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 кеширования и временного хранения данных.

3. Neo4j: графовая база данных, которая позволяет хранить данные в виде вершин и рёбер графа, что удобно для моделирования сложных взаимосвязей.

**Ход работы**

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

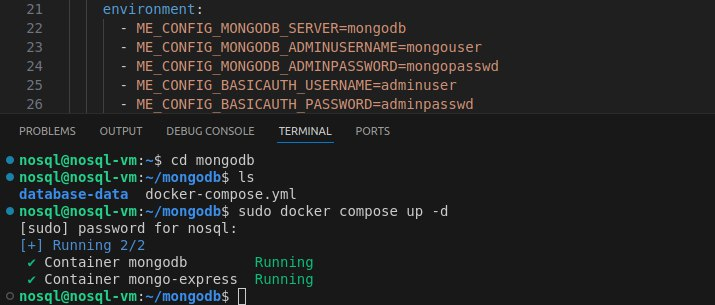


Рис. 1. Запуск MongoDB

**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.



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.

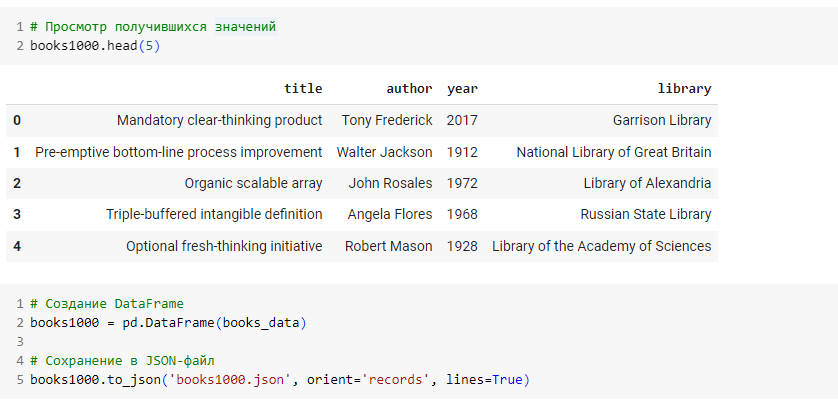


Рис. 3. Проверка сгенерированных данных и их сохранение в JSON

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

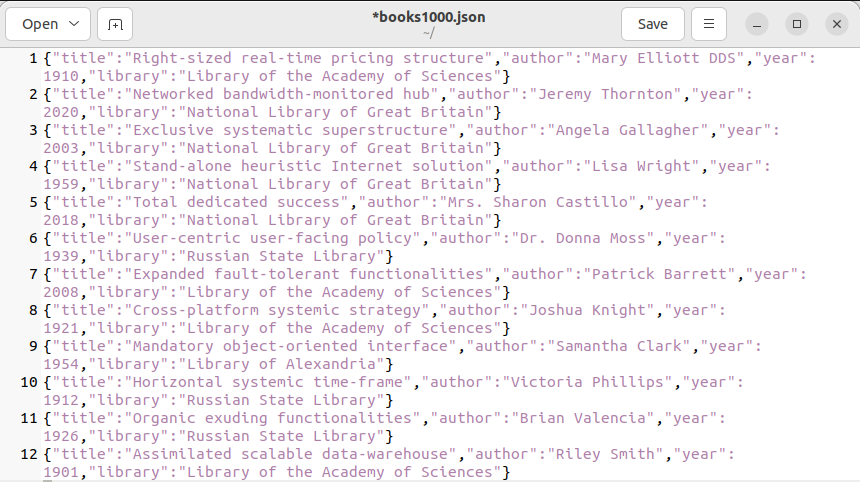


Рис. 4. Фрагмент JSON, содержащий сгенерированные данные

**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.

**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.

Рис. 5. Проверка подключения к MongoDB

****

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

Далее в среду 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.

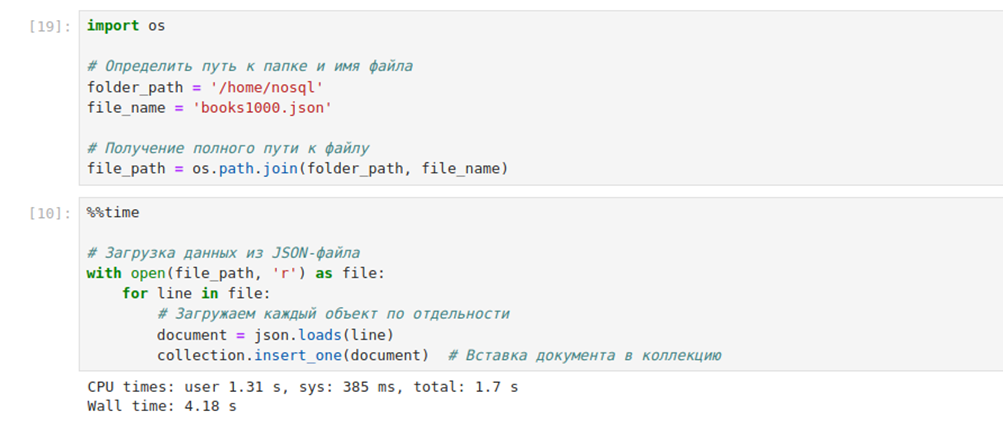


Рис. 7. Загрузка данных в коллекцию из файла JSON на 1000 строк.

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



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

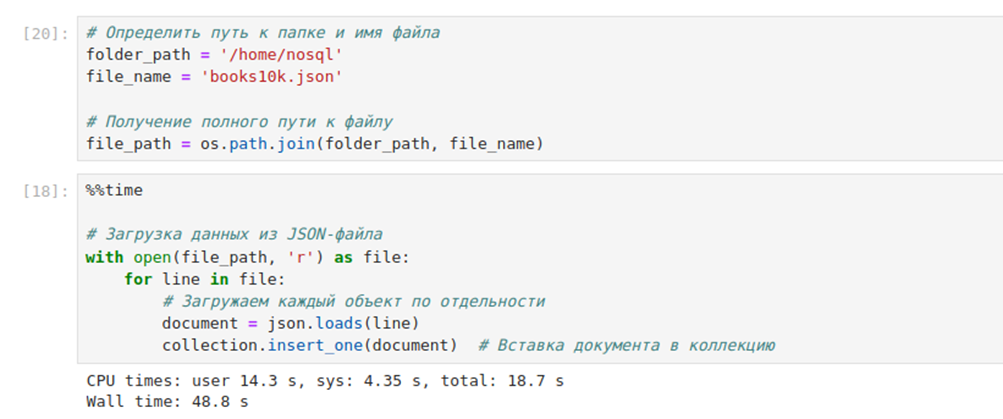


Рис. 9. Загрузка данных в коллекцию из файла JSON на 10 000 строк.

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.

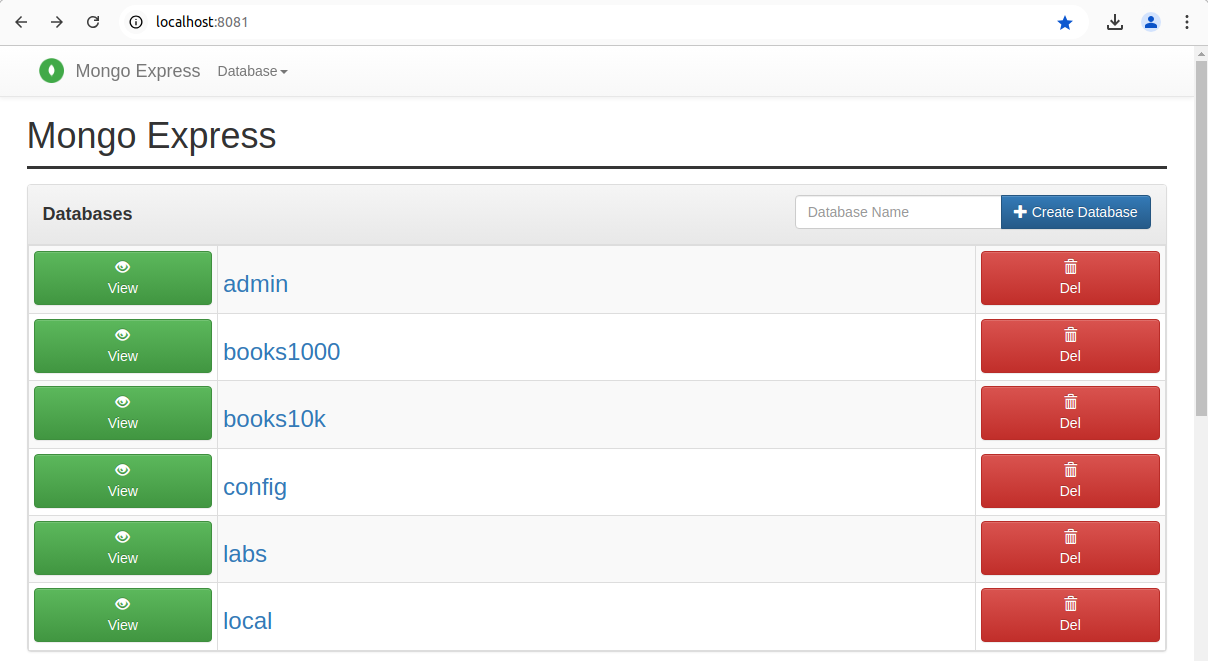


Рис. 10. Перечень баз данных в Mongo

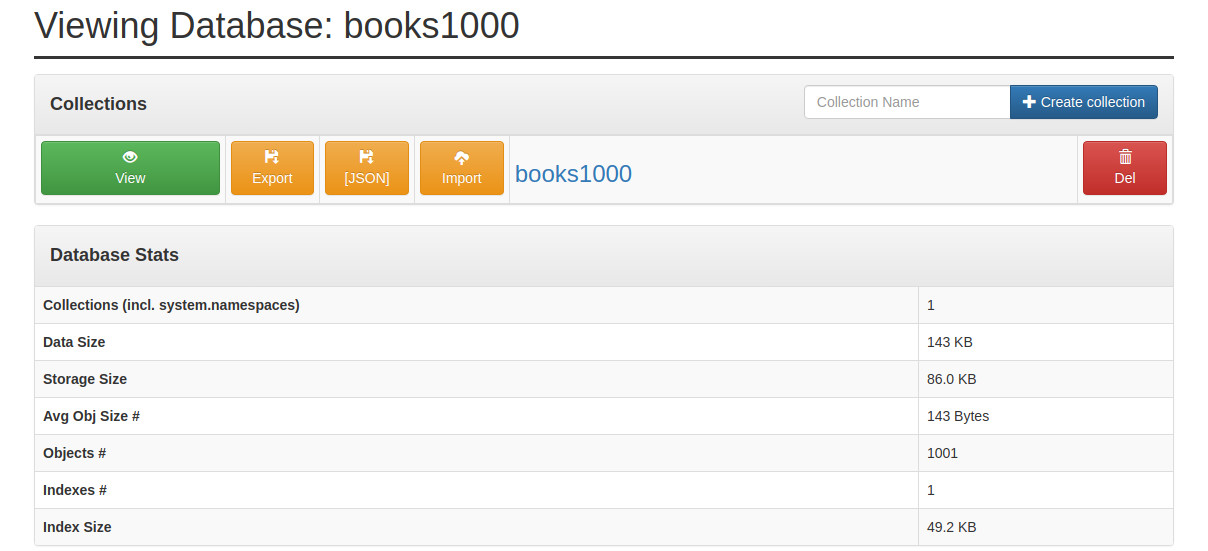


Рис. 11. Корректное количество объектов в БД для 1000 значений

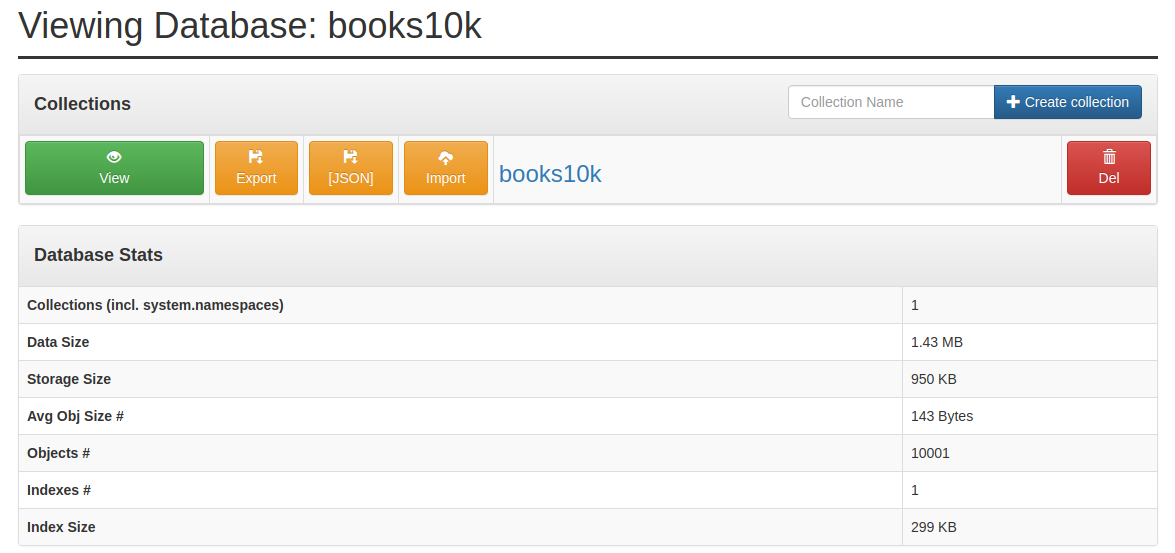
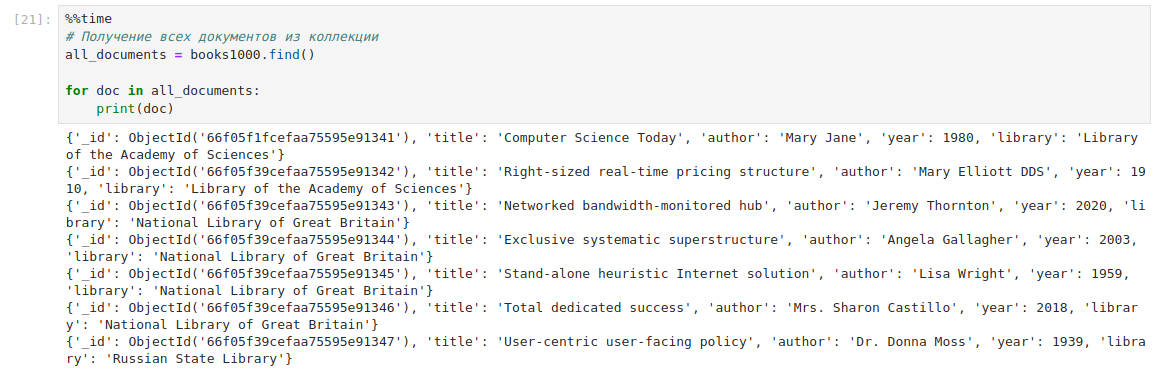


Рис. 12. Корректное количество объектов в БД для 10 000 значений.

**Шаг 5. Выборка загруженных данных.**

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



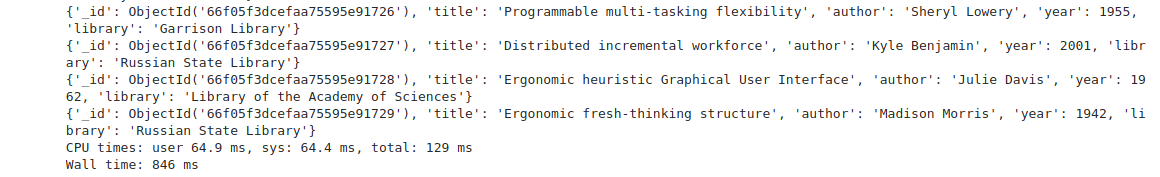
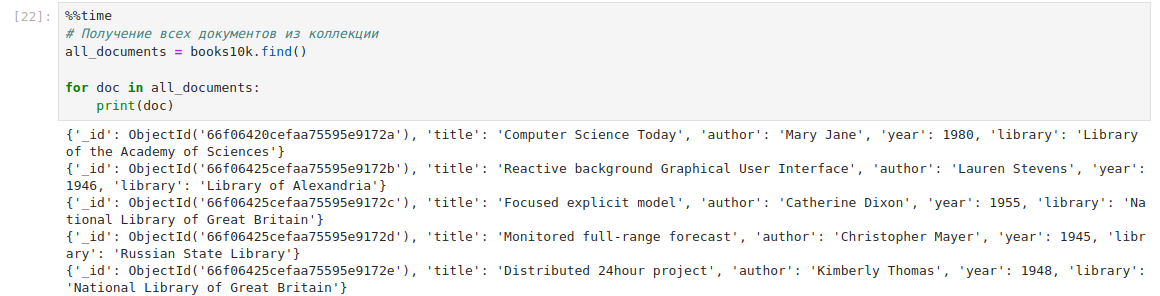


Рис. 13. Получение всех документов из коллекции на 1000 строк

****

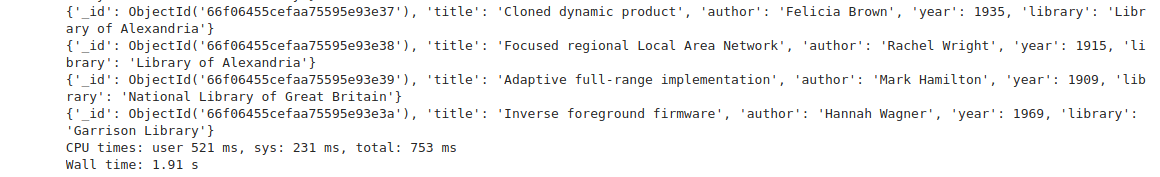
****

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

1. 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.

****

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

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

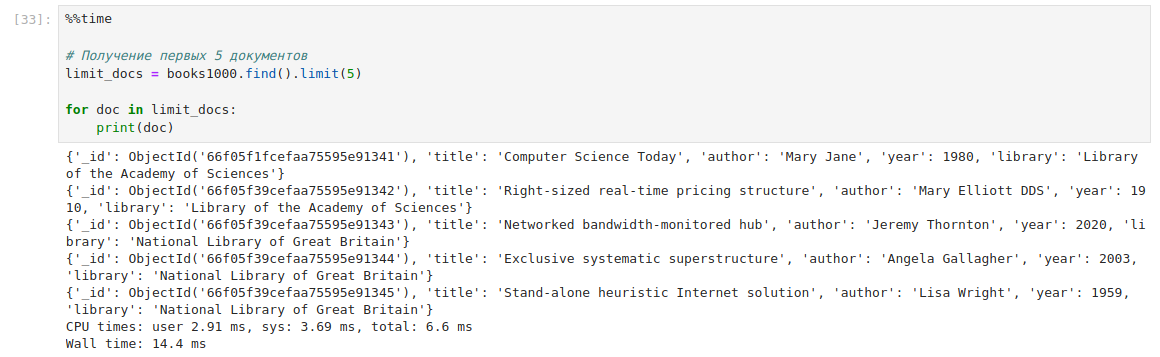
****

Рис. 16. Получение первых 5 документов из 1000

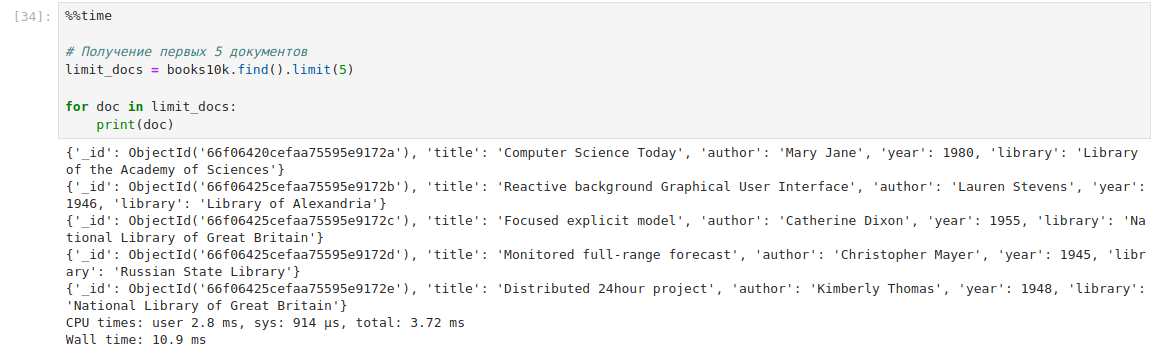
****

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.



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

1. Обновление нескольких документов. Если нужно обновить несколько документов, используется метод update\_many(). Например, чтобы изменить год для всех книг определенного автора.

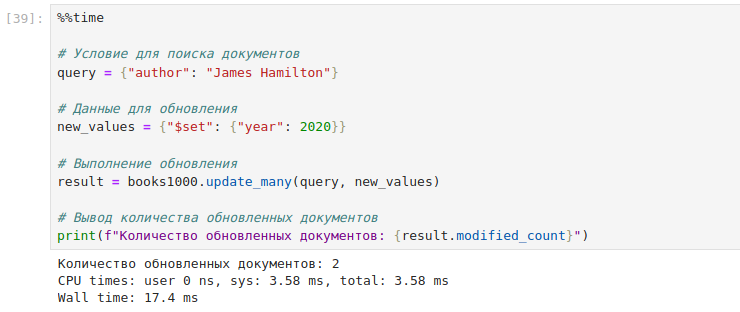


Рис. 19. Обновление данных для всех годов книги одного автора из 1000 документов

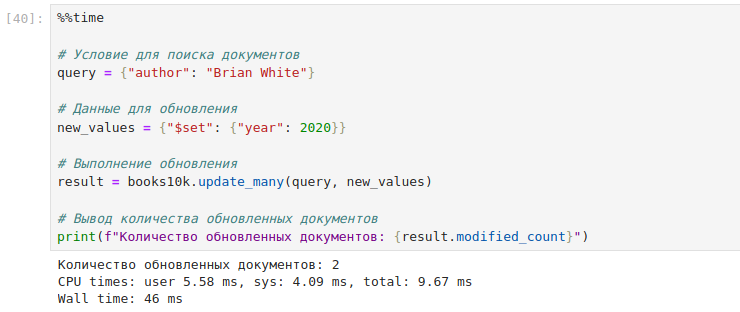


Рис. 20. Обновление данных для всех годов книги одного автора из 10 000 документов

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

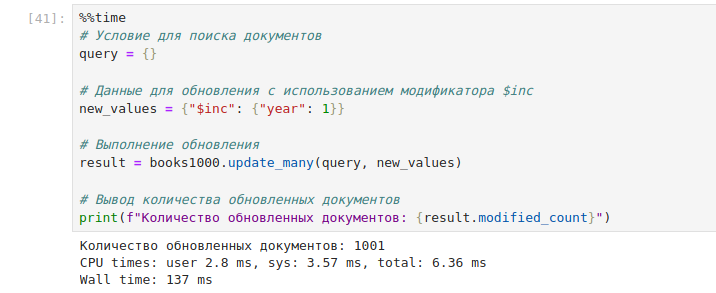


Рис. 21. Увеличение всех записей года на 1 для 1000 значений

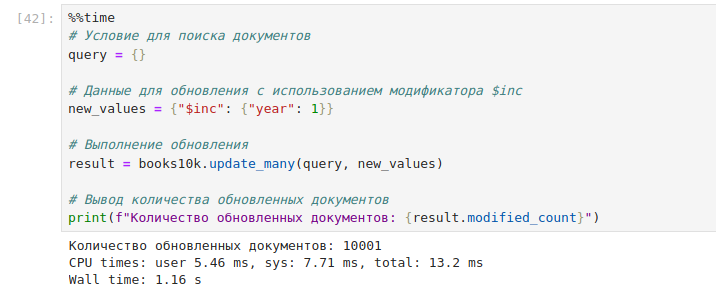


Рис. 22. Увеличение всех записей года на 1 для 10 000 значений

**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.

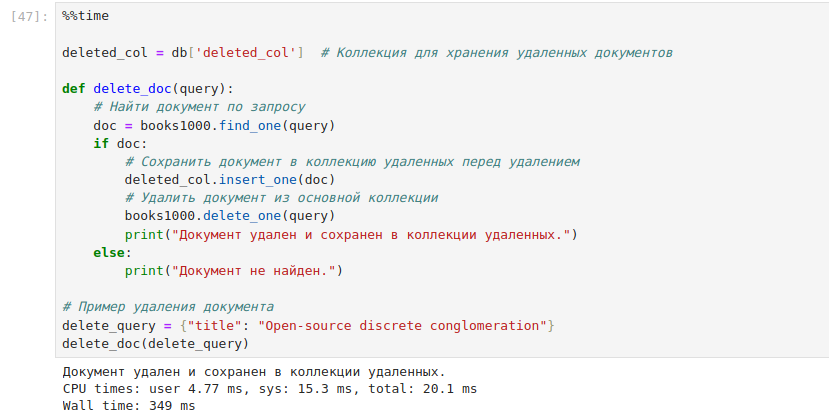


Рис. 23. Удаление документа из основной коллекции на 1000 значений

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

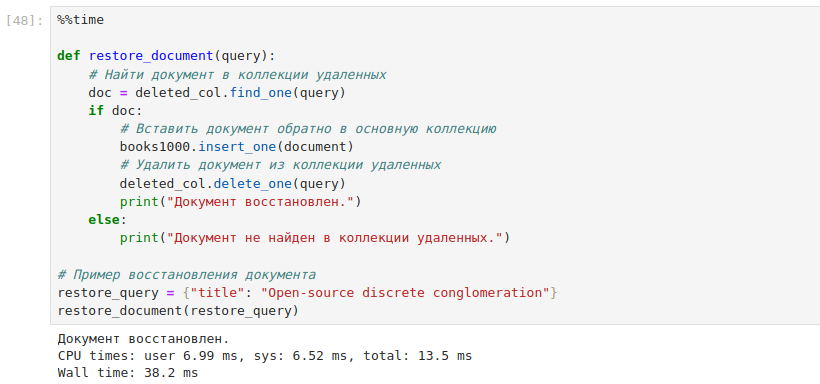


Рис. 24. Восстановление удаленного документа в коллекцию 1000 документов

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

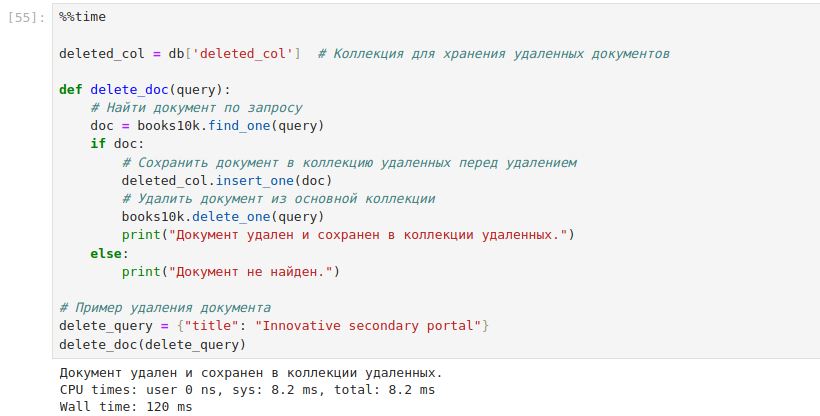


Рис. 25. Удаление документа из основной коллекции на 10 000 значений

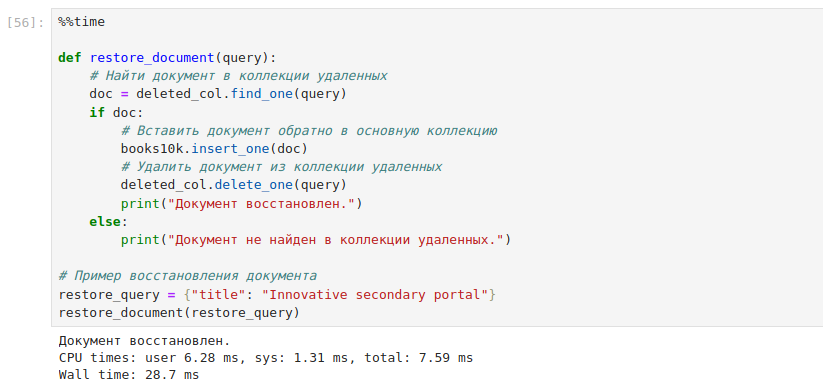


Рис. 26. Восстановление удаленного документа в коллекцию на 10 000 документов

**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.



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.



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.

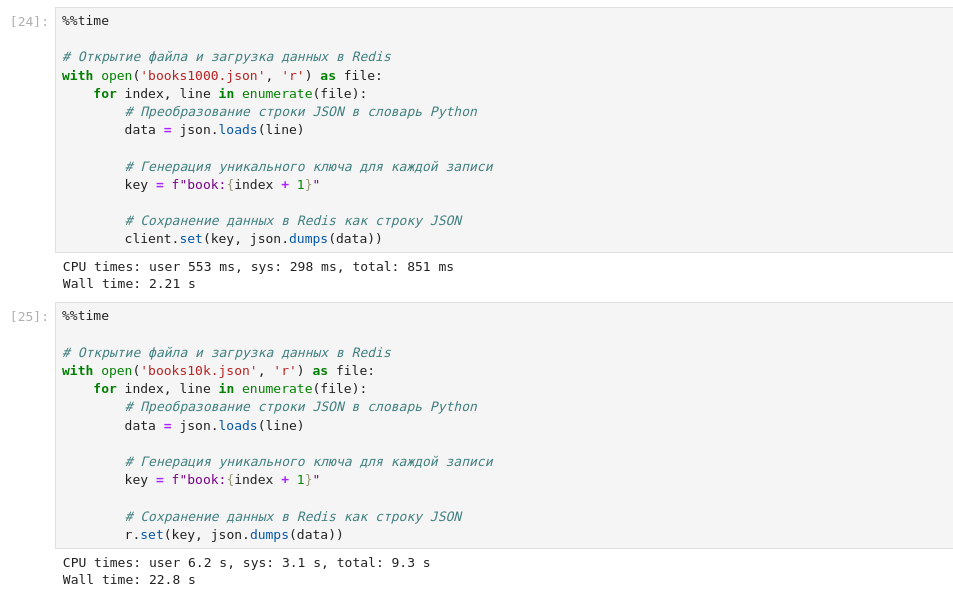


Рис. 29. Загрузка файлов JSON на 1000 и 10 000 строк в Redis

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

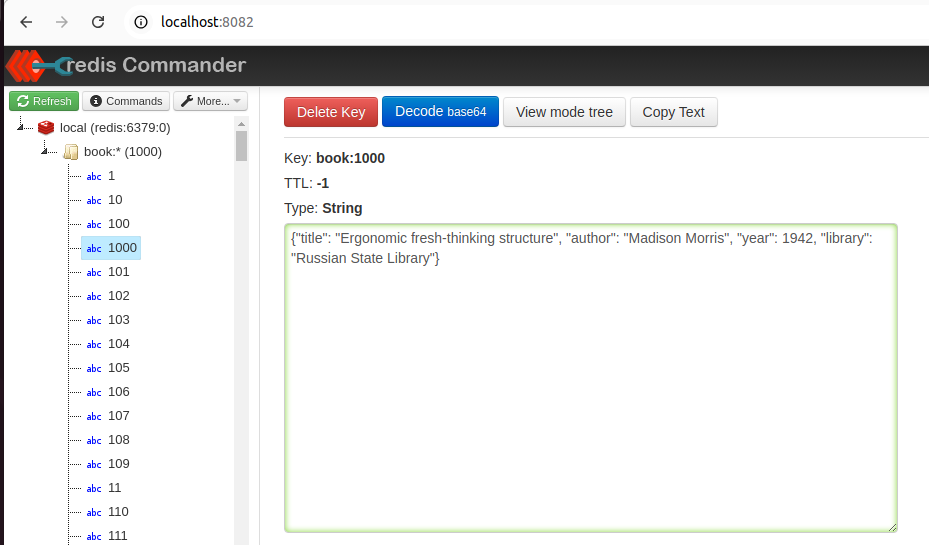


Рис. 30. Просмотр БД1, которая содержит 1000 строк

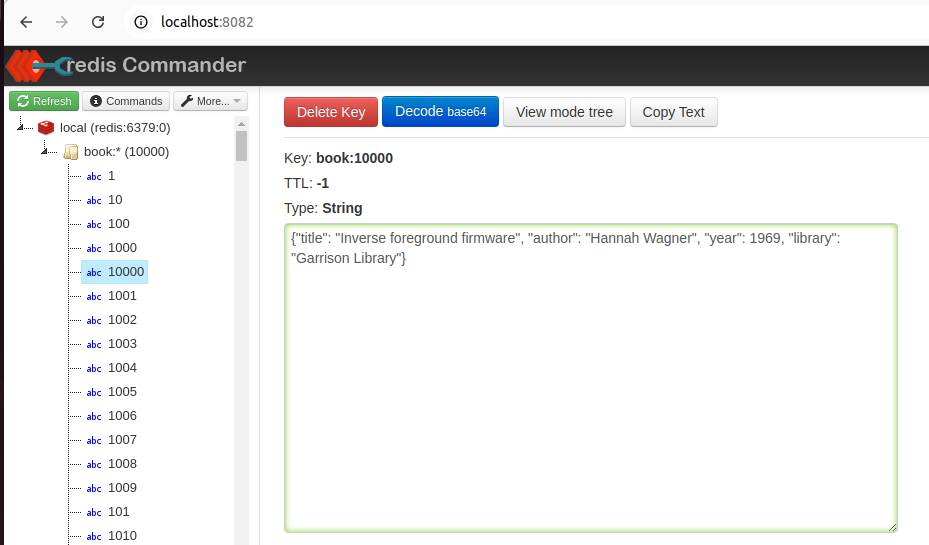


Рис. 31. Просмотр БД0, которая содержит 10000 строк.

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

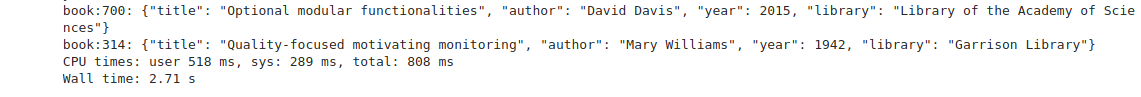
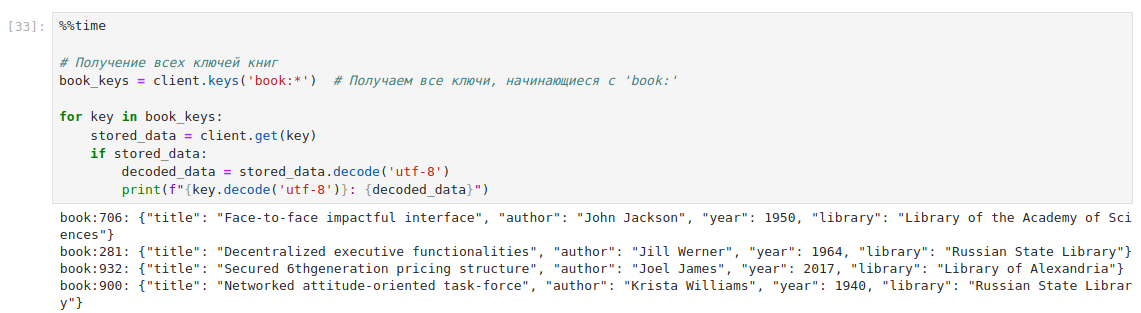


Рис. 32. Получение всех строк, которые хранятся в БД1

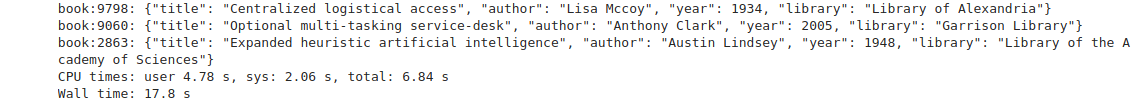


Рис. 33. Получение всех строк, которые хранятся в БД0

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.

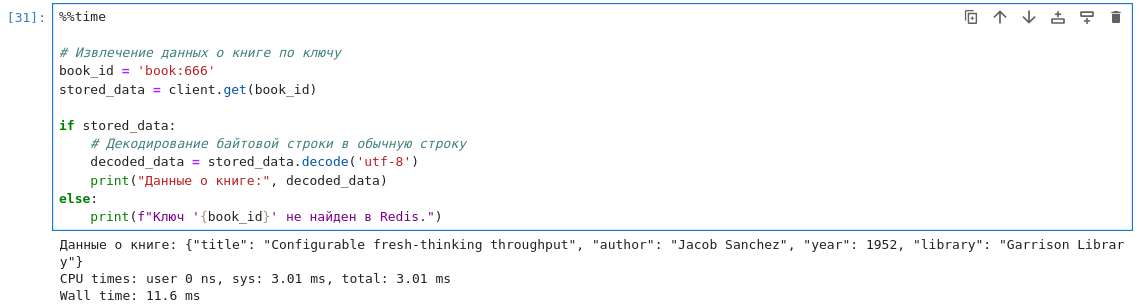


Рис. 34. Извлечение книги под номером 666 из 1000 значений

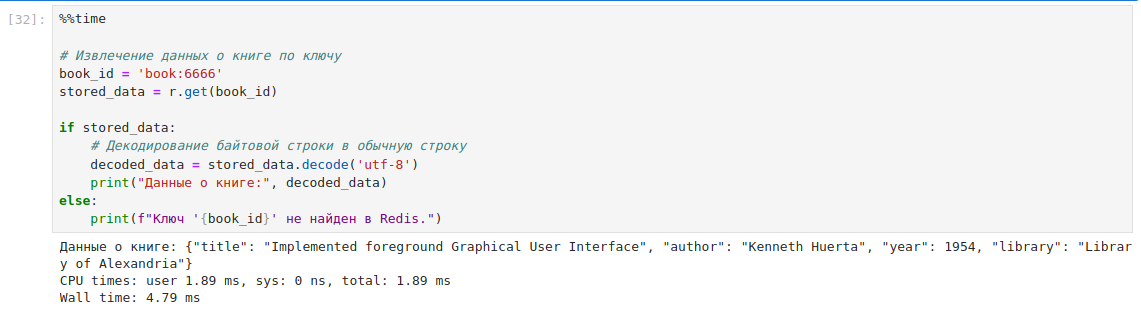


Рис. 35. Извлечение книги под номером 6666 из 10 000 значений.

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



Рис. 36. Получение первых 5 записей из 1000 значений



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.



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.

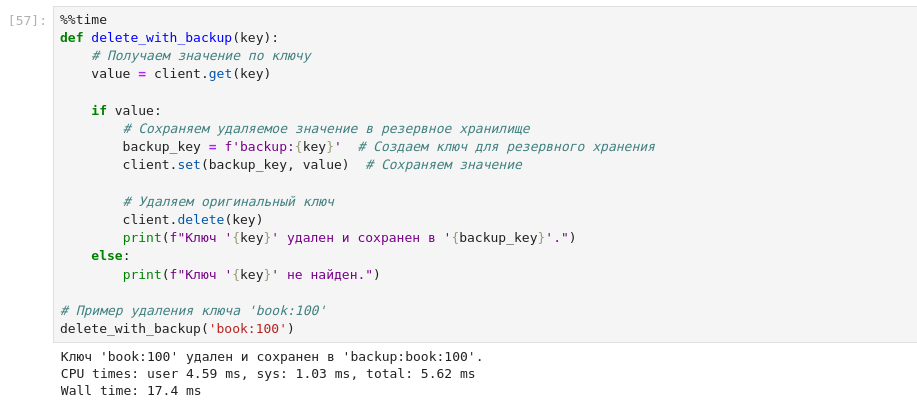


Рис. 39. Удаление строки по ключу из 1000 значений

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

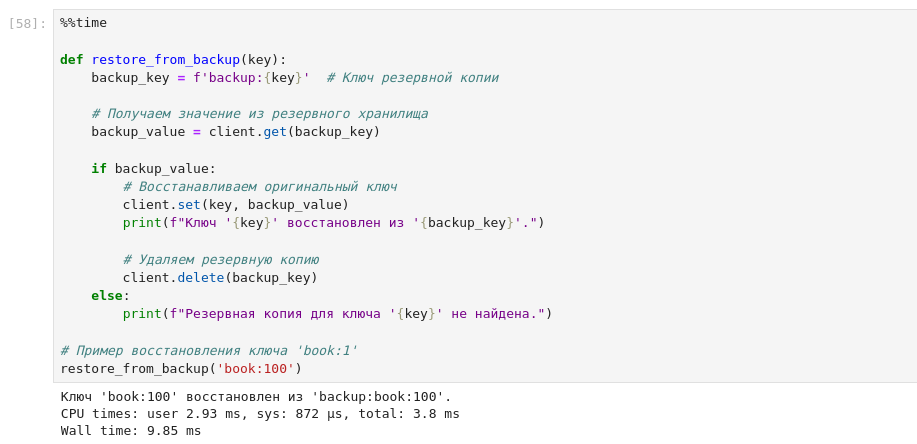


Рис. 40. Восстановление удаленной строки из 1000 записей

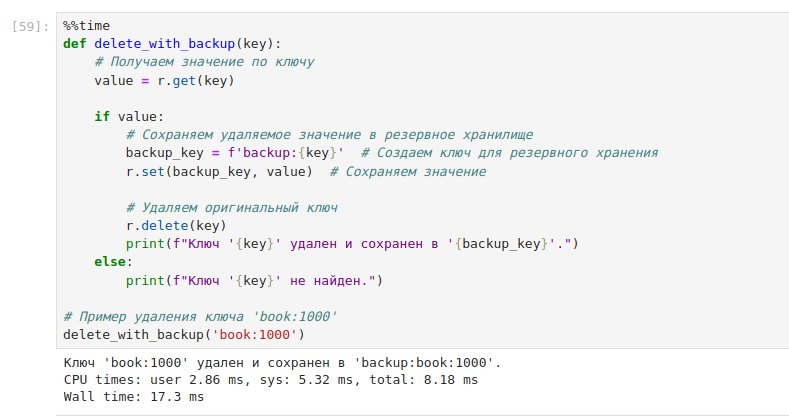


Рис. 41. Удаление строки по ключу из 10 000 значений



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 наполнить базу данных «Учебные курсы» данными о курсах, студентах, которые на них обучаются и сотрудниках, которые их разрабатывают и преподают.

Формируем запрос в консоли.



Рис. 43. Формирование запроса в консоли

После отработки запроса образуется графовая СУБД с заданными параметрами.

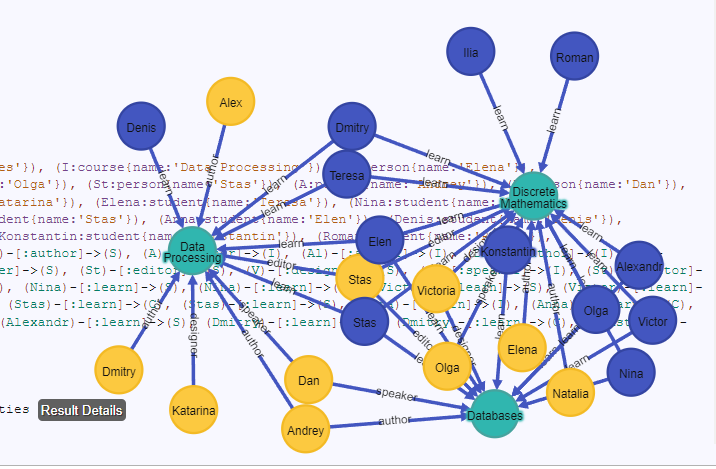


Рис. 44. Представление графовой СУБД на основе добавленных данных

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.

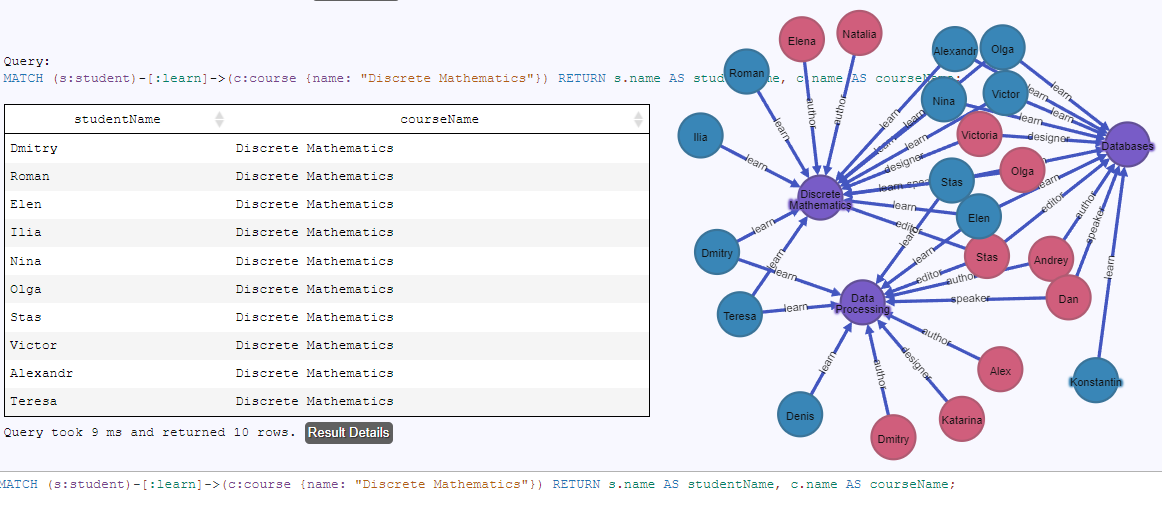


Рис. 45. Запрос 1

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;

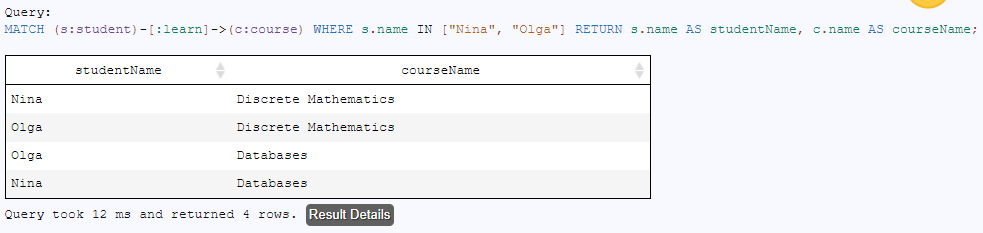


Рис. 46. Запрос 2

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;

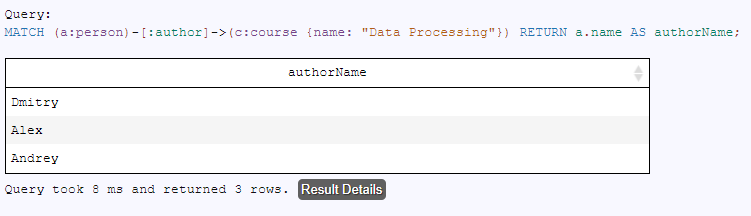


Рис. 47. Запрос 3

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;

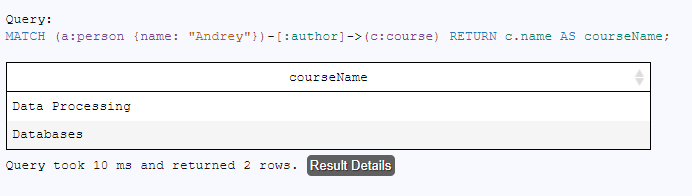


Рис. 48. Запрос 4

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;

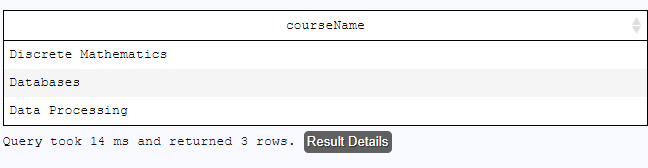


Рис. 49. Запрос 5

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.

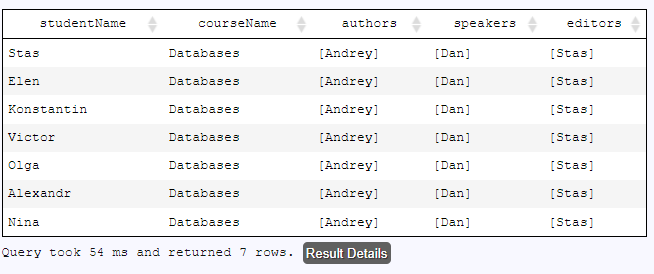


Рис. 50. Запрос 6

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;

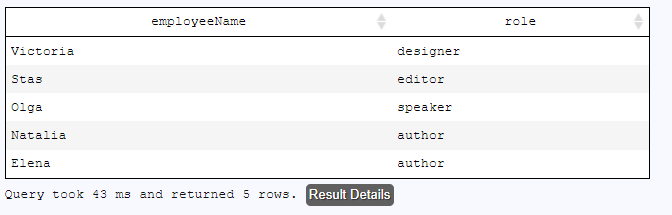


Рис. 51. Запрос 7

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

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.

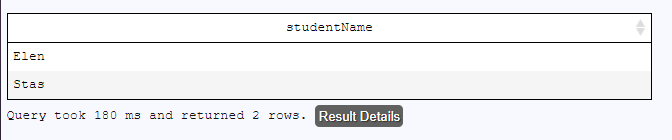


Рис. 52. Запрос 8

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;

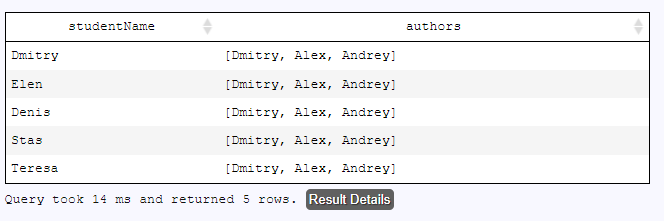


Рис. 53. Запрос 9

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.

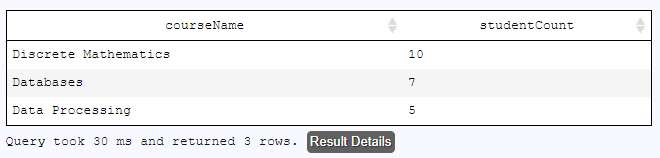


Рис. 54. Запрос 10

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;

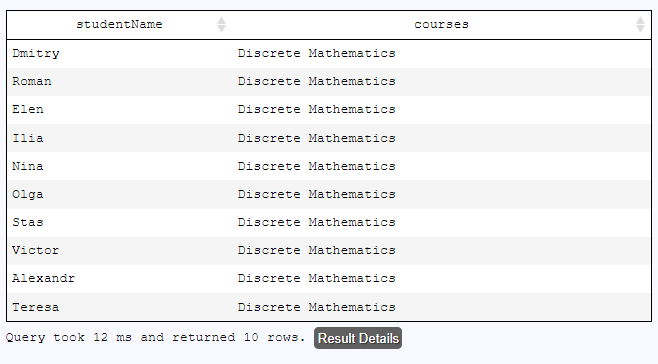


Рис. 55. Запрос 11

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;

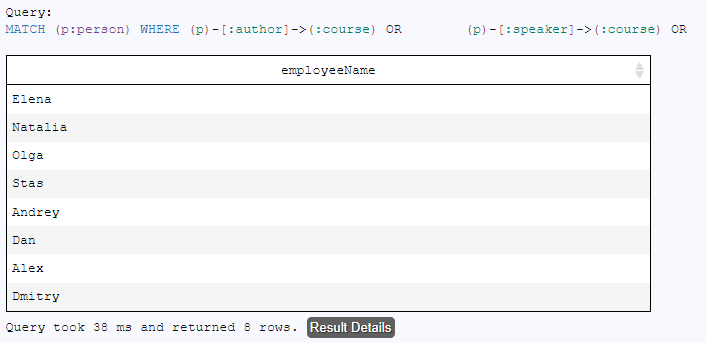


Рис. 56. Запрос 12

13. Составьте запрос, который вернет список студентов, записанных на курсы, где "Katarina" является дизайнером.

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

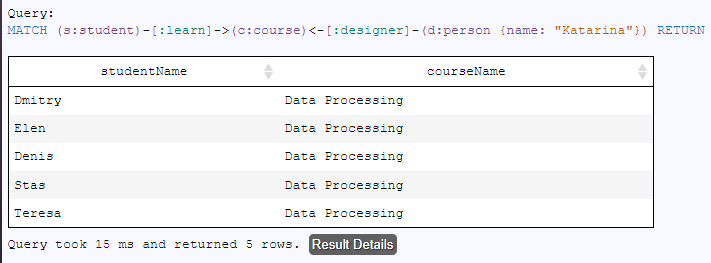


Рис. 57. Запрос 13

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;

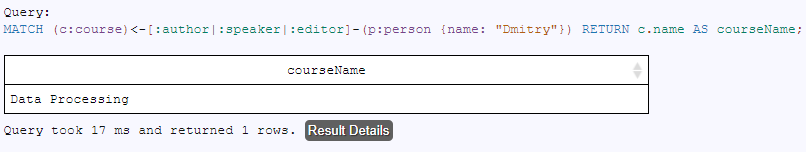


Рис. 58. Запрос 14

15. Найдите всех студентов, которые не записаны ни на один курс.

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

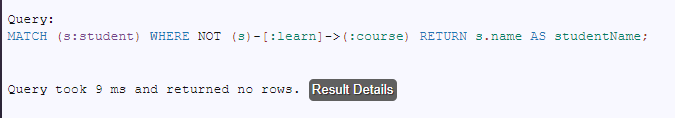


Рис. 59. Запрос 15

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;

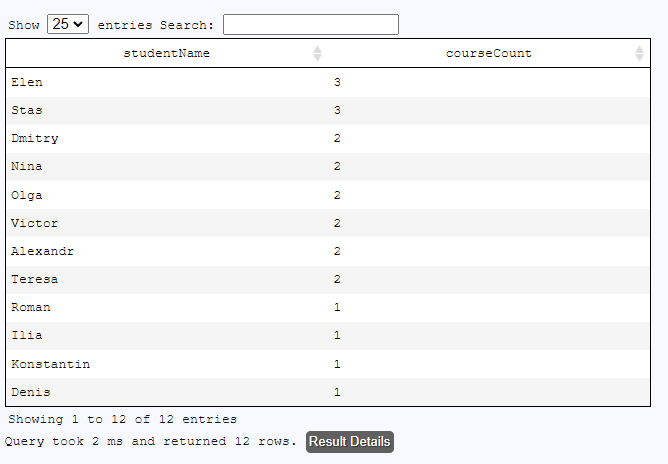


Рис. 60. Запрос 16

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;

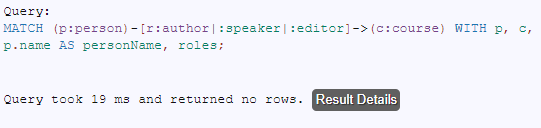


Рис. 61. Запрос 17

Таких курсов не оказалось.

18. Напишите запрос, который вернет список всех студентов, записанных на курс "Discrete Mathematics", и их наставников.

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

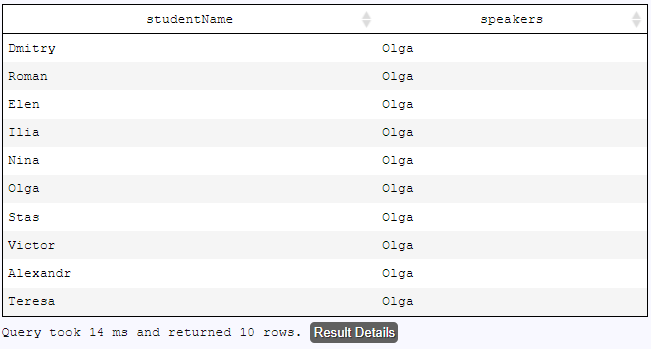


Рис. 62. Запрос 18

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;

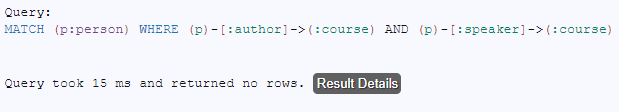


Рис. 63. Запрос 19

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

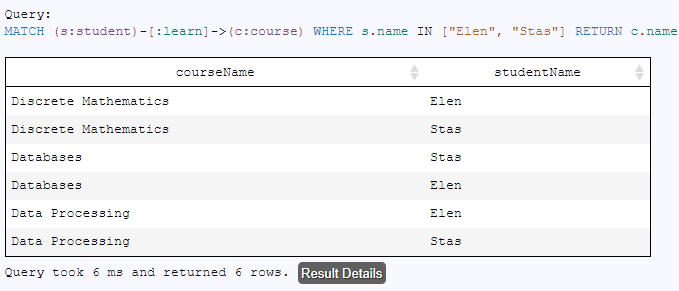


Рис. 64. Запрос 20

21. Составьте запрос, чтобы получить список студентов, которые записаны на курс "Databases" и имеют того же наставника, что и "Nina".

MATCH (n:student {name: "Nina"})-[:learn]->(c:course)<-[:speaker]-(d:person) MATCH (s:student)-[:learn]->(c) RETURN DISTINCT s.name AS studentName, d.name AS speakerName;

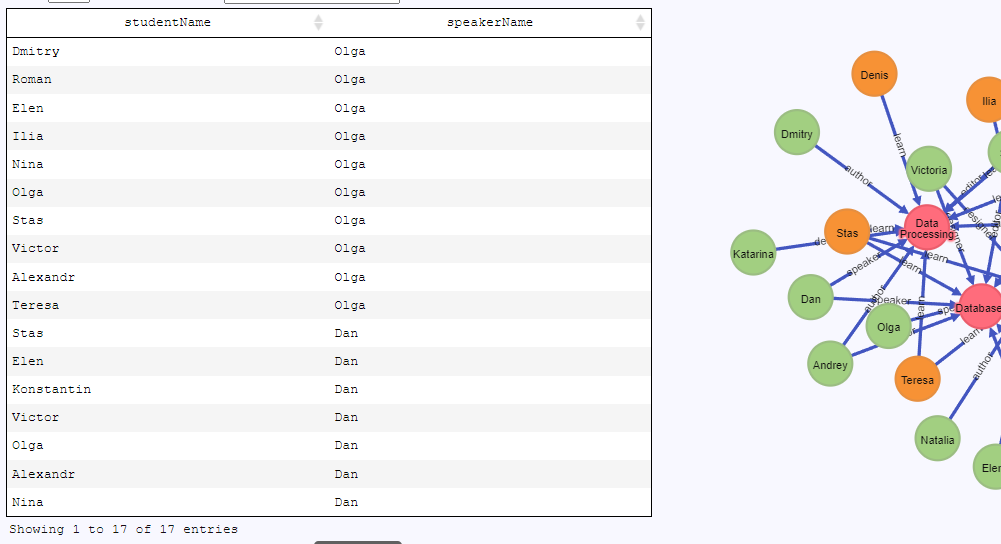


Рис. 65. Запрос 21

22. Напишите запрос, который вернет список курсов, на которых "Roman" является студентом, а "Olga" - сотрудником (в любой роли).

MATCH (r:student {name: "Roman"})-[:learn]->(c:course) MATCH (o:person {name: "Olga"})-[:author|:speaker|:editor]->(c) RETURN DISTINCT c.name AS courseName;

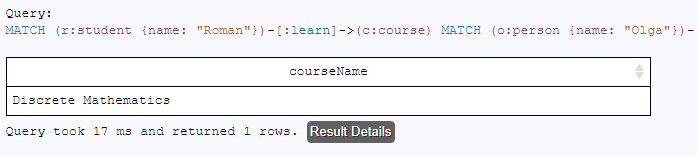


Рис. 66. Запрос 22

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;

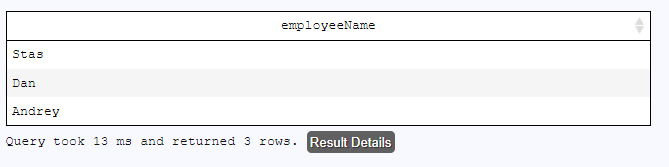


Рис. 67. Запрос 23.

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;

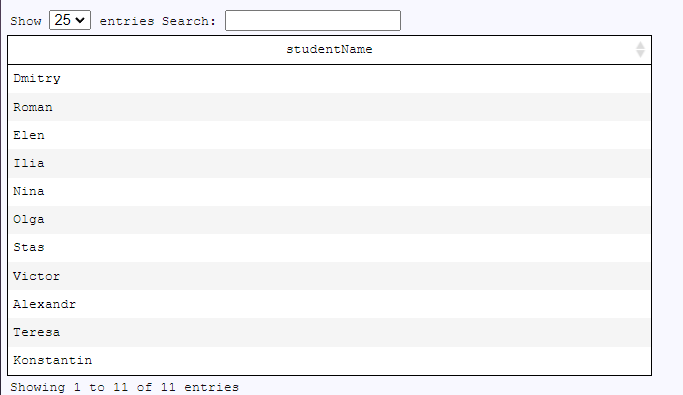


Рис. 68. Запрос 24

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;

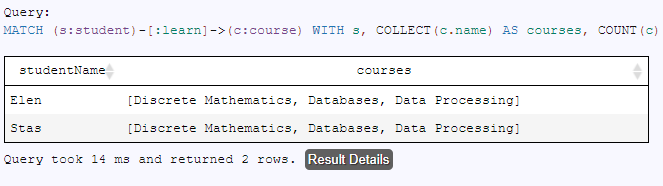


Рис. 69. Запрос 25

**Выводы по работе в СУБД Neoj4.**

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.

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

**Особенности при работе с NoSQL СУБД**

Неструктурированность. В NoSQL-базах структура данных не регламентирована вообще или лишь в малой степени. Если нужно внести изменения в поля отдельного документа, для этого не потребуется декларативно менять всю структуру таблицы. При необходимости поменять модель данных потребуется указать изменения в коде приложения.

Агрегация данных. В то время как реляционные БД сохраняют данные в виде таблиц, в нереляционных они представляют собой целостные объекты.

Гибкость – нереляционные БД могут работать с неструктурированными данными и различными моделями представления информации.

Широкая применимость – горизонтальная масштабируемость и производительность позволяют применять нереляционные БД в обработке больших данных.

Горизонтальная масштабируемость – для увеличения производительности достаточно добавить новый сервер в систему, а не наращивать мощности уже имеющихся.

**Недостатки NoSQL СУБД**

Ограниченность языка – встроенные языковые возможности нереляционных баз данных, из-за чего в работе с ними часто приходится использовать сторонние инструменты для трансляции стандартных SQL-команд.

Недостаточная надежность транзакций – из-за того, что NoSQL-БД заточены под высокую производительность и масштабируемость, в них страдает согласованность данных.