

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:
Инструменты для хранения и обработки больших данных
Лабораторная работа № 2.1

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

Выполнил(а):
Вереина М.С., АДЭУ-211
Преподаватель: Босенко Т.М.

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 JSON-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).
- JSON-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 кэширования 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

Необходимо продемонстрировать выполнение базовых операций (вставка, выборка, обновление, удаление) для СУБД MongoDB и Redis.

Вариант 3. Книги и авторы в библиотеке.

Часть 1. MongoDB.

Шаг 1. Развернуть MongoDB с помощью докера в терминале Visual Studio Code

```
21     environment:
22         - ME_CONFIG_MONGODB_SERVER=mongodb
23         - ME_CONFIG_MONGODB_ADMINUSERNAME=mongouser
24         - ME_CONFIG_MONGODB_ADMINPASSWORD=mongopasswd
25         - ME_CONFIG_BASICAUTH_USERNAME=adminuser
26         - ME_CONFIG_BASICAUTH_PASSWORD=adminpasswd

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
• 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      Running
✓ Container mongo-express Running
○ nosql@nosql-vm:~/mongodb$
```

Рис. 1. Запуск MongoDB

Шаг 2. Сгенерировать данные на тему «Книги и авторы в библиотеке».

Для этого была использована библиотека Faker. Faker – это библиотека с открытым исходным кодом, созданная Франсуа Занинотто, которая позволяет генерировать случайные данные. С ее помощью было сгенерировано 1000 записей.

```
[ ] 1 import pandas as pd
    2 from faker import Faker
    3 import random

[ ] 1 fake = Faker()
    2
    3 # Список библиотек
    4 libraries = ["Library of Alexandria", "Garrison Library", "Library of the Academy of Sciences", "National Library of Great Britain",

[ ] 1 def generate_book_data(num_records):
    2     data = []
    3     for _ in range(num_records):
    4         book = {
    5             "title": fake.catch_phrase(), # Генерация названия книги
    6             "author": fake.name(),        # Генерация имени автора
    7             "year": random.randint(1900, 2023), # Генерация года создания книги
    8             "library": random.choice(libraries) # Случайный выбор библиотеки
    9         }
   10     data.append(book)
   11     return data
   12
   13 # Генерация 1000 записей
   14 books_data = generate_book_data(1000)
```

Рис. 2. Код python для генерации данных с помощью Faker

После генерации необходимо было создать датафрейм и проверить сгенерированные данные. Они оказались корректными, поэтому далее их необходимо сохранить в файл формата JSON, так как он наиболее удобен при работе с MongoDB и Redis. Аналогичным образом были сгенерированы и сохранены данные на 10 000 значений для сравнения.

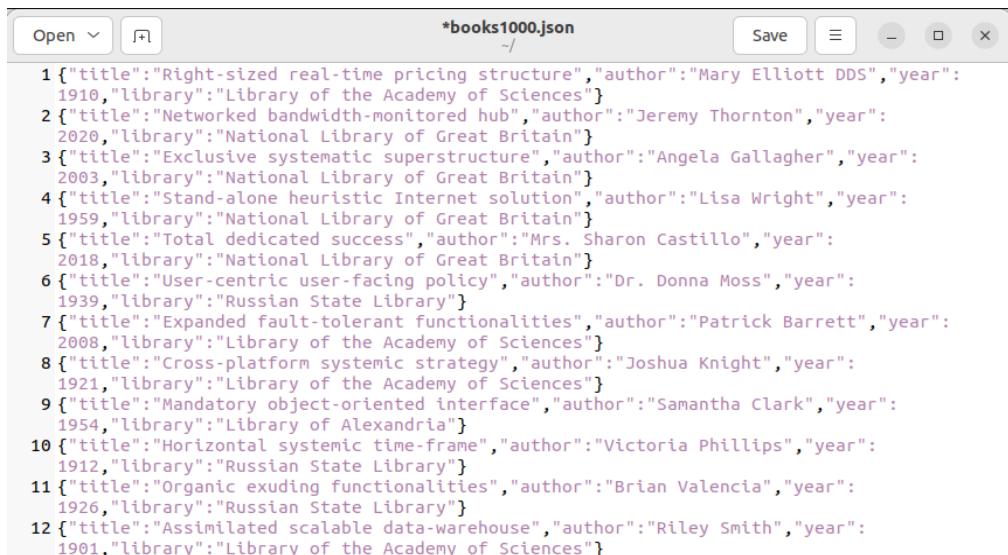
```
1 # Просмотр получившихся значений
2 books1000.head(5)
```

	title	author	year	library
0	Mandatory clear-thinking product	Tony Frederick	2017	Garrison Library
1	Pre-emptive bottom-line process improvement	Walter Jackson	1912	National Library of Great Britain
2	Organic scalable array	John Rosales	1972	Library of Alexandria
3	Triple-buffered intangible definition	Angela Flores	1968	Russian State Library
4	Optional fresh-thinking initiative	Robert Mason	1928	Library of the Academy of Sciences

```
1 # Создание DataFrame
2 books1000 = pd.DataFrame(books_data)
3
4 # Сохранение в JSON-файл
5 books1000.to_json('books1000.json', orient='records', lines=True)
```

Рис. 3. Проверка сгенерированных данных и их сохранение в JSON

Ниже приведен фрагмент файла JSON, который содержит сгенерированные данные и будет использован при работе с СУБД.



```

1 {"title": "Right-sized real-time pricing structure", "author": "Mary Elliott DDS", "year": 1910, "library": "Library of the Academy of Sciences"}
2 {"title": "Networked bandwidth-monitored hub", "author": "Jeremy Thornton", "year": 2020, "library": "National Library of Great Britain"}
3 {"title": "Exclusive systematic superstructure", "author": "Angela Gallagher", "year": 2003, "library": "National Library of Great Britain"}
4 {"title": "Stand-alone heuristic Internet solution", "author": "Lisa Wright", "year": 1959, "library": "National Library of Great Britain"}
5 {"title": "Total dedicated success", "author": "Mrs. Sharon Castillo", "year": 2018, "library": "National Library of Great Britain"}
6 {"title": "User-centric user-facing policy", "author": "Dr. Donna Moss", "year": 1939, "library": "Russian State Library"}
7 {"title": "Expanded fault-tolerant functionalities", "author": "Patrick Barrett", "year": 2008, "library": "Library of the Academy of Sciences"}
8 {"title": "Cross-platform systemic strategy", "author": "Joshua Knight", "year": 1921, "library": "Library of the Academy of Sciences"}
9 {"title": "Mandatory object-oriented interface", "author": "Samantha Clark", "year": 1954, "library": "Library of Alexandria"}
10 {"title": "Horizontal systemic time-frame", "author": "Victoria Phillips", "year": 1912, "library": "Russian State Library"}
11 {"title": "Organic exuding functionalities", "author": "Brian Valencia", "year": 1926, "library": "Russian State Library"}
12 {"title": "Assimilated scalable data-warehouse", "author": "Riley Smith", "year": 1901, "library": "Library of the Academy of Sciences"}

```

Рис. 4. Фрагмент JSON, содержащий сгенерированные данные

Шаг 3. Подготовка блокнота Jupiter для работы с СУБД. Для начала необходимо установить пакет pymongo, создать клиента и проверить, что подключение к MongoDB успешно установлено с помощью команды ping - это команда без операции, которая используется для проверки того, отвечает ли сервер на команды.



```

VereinaMS lab2-1.ipynb
File Edit View Run Kernel Tabs Settings Help

[1]: !pip install pymongo

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 json
from pymongo import MongoClient

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

[4]: client = MongoClient(mongo_uri)

client.admin.command('ping')
print("Подключение к MongoDB установлено успешно!")

Подключение к MongoDB установлено успешно!

```

Рис. 5. Проверка подключения к MongoDB

Шаг 4. Вставка сгенерированных данных. С помощью db создать новую базу данных по теме, далее создать коллекцию, это группа документов, она представляет собой аналог таблицы в реляционной базе данных. И проверить вставку тестовых данных. Вывод InsertOneResult подтверждает успешную вставку документов.

```
[5]: # Создание базы данных
db = client['books1000']

[6]: # Создание коллекции
collection = db['books1000']

[7]: # Пример вставки документа
books1000 = db['books1000']

# Вставка тестового документа
test_document = {"title": "Computer Science Today", "author": "Mary Jane", "year": 1980, "library": "Library of the Academy of Sciences"}
books1000.insert_one(test_document)

[7]: InsertOneResult(ObjectId('66f05f1fcefafa75595e91341'), acknowledged=True)
```

Рис. 6. Создание БД для 1000 записей. Вставка тестового документа.

Далее в среде выполнения необходимо загрузить файл JSON, который находится в домашней папке, с помощью определения пути его нахождения. После этого загрузить данный файл с помощью построчной вставки в коллекцию. Ниже и далее в ячейках используется метод %%time для измерения и сравнения затраченного времени.

```
[19]: import os

# Определить путь к папке и имя файла
folder_path = '/home/nosql'
file_name = 'books1000.json'

# Получение полного пути к файлу
file_path = os.path.join(folder_path, file_name)

[10]: %%time

# Загрузка данных из JSON-файла
with open(file_path, 'r') as file:
    for line in file:
        # Загружаем каждый объект по отдельности
        document = json.loads(line)
        collection.insert_one(document) # Вставка документа в коллекцию

CPU times: user 1.31 s, sys: 385 ms, total: 1.7 s
Wall time: 4.18 s
```

Рис. 7. Загрузка данных в коллекцию из файла JSON на 1000 строк.

Аналогичные действия необходимо совершить для файла, который содержит 10 000 сгенерированных записей. Создаем отдельную базу данных и коллекцию, которая будет их хранить.

```
[13]: # Создание базы данных
db = client['books10k']

[15]: # Создание коллекции
collection = db['books10k']

[16]: # Пример вставки документа
books10k = db['books10k']

# Вставка тестового документа
test_document = {"title": "Computer Science Today", "author": "Mary Jane", "year": 1980, "library": "Library of the Academy of Sciences"}
books10k.insert_one(test_document)

[16]: InsertOneResult(ObjectId('66f06420cefafa75595e9172a'), acknowledged=True)
```

Рис. 8. Создание БД и коллекции для 10 000 записей.

```
[20]: # Определить путь к папке и имя файла
      folder_path = '/home/nosql'
      file_name = 'books10k.json'

      # Получение полного пути к файлу
      file_path = os.path.join(folder_path, file_name)

[18]: %%time

      # Загрузка данных из JSON-файла
      with open(file_path, 'r') as file:
          for line in file:
              # Загружаем каждый объект по отдельности
              document = json.loads(line)
              collection.insert_one(document) # Вставка документа в коллекцию

      CPU times: user 14.3 s, sys: 4.35 s, total: 18.7 s
      Wall time: 48.8 s
```

Рис. 9. Загрузка данных в коллекцию из файла JSON на 10 000 строк.

Переходим на локальный адрес MongoDB, чтобы с помощью веб-интерфейса убедиться в том, что данные были корректно загружены. Таким образом, в MongoDB были созданы две базы данных, которые хранят загруженные данные.

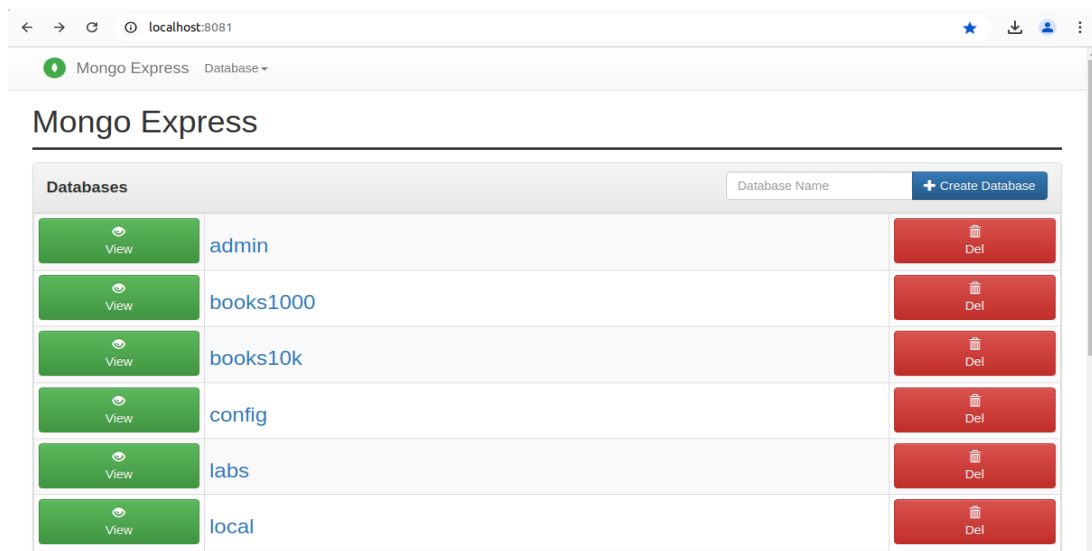


Рис. 10. Перечень баз данных в Mongo

Viewing Database: books1000

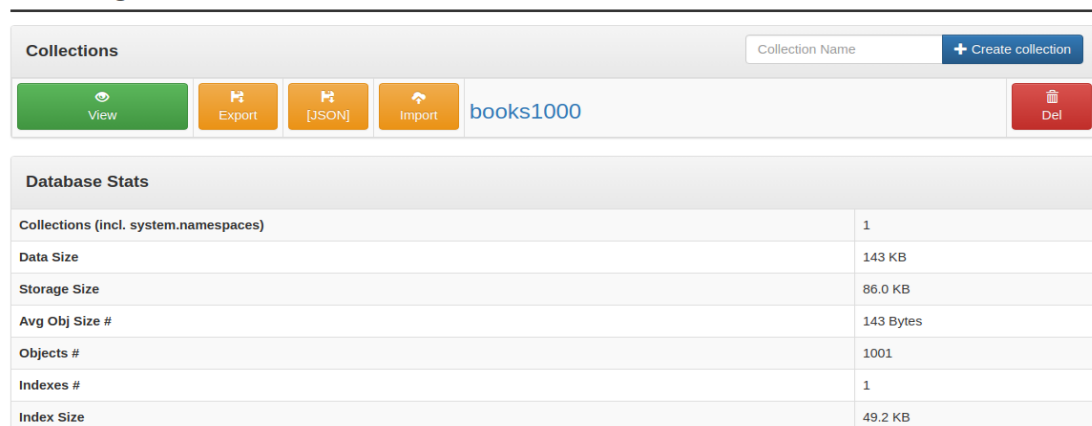


Рис. 11. Корректное количество объектов в БД для 1000 значений

Viewing Database: books10k

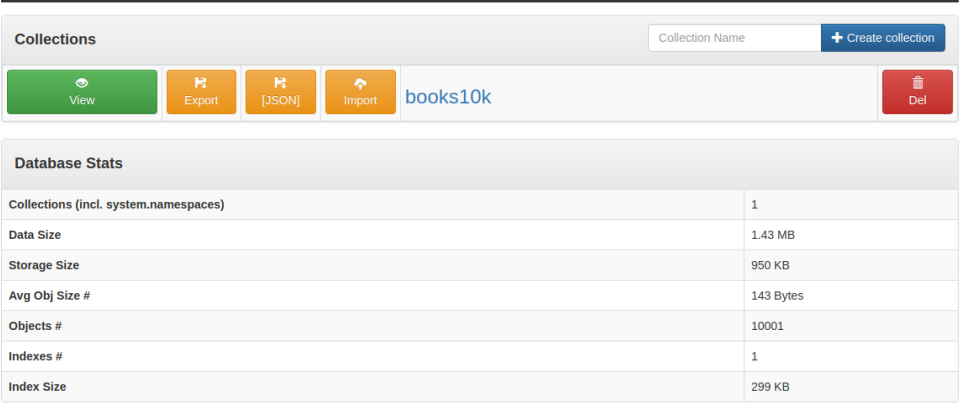


Рис. 12. Корректное количество объектов в БД для 10 000 значений.

Шаг 5. Выборка загруженных данных.

- 1) Чтобы получить все документы из коллекции, необходимо использовать метод find():

```
[21]: %%time
# Получение всех документов из коллекции
all_documents = books1000.find()

for doc in all_documents:
    print(doc)

{'_id': ObjectId('66f05f1fcef...'), 'title': 'Computer Science Today', 'author': 'Mary Jane', 'year': 1980, 'library': 'Library of the Academy of Sciences'}
{'_id': ObjectId('66f05f39cef...'), 'title': 'Right-sized real-time pricing structure', 'author': 'Mary Elliott DDS', 'year': 1910, 'library': 'Library of the Academy of Sciences'}
{'_id': ObjectId('66f05f39cef...'), 'title': 'Networked bandwidth-monitored hub', 'author': 'Jeremy Thornton', 'year': 2020, 'library': 'National Library of Great Britain'}
{'_id': ObjectId('66f05f39cef...'), 'title': 'Exclusive systematic superstructure', 'author': 'Angela Gallagher', 'year': 2003, 'library': 'National Library of Great Britain'}
{'_id': ObjectId('66f05f39cef...'), 'title': 'Stand-alone heuristic Internet solution', 'author': 'Lisa Wright', 'year': 1959, 'library': 'National Library of Great Britain'}
{'_id': ObjectId('66f05f39cef...'), 'title': 'Total dedicated success', 'author': 'Mrs. Sharon Castillo', 'year': 2018, 'library': 'National Library of Great Britain'}
{'_id': ObjectId('66f05f39cef...'), 'title': 'User-centric user-facing policy', 'author': 'Dr. Donna Moss', 'year': 1939, 'library': 'Russian State Library'}
{'_id': ObjectId('66f05f3dcef...'), 'title': 'Programmable multi-tasking flexibility', 'author': 'Sheryl Lowery', 'year': 1955, 'library': 'Garrison Library'}
{'_id': ObjectId('66f05f3dcef...'), 'title': 'Distributed incremental workforce', 'author': 'Kyle Benjamin', 'year': 2001, 'library': 'Russian State Library'}
{'_id': ObjectId('66f05f3dcef...'), 'title': 'Ergonomic heuristic Graphical User Interface', 'author': 'Julie Davis', 'year': 1962, 'library': 'Library of the Academy of Sciences'}
{'_id': ObjectId('66f05f3dcef...'), 'title': 'Ergonomic fresh-thinking structure', 'author': 'Madison Morris', 'year': 1942, 'library': 'Russian State Library'}
CPU times: user 64.9 ms, sys: 64.4 ms, total: 129 ms
Wall time: 846 ms
```

Рис. 13. Получение всех документов из коллекции на 1000 строк

```
[22]: %%time
# Получение всех документов из коллекции
all_documents = books10k.find()

for doc in all_documents:
    print(doc)

{'_id': ObjectId('66f06420cef...'), 'title': 'Computer Science Today', 'author': 'Mary Jane', 'year': 1980, 'library': 'Library of the Academy of Sciences'}
{'_id': ObjectId('66f06425cef...'), 'title': 'Reactive background Graphical User Interface', 'author': 'Lauren Stevens', 'year': 1946, 'library': 'Library of Alexandria'}
{'_id': ObjectId('66f06425cef...'), 'title': 'Focused explicit model', 'author': 'Catherine Dixon', 'year': 1955, 'library': 'National Library of Great Britain'}
{'_id': ObjectId('66f06425cef...'), 'title': 'Monitored full-range forecast', 'author': 'Christopher Mayer', 'year': 1945, 'library': 'Russian State Library'}
{'_id': ObjectId('66f06425cef...'), 'title': 'Distributed 24hour project', 'author': 'Kimberly Thomas', 'year': 1948, 'library': 'National Library of Great Britain'}
{'_id': ObjectId('66f06455cef...'), 'title': 'Cloned dynamic product', 'author': 'Felicia Brown', 'year': 1935, 'library': 'Library of Alexandria'}
{'_id': ObjectId('66f06455cef...'), 'title': 'Focused regional Local Area Network', 'author': 'Rachel Wright', 'year': 1915, 'library': 'Library of Alexandria'}
{'_id': ObjectId('66f06455cef...'), 'title': 'Adaptive full-range implementation', 'author': 'Mark Hamilton', 'year': 1989, 'library': 'National Library of Great Britain'}
{'_id': ObjectId('66f06455cef...'), 'title': 'Inverse foreground firmware', 'author': 'Hannah Wagner', 'year': 1969, 'library': 'Garrison Library'}
CPU times: user 521 ms, sys: 231 ms, total: 753 ms
Wall time: 1.91 s
```

Рис. 14. Получение всех документов из коллекции на 10 000 строк

- 2) Фильтрация по критериям. В MongoDB можно фильтровать данные, передавая параметры в метод `find()`. Например, чтобы найти документы с определенным автором необходимо использовать этот элемент через двоеточие с нужным значением.

```
[27]: %%time
# Фильтрация по автору
author_filter = {"author": "Brian White"}
filtered_documents = books1000.find(author_filter)

for doc in filtered_documents:
    print(doc)

{'_id': ObjectId('66f05f3acefaa75595e9139a'), 'title': 'Polarized foreground knowledge user', 'author': 'Brian White', 'year': 1951, 'library': 'Library of Alexandria'}
CPU times: user 5.62 ms, sys: 1.46 ms, total: 7.09 ms
Wall time: 26.3 ms

[31]: %%time
# Фильтрация по автору
author_filter = {"author": "Brian White"}
filtered_documents = books10k.find(author_filter)

for doc in filtered_documents:
    print(doc)

{'_id': ObjectId('66f06428cefaa75595e9195e'), 'title': 'Upgradable coherent firmware', 'author': 'Brian White', 'year': 1997, 'library': 'Russian State Library'}
{'_id': ObjectId('66f06440cefaa75595e92c03'), 'title': 'Cross-platform demand-driven hardware', 'author': 'Brian White', 'year': 1957, 'library': 'Garrison Library'}
CPU times: user 2.32 ms, sys: 6.83 ms, total: 9.15 ms
Wall time: 29.3 ms
```

Рис. 15. Фильтрация по автору в БД на 1000 и 10 000 документов

- 3) Ограничение количества возвращаемых документов. В MongoDB можно ограничить количество возвращаемых документов с помощью метода `limit()`, например, чтобы получить первые 5 документов коллекции.

```
[33]: %%time

# Получение первых 5 документов
limit_docs = books1000.find().limit(5)

for doc in limit_docs:
    print(doc)

{'_id': ObjectId('66f05f1f3cefaa75595e91341'), 'title': 'Computer Science Today', 'author': 'Mary Jane', 'year': 1980, 'library': 'Library of the Academy of Sciences'}
{'_id': ObjectId('66f05f39cefaa75595e91342'), 'title': 'Right-sized real-time pricing structure', 'author': 'Mary Elliott DDS', 'year': 1910, 'library': 'Library of the Academy of Sciences'}
{'_id': ObjectId('66f05f39cefaa75595e91343'), 'title': 'Networked bandwidth-monitored hub', 'author': 'Jeremy Thornton', 'year': 2020, 'library': 'National Library of Great Britain'}
{'_id': ObjectId('66f05f39cefaa75595e91344'), 'title': 'Exclusive systematic superstructure', 'author': 'Angela Gallagher', 'year': 2003, 'library': 'National Library of Great Britain'}
{'_id': ObjectId('66f05f39cefaa75595e91345'), 'title': 'Stand-alone heuristic Internet solution', 'author': 'Lisa Wright', 'year': 1959, 'library': 'National Library of Great Britain'}
CPU times: user 2.91 ms, sys: 3.69 ms, total: 6.6 ms
Wall time: 14.4 ms
```

Рис. 16. Получение первых 5 документов из 1000

```
[34]: %%time

# Получение первых 5 документов
limit_docs = books10k.find().limit(5)

for doc in limit_docs:
    print(doc)

{'_id': ObjectId('66f06420cefaa75595e9172a'), 'title': 'Computer Science Today', 'author': 'Mary Jane', 'year': 1980, 'library': 'Library of the Academy of Sciences'}
{'_id': ObjectId('66f06425cefaa75595e9172b'), 'title': 'Reactive background Graphical User Interface', 'author': 'Lauren Stevens', 'year': 1946, 'library': 'Library of Alexandria'}
{'_id': ObjectId('66f06425cefaa75595e9172c'), 'title': 'Focused explicit model', 'author': 'Catherine Dixon', 'year': 1955, 'library': 'National Library of Great Britain'}
{'_id': ObjectId('66f06425cefaa75595e9172d'), 'title': 'Monitored full-range forecast', 'author': 'Christopher Mayer', 'year': 1945, 'library': 'Russian State Library'}
{'_id': ObjectId('66f06425cefaa75595e9172e'), 'title': 'Distributed 24hour project', 'author': 'Kimberly Thomas', 'year': 1948, 'library': 'National Library of Great Britain'}
CPU times: user 2.8 ms, sys: 914 µs, total: 3.72 ms
Wall time: 10.9 ms
```

Рис. 17. Получение первых 5 документов из 10 000

Шаг 6. Обновление загруженных данных.

- 1) Обновление одного документа. Чтобы обновить один документ, можно использовать метод `update_one()`. Например, если нужно обновить автора книги с определенным названием.

```
[35]: %%time
# Условие для поиска документа
query = {"title": "Cloned executive strategy"}

# Данные для обновления
new_values = {"$set": {"author": "Alice Verben"}}

# Выполнение обновления
result = books1000.update_one(query, new_values)

print(f"Количество обновленных документов: {result.modified_count}")

Количество обновленных документов: 1
CPU times: user 344 µs, sys: 11.7 ms, total: 12 ms
Wall time: 107 ms

[37]: %%time
# Условие для поиска документа
query = {"title": "Phased systemic structure"}

# Данные для обновления
new_values = {"$set": {"author": "Alice Verben"}}

# Выполнение обновления
result = books10k.update_one(query, new_values)

print(f"Количество обновленных документов: {result.modified_count}")

Количество обновленных документов: 1
CPU times: user 2.61 ms, sys: 2.42 ms, total: 5.02 ms
Wall time: 18.8 ms
```

Рис. 18. Обновление одного документа из 1000 и 10 000 документов

- 2) Обновление нескольких документов. Если нужно обновить несколько документов, используется метод `update_many()`. Например, чтобы изменить год для всех книг определенного автора.

```
[39]: %%time

# Условие для поиска документов
query = {"author": "James Hamilton"}

# Данные для обновления
new_values = {"$set": {"year": 2020}}

# Выполнение обновления
result = books1000.update_many(query, new_values)

# Вывод количества обновленных документов
print(f"Количество обновленных документов: {result.modified_count}")

Количество обновленных документов: 2
CPU times: user 0 ns, sys: 3.58 ms, total: 3.58 ms
Wall time: 17.4 ms
```

Рис. 19. Обновление данных для всех годов книги одного автора из 1000 документов

```
[40]: %%time

# Условие для поиска документов
query = {"author": "Brian White"}

# Данные для обновления
new_values = {"$set": {"year": 2020}}

# Выполнение обновления
result = books10k.update_many(query, new_values)

# Вывод количества обновленных документов
print(f"Количество обновленных документов: {result.modified_count}")

Количество обновленных документов: 2
CPU times: user 5.58 ms, sys: 4.09 ms, total: 9.67 ms
Wall time: 46 ms
```

Рис. 20. Обновление данных для всех годов книги одного автора из 10 000 документов

- 3) Использование модификаторов. В MongoDB можно использовать модификаторы для массового изменения значений. Например, чтобы увеличить год на 1 для всех книг.

```
[41]: %%time
# Условие для поиска документов
query = {}

# Данные для обновления с использованием модификатора $inc
new_values = {"$inc": {"year": 1}}

# Выполнение обновления
result = books1000.update_many(query, new_values)

# Вывод количества обновленных документов
print(f"Количество обновленных документов: {result.modified_count}")

Количество обновленных документов: 1001
CPU times: user 2.8 ms, sys: 3.57 ms, total: 6.36 ms
Wall time: 137 ms
```

Рис. 21. Увеличение всех записей года на 1 для 1000 значений

```
[42]: %%time
# Условие для поиска документов
query = {}

# Данные для обновления с использованием модификатора $inc
new_values = {"$inc": {"year": 1}}

# Выполнение обновления
result = books10k.update_many(query, new_values)

# Вывод количества обновленных документов
print(f"Количество обновленных документов: {result.modified_count}")

Количество обновленных документов: 10001
CPU times: user 5.46 ms, sys: 7.71 ms, total: 13.2 ms
Wall time: 1.16 s
```

Рис. 22. Увеличение всех записей года на 1 для 10 000 значений

Шаг 7. Удаление и восстановление документов.

При удалении документа в MongoDB можно сначала сохранить его в отдельной коллекции для возможного восстановления. Для этого нужно создать новую коллекцию и вставить в нее удаляемый документ.

```
[47]: %%time

deleted_col = db['deleted_col'] # Коллекция для хранения удаленных документов

def delete_doc(query):
    # Найти документ по запросу
    doc = books1000.find_one(query)
    if doc:
        # Сохранить документ в коллекцию удаленных перед удалением
        deleted_col.insert_one(doc)
        # Удалить документ из основной коллекции
        books1000.delete_one(query)
        print("Документ удален и сохранен в коллекции удаленных.")
    else:
        print("Документ не найден.")

# Пример удаления документа
delete_query = {"title": "Open-source discrete conglomeration"}
delete_doc(delete_query)

Документ удален и сохранен в коллекции удаленных.
CPU times: user 4.77 ms, sys: 15.3 ms, total: 20.1 ms
Wall time: 349 ms
```

Рис. 23. Удаление документа из основной коллекции на 1000 значений

Для восстановления удаленного документа нужно извлечь его из коллекции удаленных и вставить обратно в основную коллекцию.

```
[48]: %%time

def restore_document(query):
    # Найти документ в коллекции удаленных
    doc = deleted_col.find_one(query)
    if doc:
        # Вставить документ обратно в основную коллекцию
        books1000.insert_one(doc)
        # Удалить документ из коллекции удаленных
        deleted_col.delete_one(query)
        print("Документ восстановлен.")
    else:
        print("Документ не найден в коллекции удаленных.")

# Пример восстановления документа
restore_query = {"title": "Open-source discrete conglomeration"}
restore_document(restore_query)

Документ восстановлен.
CPU times: user 6.99 ms, sys: 6.52 ms, total: 13.5 ms
Wall time: 38.2 ms
```

Рис. 24. Восстановление удаленного документа в коллекцию 1000 документов

Аналогичные действия необходимо применить и для базы данных, которая содержит 10 000 значений.

```
[55]: %%time

deleted_col = db['deleted_col'] # Коллекция для хранения удаленных документов

def delete_doc(query):
    # Найти документ по запросу
    doc = books10k.find_one(query)
    if doc:
        # Сохранить документ в коллекцию удаленных перед удалением
        deleted_col.insert_one(doc)
        # Удалить документ из основной коллекции
        books10k.delete_one(query)
        print("Документ удален и сохранен в коллекции удаленных.")
    else:
        print("Документ не найден.")

# Пример удаления документа
delete_query = {"title": "Innovative secondary portal"}
delete_doc(delete_query)

Документ удален и сохранен в коллекции удаленных.
CPU times: user 0 ns, sys: 8.2 ms, total: 8.2 ms
Wall time: 120 ms
```

Рис. 25. Удаление документа из основной коллекции на 10 000 значений

```
[56]: %%time

def restore_document(query):
    # Найти документ в коллекции удаленных
    doc = deleted_col.find_one(query)
    if doc:
        # Вставить документ обратно в основную коллекцию
        books10k.insert_one(doc)
        # Удалить документ из коллекции удаленных
        deleted_col.delete_one(query)
        print("Документ восстановлен.")
    else:
        print("Документ не найден в коллекции удаленных.")

# Пример восстановления документа
restore_query = {"title": "Innovative secondary portal"}
restore_document(restore_query)

Документ восстановлен.
CPU times: user 6.28 ms, sys: 1.31 ms, total: 7.59 ms
Wall time: 28.7 ms
```

Рис. 26. Восстановление удаленного документа в коллекцию на 10 000 документов

Сравнение времени проводимых операций в СУБД MongoDB

Кол-во документов	Загрузка из файла	Получение всех док-ов	Фильтрация по значению	Получение топ 5 док-тов	Обновление одного док-та	Обновление нескольких док-тов	Обновление модификатором	Удаление и сохранение	Восстановление
1000	4.18 с	846 мс	26.3 мс	14.4 мс	10.7 мс	17.4 мс	137 мс	349 мс	38.2 мс
10 000	48.8 с	1.91 с	29.3 мс	10.9 мс	18.8 мс	46 мс.	1.16 с	120 мс	28.7 мс

Выводы по работе с MongoDB.

1) Время выполнения операций значительно увеличивается с ростом количества документов. Например, время загрузки из файла увеличивается с 4.18 секунд до 48.8 секунд при увеличении количества документов с 1000 до 10,000. Это означает, что производительность системы может снижаться при больших объемах данных.

2) Для большинства операций время выполнения остается на уровне миллисекунд (например, фильтрация по значению и получение топ-5 документов), но некоторые операции, такие как загрузка из файла и обновление нескольких документов, показывают значительное увеличение времени выполнения. Это может указывать на необходимость оптимизации данных.

Рекомендации по улучшению работы с СУБД

1) Использование пакетных операций для обновления или удаления нескольких документов одновременно, чтобы уменьшить нагрузку на базу данных.

2) Необходимо улучшение обработки загрузки. Необходимо оптимизировать процесс загрузки данных, возможно, используя параллельные загрузки или более эффективные форматы данных для импорта.

MongoDB, как документно-ориентированная СУБД, успешно справилась с хранением JSON-данных, сгруппированных в коллекции различных размеров. В этом формате можно хранить любые JSON-документы и удобно категоризировать их по коллекциям.

Часть 2. Redis.

Шаг 1. Необходимо подготовить среду Jupiter для работы с СУБД Redis. Установка пакета redis и подключение Redis с аутентификацией по локальному хосту. По умолчанию Redis содержит 16 баз данных, которые имеют порядковые номера. Так как используется два файла – было произведено подключение client к БД 1 на 1000 строк и подключение r к БД 0 на 10 000 строк.

Redis

```
[7]: !pip install redis
Requirement already satisfied: redis in ./config/jupyterlab-desktop/jlab_server/lib/python3.12/site-packages (5.0.8)

[9]: import redis
import csv
import json

•[17]: # 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.

[18]: # Подключение ко второй базе данных
client = redis.Redis(host='localhost', port=6379, db=1)
```

Рис. 27. Подключение к СУБД redis и ее базам данных.

Шаг 2. Создать запись в базе данных Redis в формате title, author, year, library через Python с помощью хешей Redis. Хеши позволяют хранить связанные поля и значения под одним ключом. Так, с помощью сохранения и извлечения данных можно убедиться в корректности подключения к Redis.

```
[22]: # Данные о книге
book_id = "book:1" # Уникальный идентификатор книги
book_data = {
    "title": "1984",
    "author": "George Orwell",
    "year": "1949",
    "library": "Central Library"
}

# Сохранение данных в хеш Redis
client.hset(book_id, mapping=book_data)

print(f"Запись о книге '{book_data['title']}' успешно добавлена в базу данных 1.")

Запись о книге '1984' успешно добавлена в базу данных 1.

[23]: # Извлечение данных о книге
retrieved_book = client.hgetall(book_id)

# Декодирование байтовых строк в обычные строки (если необходимо)
decoded_book = {key.decode(): value.decode() for key, value in retrieved_book.items()}

print("Данные о книге:", decoded_book)

Данные о книге: {'title': '1984', 'author': 'George Orwell', 'year': '1949', 'library': 'Central Library'}
```

Рис. 28. Добавление тестовой записи в БД и ее вывод

Шаг 3. Загрузка данных в Redis.

После того, как файл будет находиться в среде Jupyter, для каждой строки в файле необходимо использовать уникальный ключ для хранения данных.

Загрузку данных из файла JSON необходимо производить построчно с помощью атрибута `line`.

```
[24]: %%time

# Открытие файла и загрузка данных в Redis
with open('books1000.json', 'r') as file:
    for index, line in enumerate(file):
        # Преобразование строки JSON в словарь Python
        data = json.loads(line)

        # Генерация уникального ключа для каждой записи
        key = f"book:{index + 1}"

        # Сохранение данных в Redis как строку JSON
        client.set(key, json.dumps(data))
```

CPU times: user 553 ms, sys: 298 ms, total: 851 ms
Wall time: 2.21 s

```
[25]: %%time

# Открытие файла и загрузка данных в Redis
with open('books10k.json', 'r') as file:
    for index, line in enumerate(file):
        # Преобразование строки JSON в словарь Python
        data = json.loads(line)

        # Генерация уникального ключа для каждой записи
        key = f"book:{index + 1}"

        # Сохранение данных в Redis как строку JSON
        r.set(key, json.dumps(data))
```

CPU times: user 6.2 s, sys: 3.1 s, total: 9.3 s
Wall time: 22.8 s

Рис. 29. Загрузка файлов JSON на 1000 и 10 000 строк в Redis

Проверку загрузки необходимо осуществить с помощью веб-интерфейса. Так, переключаясь между БД 0 и БД 1, можно увидеть их корректное наполнение сгенерированными данными.

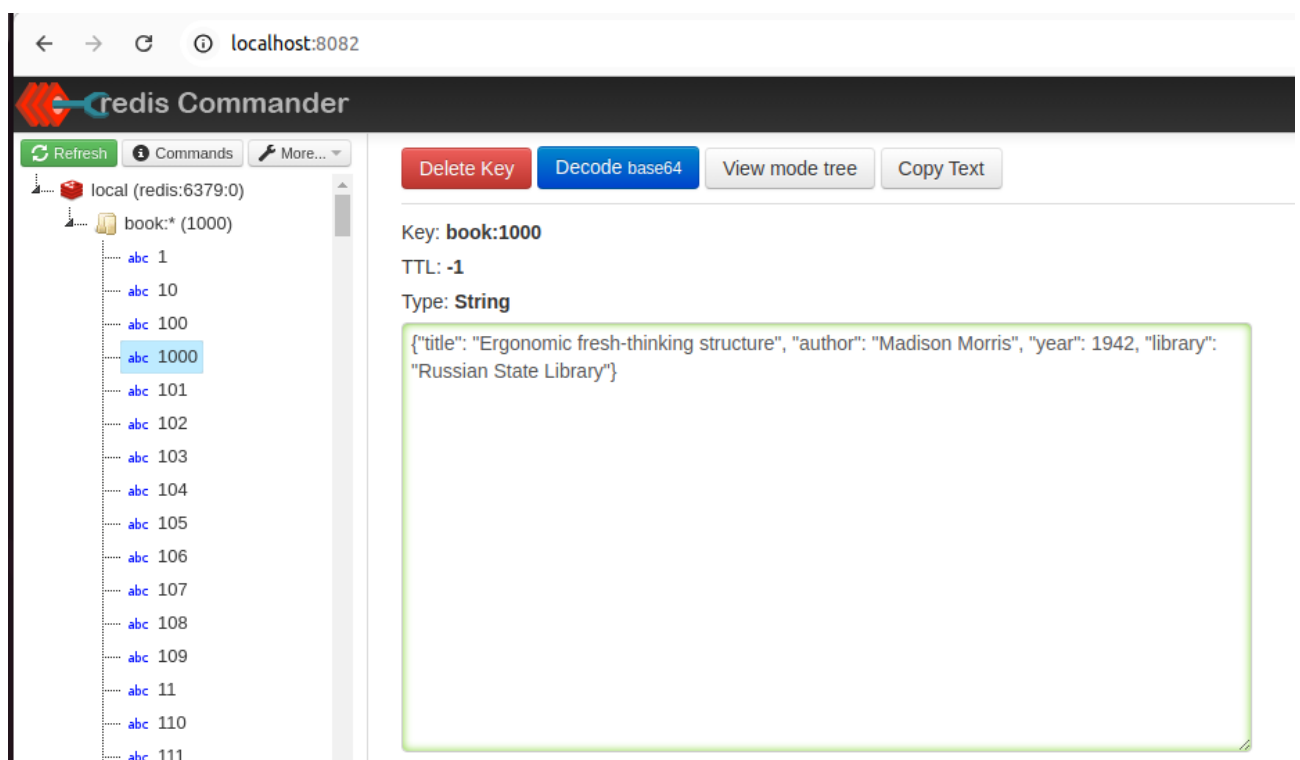


Рис. 30. Просмотр БД1, которая содержит 1000 строк

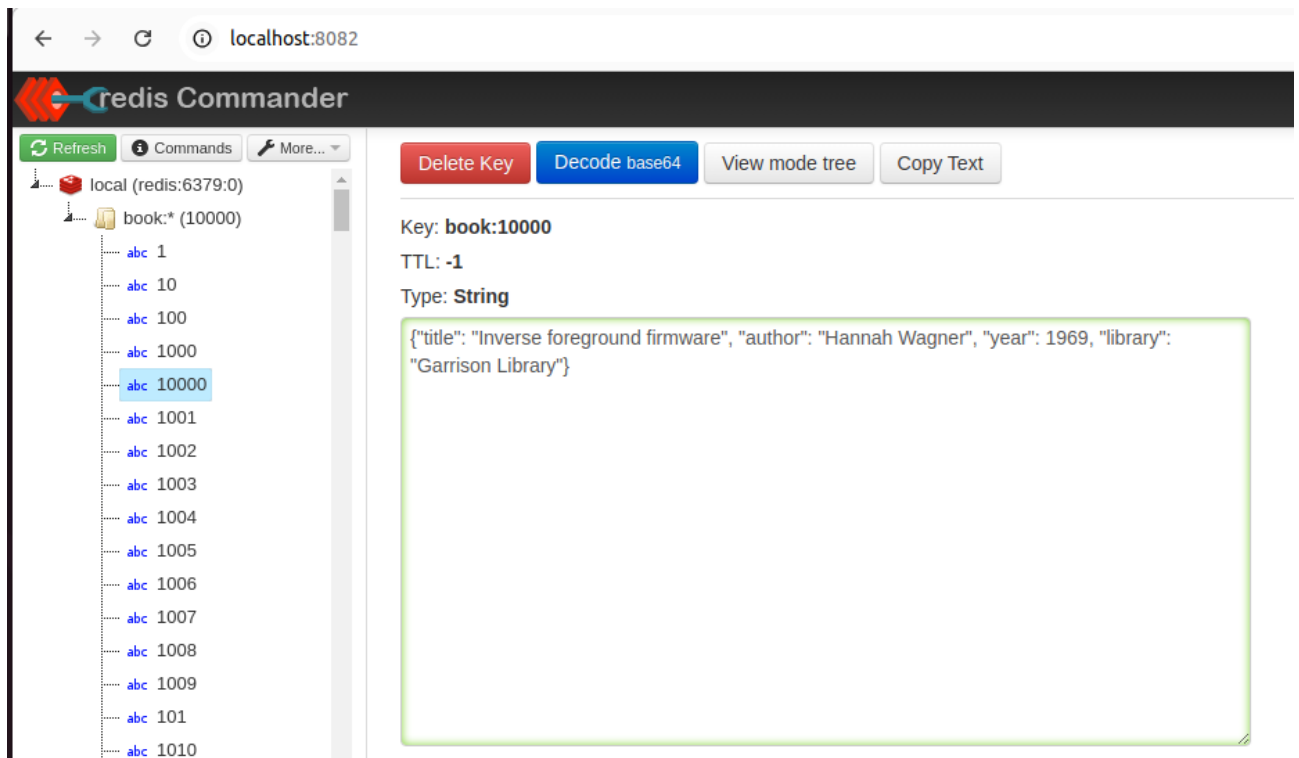


Рис. 31. Просмотр БД, которая содержит 10000 строк.

Шаг 4. Выборка данных. Так как данные в БД хранятся в виде строк, необходимо использовать метод `get()` для получения всех значений по ключу `book`.

```
[33]: %%time

# Получение всех ключей книг
book_keys = client.keys('book:*') # Получаем все ключи, начинающиеся с 'book:'

for key in book_keys:
    stored_data = client.get(key)
    if stored_data:
        decoded_data = stored_data.decode('utf-8')
        print(f"{key.decode('utf-8')}: {decoded_data}")

book:706: {"title": "Face-to-face impactful interface", "author": "John Jackson", "year": 1950, "library": "Library of the Academy of Sciences"}
book:281: {"title": "Decentralized executive functionalities", "author": "Jill Werner", "year": 1964, "library": "Russian State Library"}
book:932: {"title": "Secured 6thgeneration pricing structure", "author": "Joel James", "year": 2017, "library": "Library of Alexandria"}
book:900: {"title": "Networked attitude-oriented task-force", "author": "Krista Williams", "year": 1940, "library": "Russian State Library"}
book:700: {"title": "Optional modular functionalities", "author": "David Davis", "year": 2015, "library": "Library of the Academy of Sciences"}
book:314: {"title": "Quality-focused motivating monitoring", "author": "Mary Williams", "year": 1942, "library": "Garrison Library"}
CPU times: user 518 ms, sys: 289 ms, total: 808 ms
Wall time: 2.71 s
```

Рис. 32. Получение всех строк, которые хранятся в БД1

```
[34]: %%time

# Получение всех ключей книг
book_keys = r.keys('book:*') # Получаем все ключи, начинающиеся с 'book:'

for key in book_keys:
    stored_data = r.get(key)
    if stored_data:
        decoded_data = stored_data.decode('utf-8')
        print(f"{key.decode('utf-8')}: {decoded_data}")

book:4572: {"title": "Devolved intangible moderator", "author": "Melissa Andersen", "year": 1996, "library": "Russian State Library"}
book:6654: {"title": "Versatile system-worthy interface", "author": "Joshua Friedman", "year": 1992, "library": "National Library of Great Britain"}
book:9798: {"title": "Centralized logistical access", "author": "Lisa McCoy", "year": 1934, "library": "Library of Alexandria"}
book:9060: {"title": "Optional multi-tasking service-desk", "author": "Anthony Clark", "year": 2005, "library": "Garrison Library"}
book:2863: {"title": "Expanded heuristic artificial intelligence", "author": "Austin Lindsey", "year": 1948, "library": "Library of the Academy of Sciences"}
CPU times: user 4.78 s, sys: 2.06 s, total: 6.84 s
Wall time: 17.8 s
```

Рис. 33. Получение всех строк, которые хранятся в БД0

Так как каждая книга была сохранена с уникальным ключом в формате `book:<id>`, то, например, для первой книги ключ будет `book:1`. Чтобы извлечь данные о книге, необходимо использовать метод `get()`, подставив нужный порядковый номер.

```
[31]: %%time
# Извлечение данных о книге по ключу
book_id = 'book:666'
stored_data = client.get(book_id)

if stored_data:
    # Декодирование байтовой строки в обычную строку
    decoded_data = stored_data.decode('utf-8')
    print("Данные о книге:", decoded_data)
else:
    print(f"Ключ '{book_id}' не найден в Redis.")
```

Данные о книге: {"title": "Configurable fresh-thinking throughput", "author": "Jacob Sanchez", "year": 1952, "library": "Garrison Library"}
CPU times: user 0 ns, sys: 3.01 ms, total: 3.01 ms
Wall time: 11.6 ms

Рис. 34. Извлечение книги под номером 666 из 1000 значений

```
[32]: %%time
# Извлечение данных о книге по ключу
book_id = 'book:6666'
stored_data = r.get(book_id)

if stored_data:
    # Декодирование байтовой строки в обычную строку
    decoded_data = stored_data.decode('utf-8')
    print("Данные о книге:", decoded_data)
else:
    print(f"Ключ '{book_id}' не найден в Redis.")
```

Данные о книге: {"title": "Implemented foreground Graphical User Interface", "author": "Kenneth Huerta", "year": 1954, "library": "Library of Alexandria"}
CPU times: user 1.89 ms, sys: 0 ns, total: 1.89 ms
Wall time: 4.79 ms

Рис. 35. Извлечение книги под номером 6666 из 10 000 значений.

Получить первые 5 строк можно также с использованием метода `get()`, используя простой цикл.

```
[43]: %%time
# Получение первых 5 записей
books = []
for i in range(0, 5):
    key = f'book:{i}'
    stored_data = client.get(key)
    if stored_data:
        books.append(stored_data.decode('utf-8')) # Декодируем байтовую строку в обычную строку

print("Первые 5 книг:", books)
```

Первые 5 книг: [{'title': 'Right-sized real-time pricing structure', 'author': 'Mary Elliott DDS', 'year': 1910, 'library': 'Library of the Academy of Sciences'}, {'title': 'Networked bandwidth-monitored hub', 'author': 'Jeremy Thornton', 'year': 2020, 'library': 'National Library of Great Britain'}, {'title': 'Exclusive systematic superstructure', 'author': 'Angela Gallagher', 'year': 2003, 'library': 'National Library of Great Britain'}, {'title': 'Stand-alone heuristic Internet solution', 'author': 'Lisa Wright', 'year': 1959, 'library': 'National Library of Great Britain'}]
CPU times: user 3.88 ms, sys: 5.34 ms, total: 9.22 ms
Wall time: 18.7 ms

Рис. 36. Получение первых 5 записей из 1000 значений

```
[44]: %%time
# Получение первых 5 записей
books = []
for i in range(0, 5):
    key = f'book:{i}'
    stored_data = r.get(key)
    if stored_data:
        books.append(stored_data.decode('utf-8')) # Декодируем байтовую строку в обычную строку

print("Первые 5 книг:", books)
```

Первые 5 книг: [{'title': 'Reactive background Graphical User Interface', 'author': 'Lauren Stevens', 'year': 1946, 'library': 'Library of Alexandria'}, {'title': 'Focused explicit model', 'author': 'Catherine Dixon', 'year': 1955, 'library': 'National Library of Great Britain'}, {'title': 'Monitored full-range forecast', 'author': 'Christopher Mayer', 'year': 1945, 'library': 'Russian State Library'}, {'title': 'Distributed 24hour project', 'author': 'Kimberly Thomas', 'year': 1948, 'library': 'National Library of Great Britain'}]
CPU times: user 9.1 ms, sys: 2.93 ms, total: 12 ms
Wall time: 24.7 ms

Рис. 37. Получение первых 5 записей из 10 000 значений

Шаг 4. Обновление данных.

Так как данные хранятся как строки, можно использовать команду SET для обновления значения по ключу. Для этого необходимо указать номер книги и обновленное значение в необходимых полях.

```
[46]: %%time
# Обновление строки
client.set('book:10', '{"title":"Updated Book","author":"Mary Jane","year":2024,"library":"National Library"}')

# Проверка обновления
updated_value = client.get('book:10')
print(updated_value.decode('utf-8'))

{"title":"Updated Book","author":"Mary Jane","year":2024,"library":"National Library"}
CPU times: user 4.87 ms, sys: 572 µs, total: 5.44 ms
Wall time: 13.6 ms

[47]: %%time
# Обновление строки
r.set('book:1000', '{"title":"Updated Book","author":"Brian Mars","year":2000,"library":"American Library"}')

# Проверка обновления
updated_value = r.get('book:1000')
print(updated_value.decode('utf-8'))

{"title":"Updated Book","author":"Brian Mars","year":2000,"library":"American Library"}
CPU times: user 1.41 ms, sys: 2.27 ms, total: 3.68 ms
Wall time: 6.37 ms
```

Рис. 38. Обновление строки по ключу

Шаг 5. Удаление и восстановление данных

Чтобы удалить определенные строки в Redis с возможностью восстановления, необходимо использовать временное хранение удаляемых данных. Это можно сделать, сохраняя удаляемые записи по другому ключу перед их удалением, например, backup.

```
[57]: %%time
def delete_with_backup(key):
    # Получаем значение по ключу
    value = client.get(key)

    if value:
        # Сохраняем удаляемое значение в резервное хранилище
        backup_key = f'backup:{key}' # Создаем ключ для резервного хранения
        client.set(backup_key, value) # Сохраняем значение

        # Удаляем оригинальный ключ
        client.delete(key)
        print(f"Ключ '{key}' удален и сохранен в '{backup_key}'.")
    else:
        print(f"Ключ '{key}' не найден.")

# Пример удаления ключа 'book:100'
delete_with_backup('book:100')
```

Ключ 'book:100' удален и сохранен в 'backup:book:100'.
CPU times: user 4.59 ms, sys: 1.03 ms, total: 5.62 ms
Wall time: 17.4 ms

Рис. 39. Удаление строки по ключу из 1000 значений

Для восстановления необходимо воспользоваться методом get(), чтобы получить резервное значение и методом set(), чтобы вернуть ему прежнее расположение.

```
[58]: %%time

def restore_from_backup(key):
    backup_key = f'backup:{key}' # Ключ резервной копии

    # Получаем значение из резервного хранилища
    backup_value = client.get(backup_key)

    if backup_value:
        # Восстанавливаем оригинальный ключ
        client.set(key, backup_value)
        print(f"Ключ '{key}' восстановлен из '{backup_key}'.")

        # Удаляем резервную копию
        client.delete(backup_key)
    else:
        print(f"Резервная копия для ключа '{key}' не найдена.")

# Пример восстановления ключа 'book:1'
restore_from_backup('book:100')

Ключ 'book:100' восстановлен из 'backup:book:100'.
CPU times: user 2.93 ms, sys: 872 µs, total: 3.8 ms
Wall time: 9.85 ms
```

Рис. 40. Восстановление удаленной строки из 1000 записей

```
[59]: %%time

def delete_with_backup(key):
    # Получаем значение по ключу
    value = r.get(key)

    if value:
        # Сохраняем удаляемое значение в резервное хранилище
        backup_key = f'backup:{key}' # Создаем ключ для резервного хранения
        r.set(backup_key, value) # Сохраняем значение

        # Удаляем оригинальный ключ
        r.delete(key)
        print(f"Ключ '{key}' удален и сохранен в '{backup_key}'.")
    else:
        print(f"Ключ '{key}' не найден.")

# Пример удаления ключа 'book:1000'
delete_with_backup('book:1000')

Ключ 'book:1000' удален и сохранен в 'backup:book:1000'.
CPU times: user 2.86 ms, sys: 5.32 ms, total: 8.18 ms
Wall time: 17.3 ms
```

Рис. 41. Удаление строки по ключу из 10 000 значений

```
[60]: %%time

def restore_from_backup(key):
    backup_key = f'backup:{key}' # Ключ резервной копии

    # Получаем значение из резервного хранилища
    backup_value = r.get(backup_key)

    if backup_value:
        # Восстанавливаем оригинальный ключ
        r.set(key, backup_value)
        print(f"Ключ '{key}' восстановлен из '{backup_key}'.")

        # Удаляем резервную копию
        r.delete(backup_key)
    else:
        print(f"Резервная копия для ключа '{key}' не найдена.")

# Пример восстановления ключа 'book:1000'
restore_from_backup('book:1000')

Ключ 'book:1000' восстановлен из 'backup:book:1000'.
CPU times: user 3.59 ms, sys: 2.23 ms, total: 5.82 ms
Wall time: 14.6 ms
```

Рис. 42. Восстановление удаленной строки из 1000 записей

Сравнение времени проводимых операций в СУБД Redis

Кол-во документов	Загрузка из файла	Получение всех строк	Фильтрация по значению	Получение топ 5 док-тов	Обновление одного док-та	Удаление и сохранение	Восстановление
1000	2.21 с	2.71 с	11.6 мс	18.7 мс	13.6 мс	17.4 мс	9.85 мс
10 000	22.8 с	17.8 с	4.79 мс	24.7 мс	6.37 мс	17.3 мс	14.6 мс

Выводы по работе с Redis.

1) Время выполнения операций в Redis остается на низком уровне, даже при увеличении объема данных. Например, время фильтрации по значению для 10 000 строк составляет всего 4.79 мс, что указывает на высокую производительность системы при выполнении запросов.

2) Время загрузки из файла значительно увеличивается с 2.21 секунд для 1000 строк до 22.8 секунд для 10,000 строк. Это может указывать на необходимость оптимизации процесса загрузки данных, особенно при больших объемах.

Рекомендации по улучшению работы с СУБД

1) Для ускорения процесса загрузки можно использовать пакетные операции, что позволит сократить количество запросов к серверу и уменьшить задержку.

2) Можно пересмотреть структуру хранения данных в Redis для уменьшения избыточности и повышения эффективности доступа к данным, например, по хешам.

Redis быстрее и удобнее стандартных баз, но не так надежен и используется в основном как вспомогательная система. В применении оказался менее гибким, чем MongoDB, например, при фильтрации и обновлении нескольких значений.

Часть 3. Работа с Neo4j

Для работы с запросами в консоли **Neo4j** необходимо заполнить базу данных «Учебные курсы» данными о курсах, студентах, которые на них обучаются и сотрудниках, которые их разрабатывают и преподают.

Формируем запрос в консоли.

```
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'}),
```

Рис. 43. Формирование запроса в консоли

После отработки запроса образуется графовая СУБД с заданными параметрами.

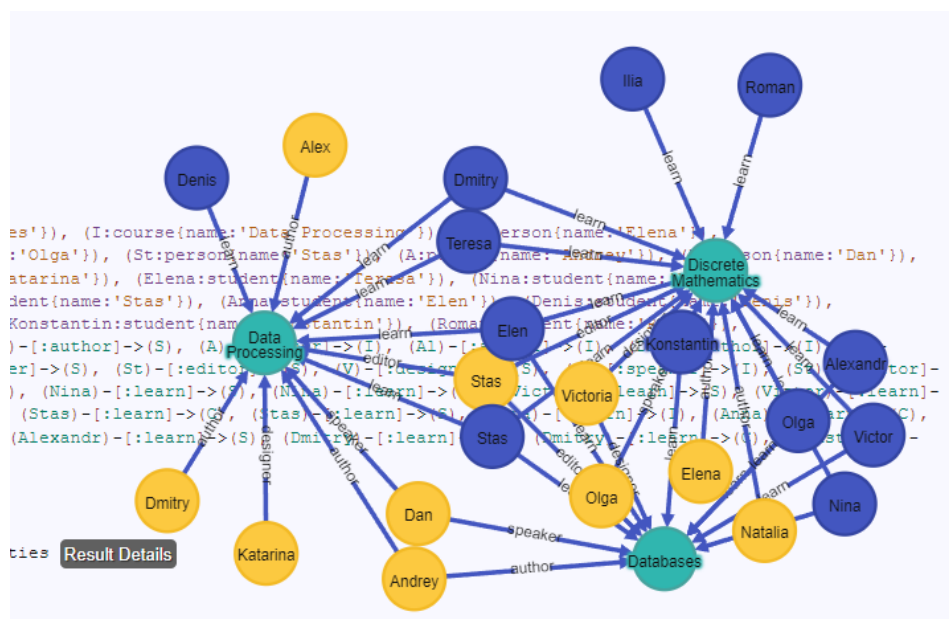


Рис. 44. Представление графовой СУБД на основе добавленных данных

Узлы базы: course - курсы, 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.

Выполнение заданий в контексте базы данных «Учебные курсы» на Neo4j

1. Напишите 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, c.name AS courseName;
```

MATCH – поиск по условию, RETURN – возврат значений. Условие в данном случае это связь learn с курсом Дискретной математики.

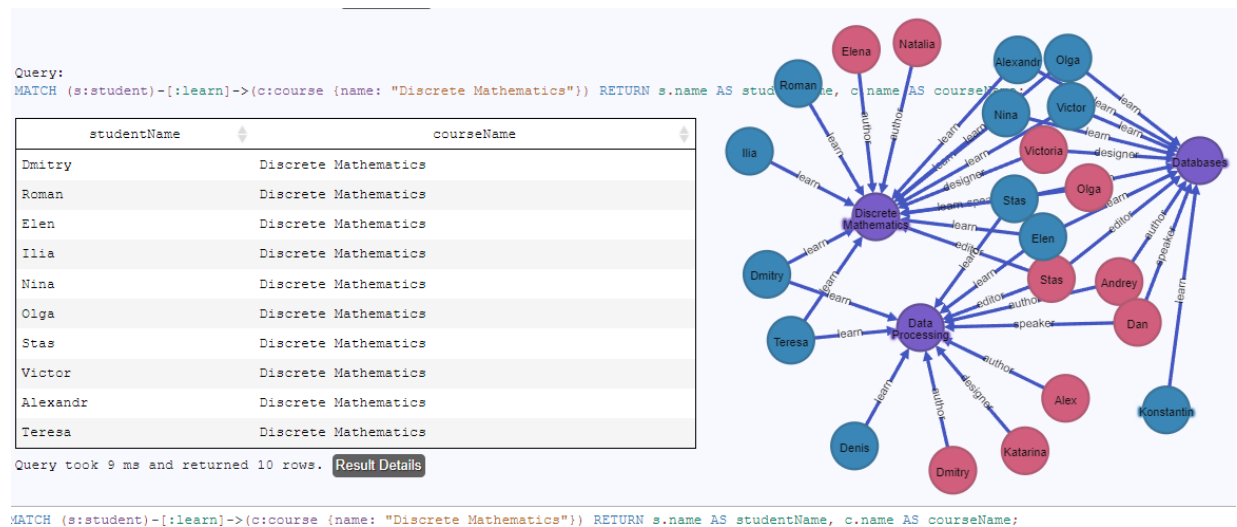


Рис. 45. Запрос 1

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

```
MATCH (s:student)-[:learn]->(c:course) WHERE s.name IN ["Nina", "Olga"] RETURN s.name AS studentName, c.name AS courseName;
```

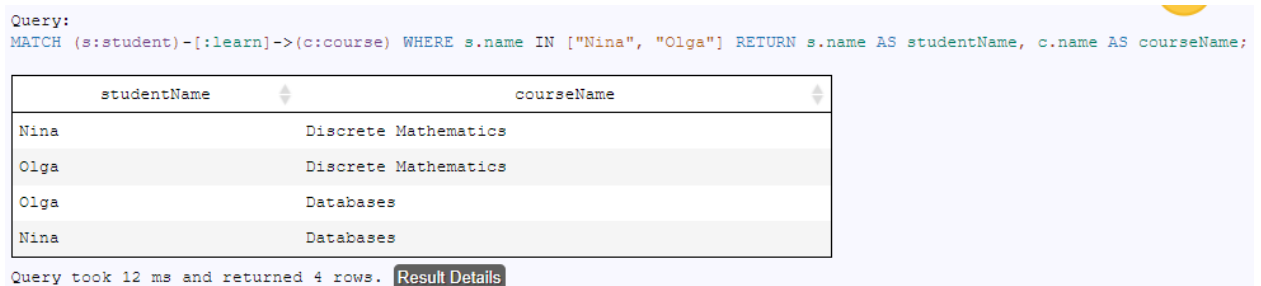


Рис. 46. Запрос 2

3. Найдите всех авторов курса "Data Processing" и верните их имена.

```
MATCH (a:person)-[:author]->(c:course {name: "Data Processing"}) RETURN a.name AS authorName;
```

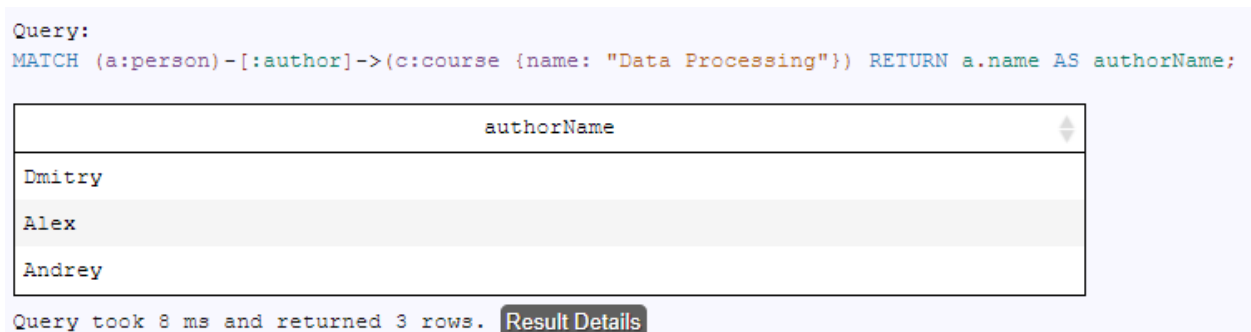


Рис. 47. Запрос 3

4. Напишите запрос, который вернет список курсов, созданных сотрудником с именем "Andrey".

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

Query:

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

courseName
Data Processing
Databases

Query took 10 ms and returned 2 rows.

[Result Details](#)

Рис. 48. Запрос 4

5. Составьте запрос для получения списка всех курсов, где "Stas" является редактором.

```
MATCH (e:person {name: "Stas"})-[:editor]->(c:course) RETURN
c.name AS courseName;
```

courseName
Discrete Mathematics
Databases
Data Processing

Query took 14 ms and returned 3 rows.

[Result Details](#)

Рис. 49. Запрос 5

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

```
MATCH (s:student)-[:learn]->(c:course {name: "Databases"})
MATCH (a:person)-[:author]->(c) MATCH (sp:person)-[:speaker]->(c)
MATCH (e:person)-[:editor]->(c) RETURN s.name AS studentName,
c.name AS courseName, COLLECT(DISTINCT a.name) AS authors,
COLLECT(DISTINCT sp.name) AS speakers, COLLECT(DISTINCT e.name) AS
editors;
```

Функция COLLECT() предназначена для агрегирования имен авторов, дикторов и редакторов в списки.

DISTINCT – отображение уникальных записей.

studentName	courseName	authors	speakers	editors
Stas	Databases	[Andrey]	[Dan]	[Stas]
Elen	Databases	[Andrey]	[Dan]	[Stas]
Konstantin	Databases	[Andrey]	[Dan]	[Stas]
Victor	Databases	[Andrey]	[Dan]	[Stas]
Olga	Databases	[Andrey]	[Dan]	[Stas]
Alexandr	Databases	[Andrey]	[Dan]	[Stas]
Nina	Databases	[Andrey]	[Dan]	[Stas]

Query took 54 ms and returned 7 rows. [Result Details](#)

Рис. 50. Запрос 6

7. Найдите всех сотрудников, которые имеют отношение к созданию курса "Discrete Mathematics", и определите их роль.

```
MATCH (p:person)-[r]->(c:course {name: "Discrete Mathematics"}) RETURN p.name AS employeeName, type(r) AS role;
```

employeeName	role
Victoria	designer
Stas	editor
Olga	speaker
Natalia	author
Elena	author

Query took 43 ms and returned 5 rows. [Result Details](#)

Рис. 51. Запрос 7

8. Напишите запрос, чтобы найти всех студентов, которые учатся на всех трех курсах D

```
MATCH (s:student)-[:learn]->(c:course) WHERE c.name IN ["Discrete Mathematics", "Databases", "Data Processing"] WITH s, COUNT(c) AS coursesCount WHERE coursesCount = 3 RETURN s.name AS studentName;
```

Используется конструкция WITH, чтобы сгруппировать студентов и посчитать количество курсов, на которых они записаны.

e
M
a
t
h
e
m

studentName
Elen
Stas

Query took 180 ms and returned 2 rows. [Result Details](#)

Рис. 52. Запрос 8

9. Составьте запрос, который вернет список студентов, обучающихся на курсе "Data Processing", и перечислите авторов этого курса.

```
MATCH (s:student)-[:learn]->(c:course {name: "Data
Processing"}) MATCH (a:person)-[:author]->(c) RETURN s.name AS
studentName, COLLECT(a.name) AS authors;
```

studentName	authors
Dmitry	[Dmitry, Alex, Andrey]
Elen	[Dmitry, Alex, Andrey]
Denis	[Dmitry, Alex, Andrey]
Stas	[Dmitry, Alex, Andrey]
Teresa	[Dmitry, Alex, Andrey]

Query took 14 ms and returned 5 rows. [Result Details](#)

Рис. 53. Запрос 9

10. Напишите 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 ORDER BY studentCount DESC;
```

Функция ORDER BY DESC сортирует результаты по количеству студентов в порядке убывания.

courseName	studentCount
Discrete Mathematics	10
Databases	7
Data Processing	5

Query took 30 ms and returned 3 rows. [Result Details](#)

Рис. 54. Запрос 10

11. Найдите всех студентов, которые учатся у "Elena", и верните список курсов, на которые они записаны.

```
MATCH (s:student)-[:learn]->(c:course) MATCH (e:person {name:
"Elena"})-[:author]->(c) RETURN s.name AS studentName, c.name AS
courses;
```

studentName	courses
Dmitry	Discrete Mathematics
Roman	Discrete Mathematics
Elen	Discrete Mathematics
Ilia	Discrete Mathematics
Nina	Discrete Mathematics
Olga	Discrete Mathematics
Stas	Discrete Mathematics
Victor	Discrete Mathematics
Alexandr	Discrete Mathematics
Teresa	Discrete Mathematics

Query took 12 ms and returned 10 rows. [Result Details](#)

Рис. 55. Запрос 11

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

```
MATCH (p:person) WHERE (p)-[:author]->(:course) OR (p)-
[:speaker]->(:course) OR (p)-[:editor]->(:course) RETURN DISTINCT
p.name AS employeeName;
```

Query:
MATCH (p:person) WHERE (p)-[:author]->(:course) OR (p)-[:speaker]->(:course) OR

employeeName
Elena
Natalia
Olga
Stas
Andrey
Dan
Alex
Dmitry

Query took 38 ms and returned 8 rows. [Result Details](#)

Рис. 56. Запрос 12

13. Составьте запрос, который вернет список студентов, записанных на курсы, где

```
MATCH (s:student)-[:learn]->(c:course)<-[:designer]-(d:person {name: "Katarina"}) RETURN s.name AS studentName, c.name AS courseName;
```

Query:

```
MATCH (s:student)-[:learn]->(c:course)<-[:designer]-(d:person {name: "Katarina"}) RETURN
```

studentName	courseName
Dmitry	Data Processing
Elen	Data Processing
Denis	Data Processing
Stas	Data Processing
Teresa	Data Processing

Query took 15 ms and returned 5 rows. [Result Details](#)

Рис. 57. Запрос 13

14. Напишите запрос, чтобы найти все курсы, которые созданы командой сотрудников (автор, диктор, редактор), включающей "Dmitry".

```
MATCH (c:course)<-[:author|:speaker|:editor]-(p:person {name: "Dmitry"}) RETURN c.name AS courseName;
```

Query:

```
MATCH (c:course)<-[:author|:speaker|:editor]-(p:person {name: "Dmitry"}) RETURN c.name AS courseName;
```

courseName
Data Processing

Query took 17 ms and returned 1 rows. [Result Details](#)

Рис. 58. Запрос 14

15. Найдите 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 9 ms and returned no rows. [Result Details](#)

Рис. 59. Запрос 15

Таких значений не оказалось.

16. Напишите 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 ORDER BY courseCount DESC;
```

Show entries Search:

studentName	courseCount
Elen	3
Stas	3
Dmitry	2
Nina	2
Olga	2
Victor	2
Alexandr	2
Teresa	2
Roman	1
Ilia	1
Konstantin	1
Denis	1

Showing 1 to 12 of 12 entries
 Query took 2 ms and returned 12 rows. [Result Details](#)

Рис. 60. Запрос 16

17. Составьте запрос, чтобы получить список всех курсов, где один и тот же человек выполняет несколько ролей (например, автор и редактор).

```
MATCH (p:person)-[r:author|:speaker|:editor]->(c:course)
WITH p, c, COLLECT(DISTINCT type(r)) AS roles WHERE SIZE(roles) >
1 RETURN DISTINCT c.name AS courseName, p.name AS personName,
roles;
```

Query:
 MATCH (p:person)-[r:author|:speaker|:editor]->(c:course) WITH p, c,
 p.name AS personName, roles;
 Query took 19 ms and returned no rows. [Result Details](#)

Рис. 61. Запрос 17

Таких курсов не оказалось.

18. Напишите запрос, который вернет список всех студентов, записанных на курс

```
MATCH (s:student)-[:learn]->(c:course {name: "Discrete Mathematics"}) MATCH (d:person)-[:speaker]->(c) RETURN s.name AS studentName, d.name AS speakers;
```

studentName	speakers
Dmitry	Olga
Roman	Olga
Elen	Olga
Ilia	Olga
Nina	Olga
Olga	Olga
Stas	Olga
Victor	Olga
Alexandr	Olga
Teresa	Olga

Query took 14 ms and returned 10 rows. [Result Details](#)

Рис. 62. Запрос 18

19. Найдите всех сотрудников, которые являются одновременно авторами и дикторами какого-либо курса.

```
MATCH (p:person) WHERE (p)-[:author]->(:course) AND (p)-[:speaker]->(:course) RETURN DISTINCT p.name AS employeeName;
```

Query:

```
MATCH (p:person) WHERE (p)-[:author]->(:course) AND (p)-[:speaker]->(:course)
```

Query took 15 ms and returned no rows. [Result Details](#)

Рис. 63. Запрос 19

Таких сотрудников не оказалось.

20. Напишите запрос, чтобы найти все курсы, на которые записаны студенты "Elen" и "Stas"

```
MATCH (s:student)-[:learn]->(c:course) WHERE s.name IN ["Elen", "Stas"] RETURN c.name AS courseName, s.name AS studentName
```


Query:

```
MATCH (r:student {name: "Roman"})-[:learn]->(c:course) MATCH (o:person {name: "Olga"})-
```

courseName
Discrete Mathematics

Query took 17 ms and returned 1 rows. [Result Details](#)

Рис. 66. Запрос 22

23. Найдите всех сотрудников, которые принимали участие в создании курсов, на которых учится студент "Konstantin".

```
MATCH (s:student {name: "Konstantin"})-[:learn]->(c:course)
MATCH (e:person)-[:author|:speaker|:editor]->(c) RETURN DISTINCT
e.name AS employeeName;
```

employeeName
Stas
Dan
Andrey

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

Рис. 67. Запрос 23.

24. Напишите запрос, чтобы найти студентов, которые записаны на курсы, на которых "Victoria" является дизайнером.

```
MATCH (v:person {name: "Victoria"})-[:designer]->(c:course)
MATCH (s:student)-[:learn]->(c) RETURN DISTINCT s.name AS
studentName;
```

Show entries Search:

studentName
Dmitry
Roman
Elen
Ilia
Nina
Olga
Stas
Victor
Alexandr
Teresa
Konstantin

Showing 1 to 11 of 11 entries

Рис. 68. Запрос 24

25. Составьте запрос, чтобы получить список студентов, которые учатся на более чем двух курсах, и перечислите эти курсы.

```
MATCH (s:student)-[:learn]->(c:course) WITH s,
COLLECT(c.name) AS courses, COUNT(c) AS courseCount WHERE
courseCount > 2 RETURN s.name AS studentName, courses;
```

Query:
MATCH (s:student)-[:learn]->(c:course) WITH s, COLLECT(c.name) AS courses, COUNT(c)

studentName	courses
Elen	[Discrete Mathematics, Databases, Data Processing]
Stas	[Discrete Mathematics, Databases, Data Processing]

Query took 14 ms and returned 2 rows. [Result Details](#)

Рис. 69. Запрос 25

Выводы по работе в СУБД Neo4j.

1) Производительность графовых баз данных, таких как базы данных Neo4j, остается высокой даже при значительном увеличении объема данных. В данном случае данных было немного, запросы занимали в среднем 10-15 мс.

2) База данных Neo4j позволяет исследовать различные пути и связи между данными и максимально эффективно запрашивать их. Кроме того, можно легко извлекать сложные данные из базы данных, даже если они сильно связаны. В данном случае данные были весьма запутаны, так как со стороны сотрудников могли производиться 4 действия.

Такое представление позволило быстрее ориентироваться на связи между данными и подтверждать корректность выполнения запросов.

3) Легкость в освоении языка запросов для Neo4j (Cypher), его принцип схож с основами SQL, поэтому получилось быстро освоить и писать различные запросы.