevrolet Model A', 2.3), (b'Chevrolet Model C', 2.6), (b'Ford Model B', 2.7), (b'BMW Model A', 3.1), (b'BMW Model B', 3.3), (b'Honda Model A', 3.4), (b'Toyota Model C',
3.7), (b'Chevrolet Model B', 3.8), (b'Honda Model B', 4.5), (b'BMW Model C', 4.6)]
```

Рис. 23

Проверка загрузки данных в Redis Commander (Рис. 24).

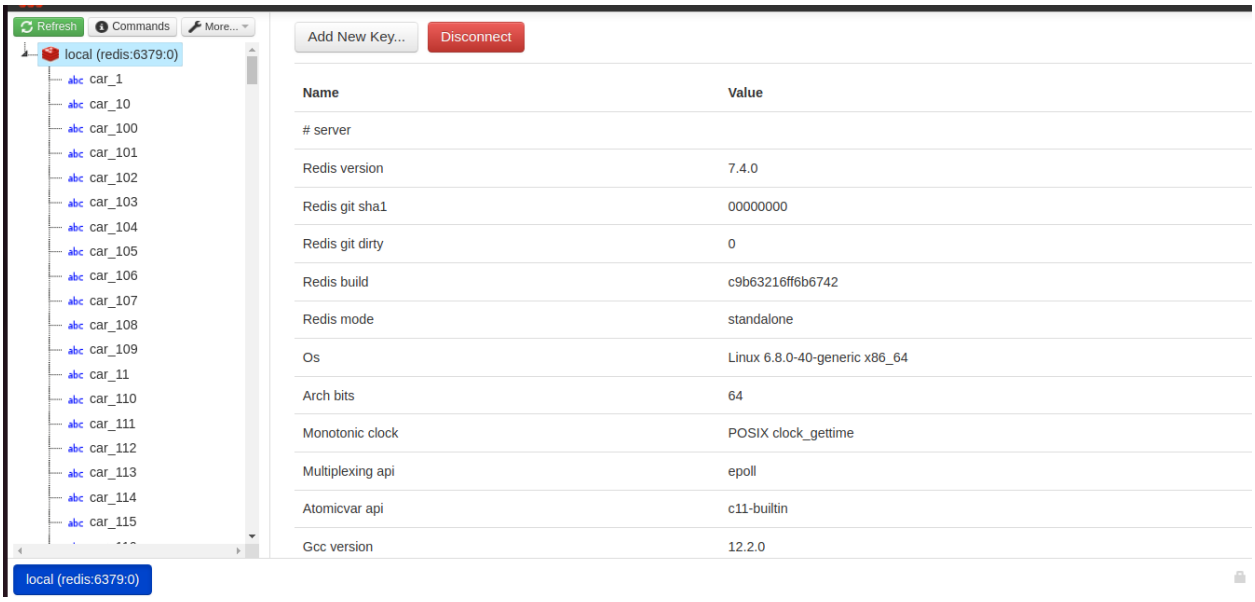


Рис. 24

Получение данных (Рис. 25).

```python
# Получение данных о машинах и владельцах
print_separator("Получение данных о машинах и владельцах")

# Получаем автомобили и их детали
for i in range(1, 6):
    # Получаем строку (название автомобиля)
    car = r.get(f"car_{i}")
    print(f"Автомобиль {i}: {car.decode('utf-8') if car else 'Не найден'}")

    # Получаем список владельцев
    owners = r.lrange(f"owners_{i}", 0, -1)
    owners_list = [owner.decode('utf-8') for owner in owners]
    print(f"Владельцы {i}: {owners_list if owners else 'Не найдено'}")

    # Получаем множество услуг
    services = r.smembers(f"services_{i}")
    services_set = {service.decode('utf-8') for service in services}
    print(f"Услуги {i}: {services_set if services else 'Не найдено'}")

    # Получаем хэш с деталями автомобиля
    details = r.hgetall(f"details_{i}")
    details_dict = {key.decode('utf-8'): value.decode('utf-8') for key, value in details.items()}
    print(f"Детали {i}: {details_dict if details else 'Не найдено'}")

# Получаем упорядоченное множество по объему двигателя
engine_sizes = r.zrange(f"engine_sizes", 0, -1, withscores=True)
engine_sizes_list = [(car.decode('utf-8'), score) for car, score in engine_sizes]
print(f"Упорядоченное множество по объему двигателя: {engine_sizes_list if engine_sizes else 'Не найдено'}")

print_separator("Получение завершено")
```

```
==================== Получение данных о машинах и владельцах ====================
Автомобиль 1: Honda Model A
Владельцы 1: ['Christian Jones, 29, Female', 'Christopher Vargas, 46, Female']
Услуги 1: {'service_1', 'service_2', 'service_3'}
Детали 1: {'brand': 'Honda', 'model': 'Model A', 'engine_size': '3.4', 'ownre': 'Christian Jones, 29, Female', 'owner': 'Christopher Vargas, 46, Female'}
Автомобиль 2: BMW Model B
Владельцы 2: ['Jerome Barnes, 80, Female', 'Matthew Williams, 74, Female']
Услуги 2: {'service_1', 'service_2', 'service_3'}
Детали 2: {'brand': 'BMW', 'model': 'Model B', 'engine_size': '3.3', 'ownre': 'Jerome Barnes, 80, Female', 'owner': 'Matthew Williams, 74, Female'}
Автомобиль 3: Toyota Model C
```

Рис. 25

Обновление данных (Рис. 26).

```python
# Обновление данных
print_separator("Обновление данных о машинах и владельцах")

# Обновление строки
r.set("car_1", "New Toyota Camry")
print(f"Обновлено название автомобиля: car_1 = {r.get('car_1')}")

# Обновление списка владельцев
r.lset("owners_1", 0, "Игорь Сидоров")  # Можем заменить первого владельца
print(f"Обновлен список владельцев: owners_1 = {r.lrange('owners_1', 0, -1)}")

# Обновление множества услуг
r.sadd("services_1", "Новое ТО")  # Добавляем новую услугу
print(f"Обновлено множество услуг: services_1 = {r.smembers('services_1')}")

# Обновление хэша с деталями автомобиля
r.hset("details_1", "цвет", "красный")  # Обновляем цвет автомобиля
print(f"Обновлены детали автомобиля: details_1 = {r.hgetall('details_1')}")

# Обновление упорядоченного множества по объему двигателя
r.zadd("engine_sizes", {"New Toyota Camry": 2.5})  # Обновляем объем двигателя
print(f"Обновлено упорядоченное множество (объемы двигателей): engine_sizes = {r.zrange('engine_sizes', 0, -1, withscores=True)}")

print_separator("Обновление завершено")
```

```
==================== Обновление данных о машинах и владельцах ====================
Обновлено название автомобиля: car_1 = b'New Toyota Camry'
Обновлен список владельцев: owners_1 = [b'\xd0\x98\xd0\xb3\xd0\xbe\xd1\x80\xd1\x8c \xd0\xa1\xd0\xb8\xd0\xb4\xd0\xbe\xd1\x80\xd0\xbe\xd0\xb2', b'Christopher Vargas, 46, Female']
Обновлено множество услуг: services_1 = {b'\xd0\x9d\xd0\xbe\xd0\xb2\xd0\xbe\xd0\xb5 \xd0\xa2\xd0\x9e', b'\xd0\x97\xd0\xb0\xd0\xbc\xd0\xb5\xd0\xbd\xd0\xb0 \xd0\xbc\xd0\xb0\xd1\x81\xd0\xbb\xd0\xb0', b'\xd0\x9f\xd1\x80\xd0\xbe\xd0\xb2\xd0\xb5\xd1\x80\xd0\xba\xd0\xb0 \xd1\x82\xd0\xbe\xd1\x80\xd0\xbc\xd0\xbe\xd0\xb7\xd0\xbe\xd0\xb2'}
Обновлены детали автомобиля: details_1 = {b'brand': b'Honda', b'model': b'Model A', b'engine_size': b'3.4', b'ownre': b'Christian Jones, 29, Female', b'owner': b'Christopher Vargas, 46, Female', b'\xd0\xb3\xd0\xbe\xd0\xb4': b'2023', b'\xd1\x86\xd0\xb2\xd0\xb5\xd1\x82': b'\xd0\xba\xd1\x80\xd0\xb0\xd1\x81\xd0\xbd\xd1\x8b\xd0\xb9', b'new_member': b'\xd0\x9d\xd0\xbe\xd0\xb2\xd1\x8b\xd0\xb9 \xd1\x87\xd0\xbb\xd0\xb5\xd0\xbd'}
Обновлено упорядоченное множество (объемы двигателей): engine_sizes = [(b'Chevrolet Model B', 1.8), (b'Honda Model A', 1.9), (b'Toyota Model A', 2.0), (b'Chevrolet Model A', 2.4), (b'Ford Model A', 2.4), (b'Honda Model C', 2.4), (b'New Toyota Camry', 2.5), (b'\xd0\x9d\xd0\xbe\xd0\xb2\xd1\x8b\xd0\xb9 Toyota Camry', 2.5), (b'Chevrolet Model C', 2.6), (b'BMW Model B', 2.9), (b'Ford Model B', 3.3), (b'Honda Model B', 3.4), (b'Ford Model C', 3.7), (b'Toyota Model C', 4.1), (b'BMW Model A', 4.2), (b'BMW Model C', 4.4), (b'Toyota Model B', 4.6), (b'\xd0\x9d\xd0\xbe\xd0\xb2\xd1\x8b\xd0\xb9 \xd1\x87\xd0\xbb\xd0\xb5\xd0\xbd', 5.0)]

==================== Обновление завершено ====================
```

Рис. 26

Удаление данных (Рис. 27).

```
: # Функция для разделителя в выводе
def print_separator(message):
    print(f"\n{'-' * 30}\n{message}\n{'-' * 30}")

# Удаление данных
print_separator("Удаление данных")

# Список ключей для удаления
keys_to_delete = ["string_5", "list_5", "set_5", "hash_5", "zset_5"]

# Удаление указанных ключей
r.delete(*keys_to_delete)

# Сообщение об успешном удалении ключей
print(f"Удалены ключи: {', '.join(keys_to_delete)}")
```

```
------------------------------
Удаление данных
------------------------------
Удалены ключи: string_5, list_5, set_5, hash_5, zset_5
```

Рис. 27

Проверка удаления данных (Рис. 28).

```
print_separator("Проверка удаленных данных")
for key in ["string_5", "list_5", "set_5", "hash_5", "zset_5"]:
    print(f"Существует ли ключ {key}? {r.exists(key)}")
```

```
------------------------------
Проверка удаленных данных
------------------------------
Существует ли ключ string_5? 0
Существует ли ключ list_5? 0
Существует ли ключ set_5? 0
Существует ли ключ hash_5? 0
Существует ли ключ zset_5? 0
```

Рис. 28

```
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)
```

Рис. 29

Выгрузка данных в CSV файл (Рис. 31, Рис. 32).

```python
def dump_redis_to_csv(filename='redis_dump.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. 30

```python
# Выполнение выгрузки
dump_redis_to_csv()
```

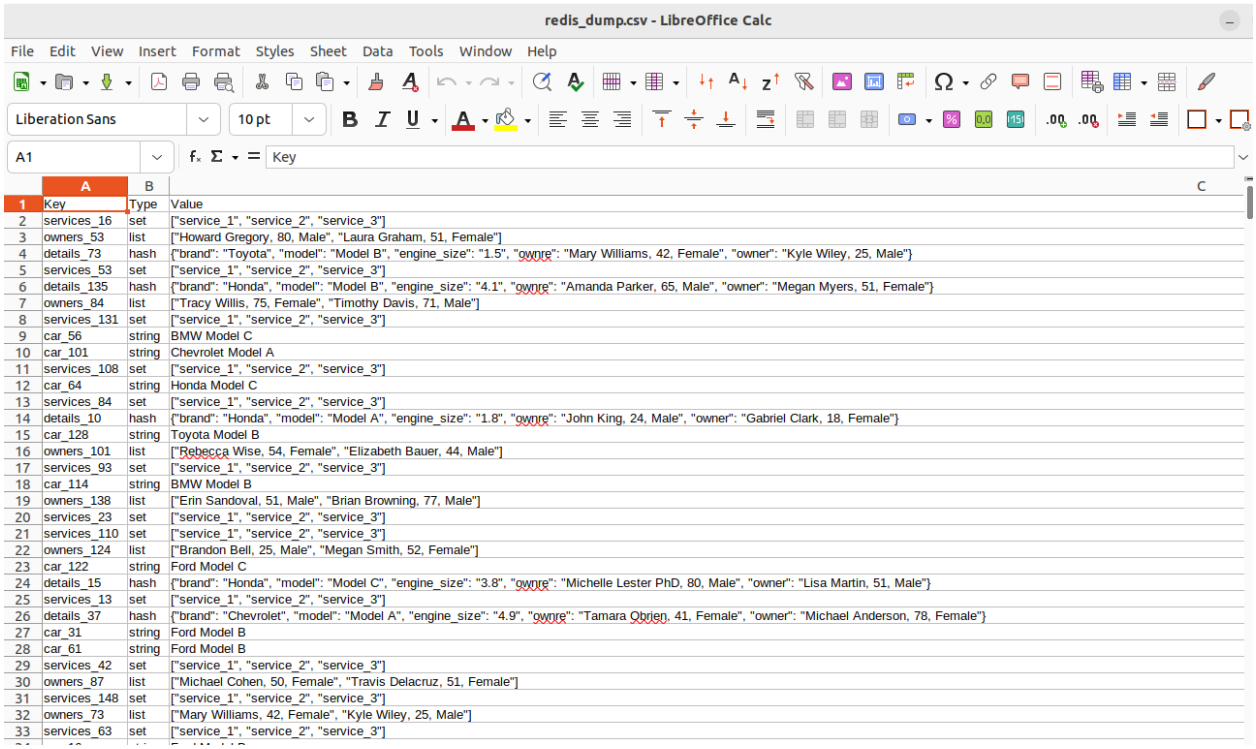
```
Данные сохранены в файл 'redis_dump.csv'
```

```
ls
```

```
 2-1.pdf              'lab 2-1 (1).ipynb'   mongodb-compass_1.43.6_amd64.deb
 MongoDB_Redis.ipynb  'lab 2-1.ipynb'       redis_dump.csv
```

Рис. 31

Проверка выгруженных данных (Рис. 32).



	A	B	C
1	Key	Type	Value
2	services_16	set	["service_1", "service_2", "service_3"]
3	owners_53	list	["Howard Gregory, 80, Male", "Laura Graham, 51, Female"]
4	details_73	hash	{"brand": "Toyota", "model": "Model B", "engine_size": "1.5", "owner": "Mary Williams, 42, Female", "owner": "Kyle Wiley, 25, Male"}
5	services_53	set	["service_1", "service_2", "service_3"]
6	details_135	hash	{"brand": "Honda", "model": "Model B", "engine_size": "4.1", "owner": "Amanda Parker, 65, Male", "owner": "Megan Myers, 51, Female"}
7	owners_84	list	["Tracy Willis, 75, Female", "Timothy Davis, 71, Male"]
8	services_131	set	["service_1", "service_2", "service_3"]
9	car_56	string	BMW Model C
10	car_101	string	Chevrolet Model A
11	services_108	set	["service_1", "service_2", "service_3"]
12	car_64	string	Honda Model C
13	services_84	set	["service_1", "service_2", "service_3"]
14	details_10	hash	{"brand": "Honda", "model": "Model A", "engine_size": "1.8", "owner": "John King, 24, Male", "owner": "Gabriel Clark, 18, Female"}
15	car_128	string	Toyota Model B
16	owners_101	list	["Rebecca Wise, 54, Female", "Elizabeth Bauer, 44, Male"]
17	services_93	set	["service_1", "service_2", "service_3"]
18	car_114	string	BMW Model B
19	owners_138	list	["Erin Sandoval, 51, Male", "Brian Browning, 77, Male"]
20	services_23	set	["service_1", "service_2", "service_3"]
21	services_110	set	["service_1", "service_2", "service_3"]
22	owners_124	list	["Brandon Bell, 25, Male", "Megan Smith, 52, Female"]
23	car_122	string	Ford Model C
24	details_15	hash	{"brand": "Honda", "model": "Model C", "engine_size": "3.8", "owner": "Michelle Lester PhD, 80, Male", "owner": "Lisa Martin, 51, Male"}
25	services_13	set	["service_1", "service_2", "service_3"]
26	details_37	hash	{"brand": "Chevrolet", "model": "Model A", "engine_size": "4.9", "owner": "Tamara Obrien, 41, Female", "owner": "Michael Anderson, 78, Female"}
27	car_31	string	Ford Model B
28	car_61	string	Ford Model B
29	services_42	set	["service_1", "service_2", "service_3"]
30	owners_87	list	["Michael Cohen, 50, Female", "Travis Delacruz, 51, Female"]
31	services_148	set	["service_1", "service_2", "service_3"]
32	owners_73	list	["Mary Williams, 42, Female", "Kyle Wiley, 25, Male"]
33	services_63	set	["service_1", "service_2", "service_3"]

Рис. 32

3. Задание в Neo4j

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.

CREATE (C:course{name:'Discrete Mathematics'}), (S:course{name:'Databases'}), (I:course{name:'Data Processing'}), (E:person{name:'Elena'}), (N:person{name:'Natalia'}), (V:person{name:'Victoria'}), (O:person{name:'Olga'}), (St:person{name:'Stas'}), (A:person{name:'Andrey'}), (D:person{name:'Dan'}), (Al:person{name:'Alex'}), (Dm:person{name:'Dmitry'}), (K:person{name:'Katarina'}), (Elena:student{name:'Teresa'}), (Nina:student{name:'Nina'}), (Victor:student{name:'Victor'}), (Olga:student{name:'Olga'}), (Stas:student{name:'Stas'}), (Anna:student{name:'Elen'}), (Denis:student{name:'Denis'}), (Alexandr:student{name:'Alexandr'}), (Dmitry:student{name:'Dmitry'}), (Konstantin:student{name:'Konstantin'}),

(Roman:student{name:'Roman'}), (Ilia:student{name:'Ilia'}), (E)-[:author]->(C), (N)-[:author]->(C), (A)-[:author]->(S), (A)-[:author]->(I), (Al)-[:author]->(I), (Dm)-[:author]->(I), (O)-[:speaker]->(C), (St)-[:editor]->(C), (V)-[:designer]->(C), (D)-[:speaker]->(S), (St)-[:editor]->(S), (V)-[:designer]->(S), (D)-[:speaker]->(I), (St)-[:editor]->(I), (K)-[:designer]->(I), (Elena)-[:learn]->(I), (Elena)-[:learn]->(C), (Nina)-[:learn]->(S), (Nina)-[:learn]->(C), (Victor)-[:learn]->(S), (Victor)-[:learn]->(C), (Olga)-[:learn]->(S), (Olga)-[:learn]->(C), (Stas)-[:learn]->(I), (Stas)-[:learn]->(C), (Stas)-[:learn]->(S), (Anna)-[:learn]->(I), (Anna)-[:learn]->(C), (Anna)-[:learn]->(S), (Denis)-[:learn]->(I), (Alexandr)-[:learn]->(C), (Alexandr)-[:learn]->(S), (Dmitry)-[:learn]->(I), (Dmitry)-[:learn]->(C), (Konstantin)-[:learn]->(S), (Roman)-[:learn]->(C), (Ilia)-[:learn]->(C)

Создание данных (Рис. 33).

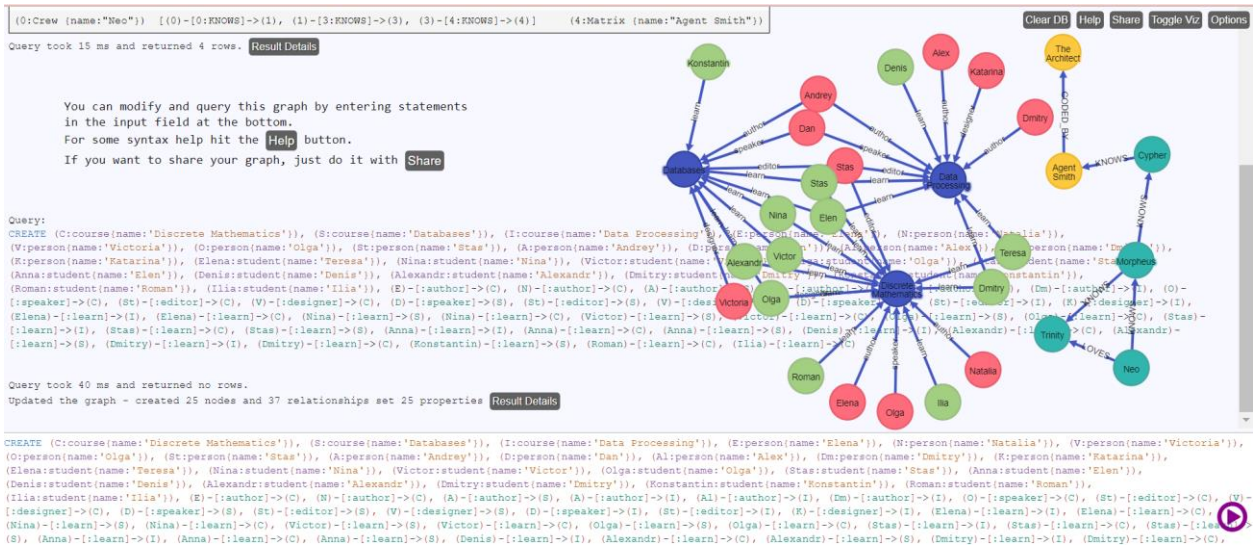


Рис. 33

Задание № 9: Составьте запрос, который вернет список студентов, обучающихся на курсе "Data Processing", и перечислите авторов этого курса.

Запрос на получение списка студентов, обучающихся на курсе "Data Processing", и перечисление авторов этого курса:

MATCH (s:student)-[:learn]->(c:course {name: 'Data Processing'}),

3 (a:person)-[:author]->(c)

4RETURN s.name AS StudentName, collect(a.name) AS Authors

Пояснение запроса:

MATCH находим студентов (**s:student**), которые имеют отношение **learn** к курсу **c:course** с именем "Data Processing".

Также находим авторов (**a:person**), которые имеют отношение **author** к тому же курсу.

В **RETURN** возвращаем имена студентов и собираем имена авторов в список с помощью функции **collect()**, чтобы отобразить их в одной строке.

Результат запроса (Рис. 34).

Detailed Query Results

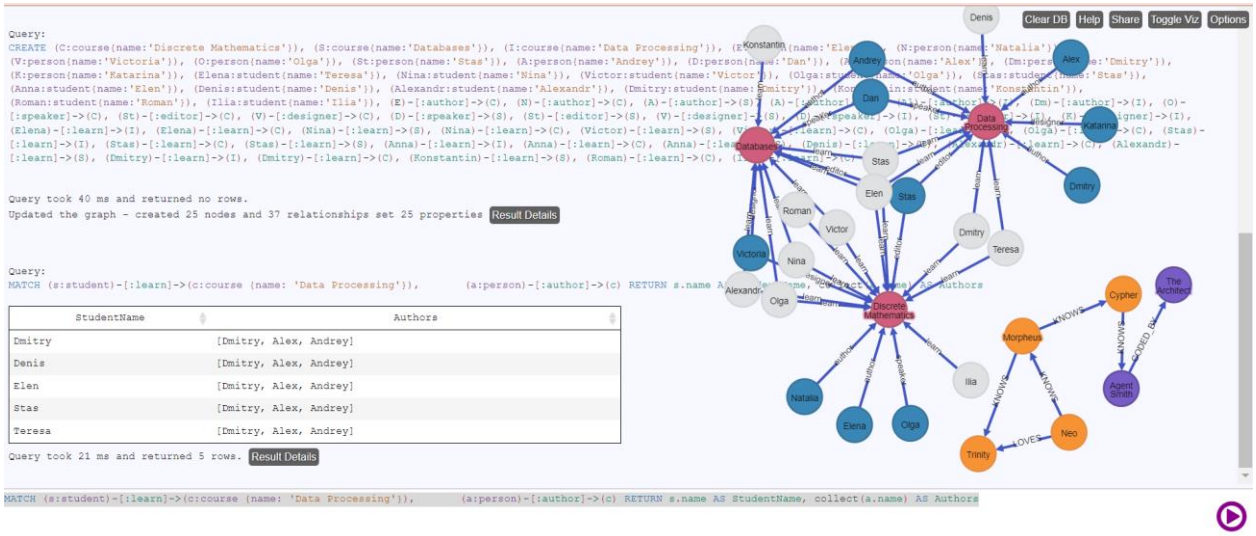
Query Results

StudentName	Authors
"Dmitry"	["Dmitry","Alex","Andrey"]
"Denis"	["Dmitry","Alex","Andrey"]
"Elen"	["Dmitry","Alex","Andrey"]
"Stas"	["Dmitry","Alex","Andrey"]
"Teresa"	["Dmitry","Alex","Andrey"]

5 rows
21 ms

Рис. 34

5 студентов обучаются на курсе «Data Processing» : Дмитрий, Денис, Елен, Стас, Тереза. Авторы на курсе одинаковые у всех студентов: Дмитрий, Алекс, Андрей. Скорость выполнения запроса - 21 мсек.



Сравнение используемых СУБД в лабораторной работе представлено в таблице 1.

Таблица 1

СУБД	Основные характеристики	Преимущества	Недостатки
MongoDB	Документо-ориентированная база данных, использующая JSON-подобные документы.	- Гибкость структуры данных - Высокая масштабируемость - Поддержка агрегаций и индексов	- Ограниченная поддержка транзакций - Меньшая производительность по сравнению с реляционными СУБД для сложных запросов
Redis	Ключ-значение, высокопроизводительная база данных в памяти.	- Очень высокая скорость доступа - Поддержка сложных структур данных (списки, множества)	- Ограничение на объем данных (в памяти) - Не подходит для сложных запросов и аналитики
Neo4j	Графовая база данных, основанная на узлах и связях.	- Эффективное представление сложных взаимосвязей - Высокая скорость обработки графовых запросов	- Меньшая популярность и поддержка - Может быть сложной в использовании при простых задачах

Вывод: каждая СУБД обладает своими особенностями и использование конкретной СУБД зависит от поставленной задачи, если необходимо хранить временные данные, то идеально подойдёт Redis, в MongoDB гибкая структура, которая идеально подойдёт для аналитики, Neo4j идеально подойдёт для сложных запросов и выявлений взаимосвязей.