

Департамент образования и науки города Москвы

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

Институт цифрового образования

Департамент информатики, управления и технологий

Лабораторная работа 2.1.

Изучение методов хранения данных на основе NoSQL.

Выполнил студент группы АДЭУ-221

Джамалова Сабина Шахиновна

Проверил доцент

Босенко Тимур Муртазович

Москва

2025

Оглавление	
Введение	3
Шаг 1. Подготовка рабочего окружения на виртуальной машине.	3
Шаг 2. Доступ к инструментам управления	3
Работа с MongoDB	5
Задания для работы с MongoDB	16
Redis	23
Задание для индивидуального выполнения на Python	28
Neo4J	30
Дополнительное задание Neo4j	33
Индивидуальное задание для самостоятельной работы	41
Задание 1 (MongoDB).	41
Задание 2 (Neo4j)	43
Задание 3 (Redis).	45
Шаг 4. Программное взаимодействие с базами данных (Python)	47
4.1. Работа с MongoDB через pymongo	47
4.2. Работа с Redis через redis-py	49
4.3. Работа с Neo4j через neo4j-driver	51
Выводы	53

Введение

Цель работы: изучение и применение трех типов NoSQL баз данных: документо-ориентированной (MongoDB), графовой (Neo4j) и ключ-значение (Redis). Научиться создавать, заполнять и анализировать структуры данных в каждой из систем, а также выполнять запросы для получения необходимой информации, развивая навыки работы с нереляционными моделями данных.

Вариант задания: 6

Задание 1 (MongoDB). Найти все фильмы жанров "Family" или "Mystery" (\$in) и добавить им в массив genres новый тег "classic" (\$addToSet).

Задание 2 (Neo4j). Найти всех режиссеров, которые также снимались в фильмах в качестве актеров.

Задание 3 (Redis). Смоделировать корзину покупок: для пользователя user:205 в хэш cart:205 добавить 2 товара (product_id и quantity). Увеличить количество одного из товаров на 2 (HINCRBY).

Выполнение:

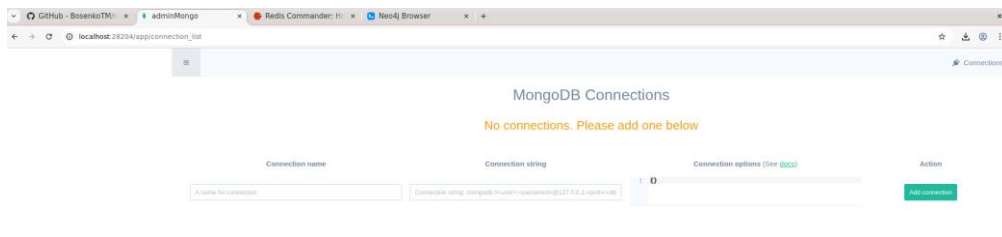
Шаг 1. Подготовка рабочего окружения на виртуальной машине.

1. Переход в необходимую директорию и запуск контейнеров:

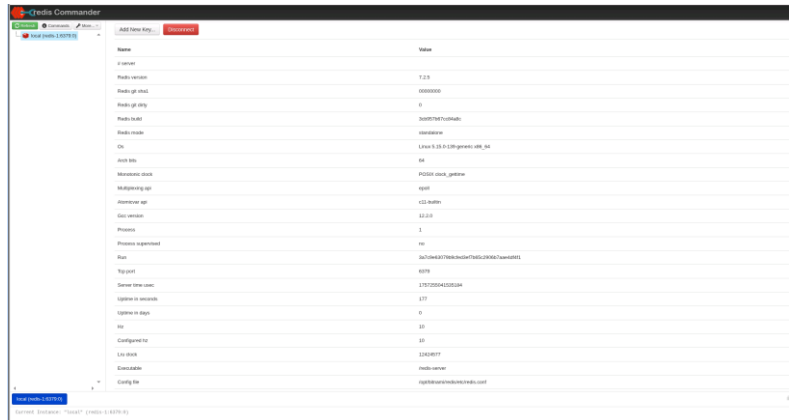
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
● mgpu@mgpu-vm:~$ cd /home/mgpu/Downloads/idb/nosql-workshop/01-environment/docker
● mgpu@mgpu-vm:~/Downloads/idb/nosql-workshop/01-environment/docker$ sudo docker compose up -d
[sudo] password for mgpu:
[+] Running 10/10
  ✓ Network nosql-platform      Created
  ✓ Container admin-mongo       Started
  ✓ Container cassandra-1       Started
  ✓ Container jupyter           Started
  ✓ Container neo4j-1           Started
  ✓ Container redis-commander    Started
  ✓ Container redis-1           Started
  ✓ Container mongo-express      Started
  ✓ Container cassandra-web      Started
  ✓ Container mongo-1           Started
○ mgpu@mgpu-vm:~/Downloads/idb/nosql-workshop/01-environment/docker$
```

Шаг 2. Доступ к инструментам управления

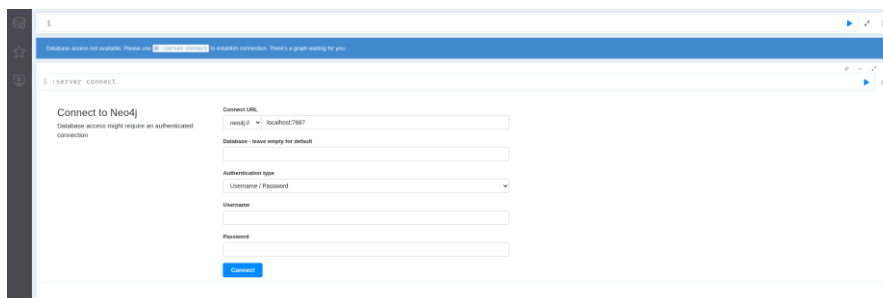
1. MongoDB



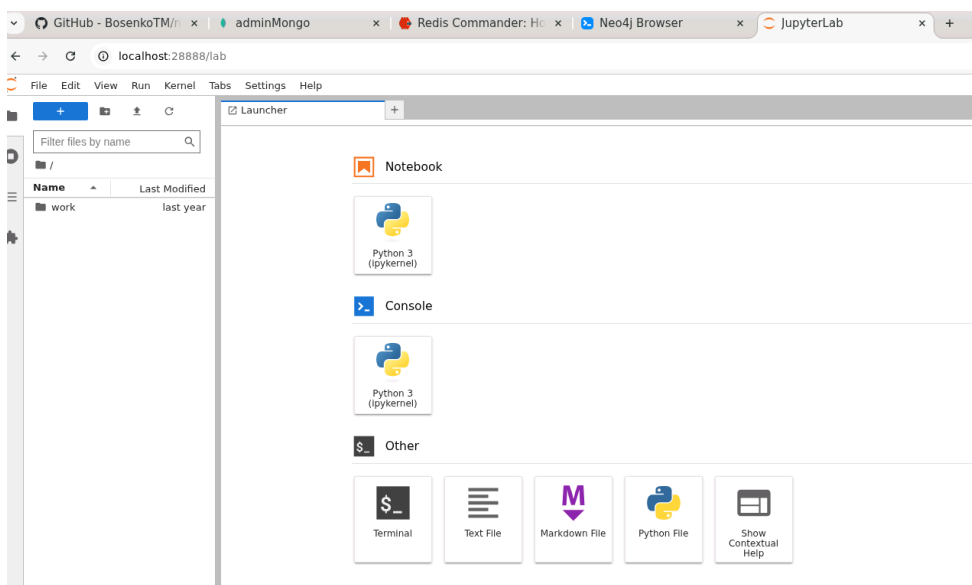
2. Redis



3. Neo4j



4. Python/Jupyter



Шаг 3. Выполнение заданий

Работа с MongoDB

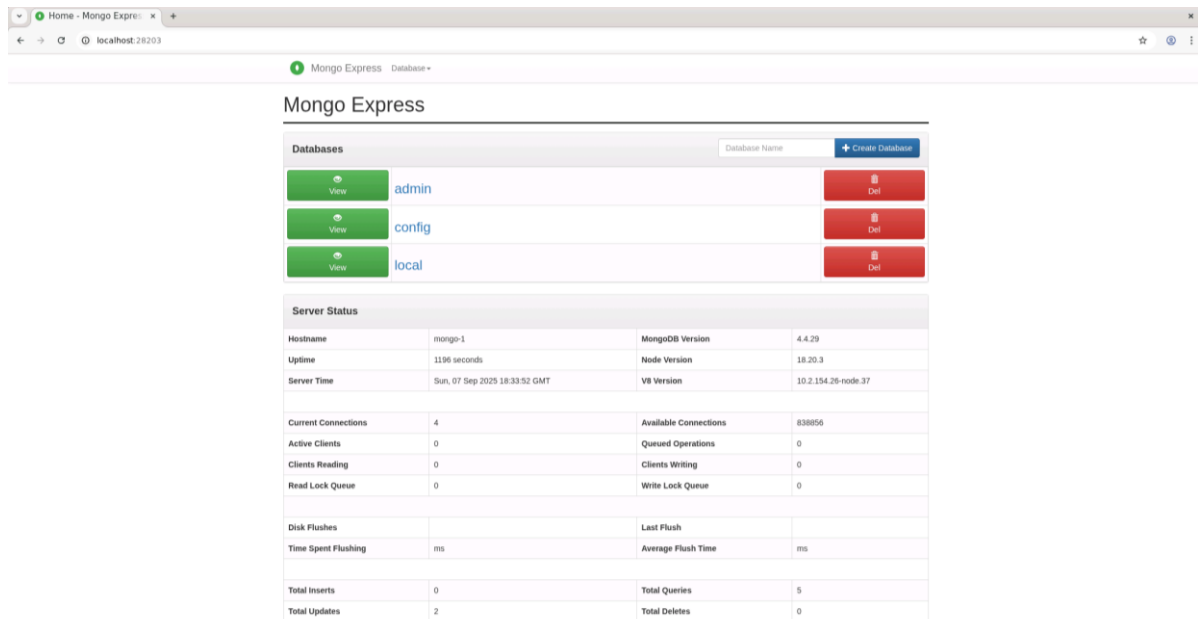
Использование утилиты командной строки MongoDB

`sudo docker exec -ti mongo-1 mongo -u "root" -p "abc123!"`

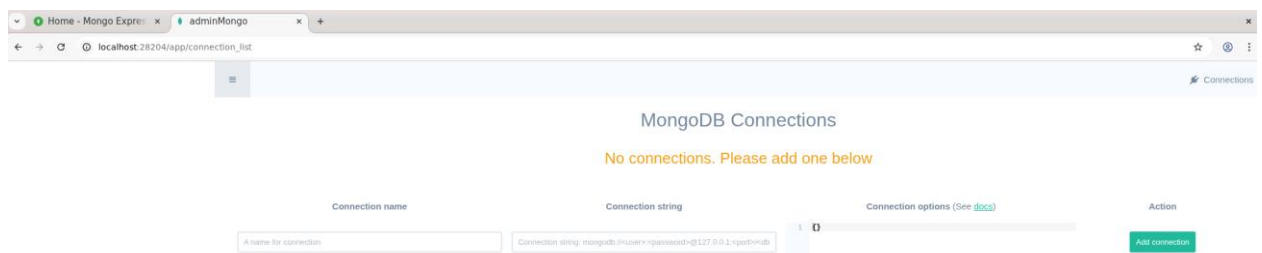
```
ngpu@ngpu-vm:~/Downloads/ldb/nosql-workshop/01-environment/docker$ sudo docker exec -ti mongo-1 mongo -u "root" -p "abc123!"
MongoDB shell version v4.4.29
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("c5dc0624-3430-4037-ab51-6da1cc1ff086") }
MongoDB server version: 4.4.29
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
  https://community.mongodb.com
...
The server generated these startup warnings when booting:
  2025-09-07T18:13:56.478+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
>
```

Использование браузерного графического интерфейса

Mongo Express



Admin Mongo



Добавление подключения:

adminMongo

Connections

MongoDB Connections

Connection name	Connection string	Connection options	Action
Data Platform	mongodb://mongo-1:27017	{}	<div>DeleteConnectUpdate</div>
A name for connection	Connection string: mongodb://user:<password>@127.0.0.1:<port>/?db=	1 {}	<div>Add Connection</div>

Практическая работа 1. Создание документов в MongoDB

Подключение

New Connection

Manage your connection settings

URI ⓘ

Edit Connection String ⚙

mongodb://root:abc123!@localhost:27017

Name

localhost:27017

Color

No Color ▾

☐ Favorite this connection

Favoriting a connection will pin it to the top of your list of connections

➤ Advanced Connection Options

How do I find my connection string in Atlas?

If you have an Atlas cluster, go to the Cluster view. Click the 'Connect' button for the cluster to which you wish to connect. [See example](#) ⓘ

How do I format my connection string?

[See example](#) ⓘ

Cancel

Save

Connect

Save & Connect

Создание БД

Create Database

Database Name

filmdb

Collection Name

movies

➤ Additional preferences (e.g. Custom collation, Clustered collections)



Cancel

Create Database

Вставка и ее проверка

Insert Document

To collection filmdb.movies

VIEW  

1 {

2 "id": "0110912",

3 "title": "Pulp Fiction",

4 "year": 1994,

5 "runtime": 154,

6 "languages": ["en", "es", "fr"],

7 "rating": 8.9,

8 "votes": 2084331,

9 "genres": ["Crime", "Drama"],

10 "plotOutline": "Jules Winnfield (Samuel L. Jackson) and

11 "coverUrl": "https://m.media-amazon.com/images/M/MV5BNG

12 "actors": [

13 { "actorID": "0000619", "name": "Tim Roth"},

14 { "actorID": "0001625", "name": "Amanda Plummer"},

15 { "actorID": "0522503", "name": "Laura Lovelace"},

16 { "actorID": "0000237", "name": "John Travolta"},

17 { "actorID": "0000168", "name": "Samuel L. Jackson"},

18 { "actorID": "0482851", "name": "Phil LaMarr"},

19 { "actorID": "0001844", "name": "Frank Whaley"},

20 { "actorID": "0824882", "name": "Burr Steers"},

21 { "actorID": "0000246", "name": "Bruce Willis"},

22 { "actorID": "0000609", "name": "Ving Rhames"},

23 { "actorID": "0000235", "name": "Uma Thurman"},

24 { "actorID": "0000233", "name": "Quentin Tarantino"},

25],

26 "directors": [

localhost:27017

admin

temproles

tempusers

config

filmdb

movies

local

Cancel

Insert

Поиск документа

MongoDB Compass - localhost:27017/filmdb.movies

Connections Edit View Collection Help

localhost:27017 movies +

localhost:27017 > filmdb > movies [Open MongoDB shell](#)

Documents Aggregations Schema Indexes Validation

{ "title": "Pulp Fiction" }

[Generate query](#) [Explain](#) [Reset](#) [Find](#) [Options](#)

[ADD DATA](#) [EXPORT DATA](#) [UPDATE](#) [DELETE](#) [INSIGHT](#)

25 1-1 of 1

```
_id: ObjectId('68bdd57cd09d46b0d6e711e')
id: "0110912"
title: "Pulp Fiction"
year: 1994
runtime: 154
languages: Array (3)
rating: 8.9
votes: 2084331
genres: Array (2)
plotOutline: "Jules Winnfield (Samuel L. Jackson) and Vincent Vega (John Travolta) a..."
coverUrl: "https://m.media-amazon.com/images/M/MV5BNGNlMDIzZTU0NTB1Z100NTRLLWFjM2..."
actors: Array (12)
directors: Array (1)
producers: Array (7)
```

Добавление новых документов

```
< {
  acknowledged: true,
  insertedId: ObjectId('68bdd8b5591415381d480991')
}
> db.getCollectionNames()
< [ 'movies' ]
test> |
```

Список документов

```
> db.movies.find()
< {
  _id: ObjectId('68bdd72d6b8a24866165e3e8'),
  id: '0133093',
  title: 'The Matrix',
  year: 1999,
  runtime: 136,
  languages: [
    'en'
  ],
  rating: 8.7,
  votes: 1496538,
  genres: [
    'Action',
    'Sci-Fi'
  ],
  plotOutline: "Thomas A. Anderson is a man living two lives. By day he is an average computer programmer and by night a hacker known as Neo",
  coverUrl: 'https://m.media-amazon.com/images/M/MV5BNzQzOTk3OTAtNDQ0Zi00ZTVkLWI0MTETMDllZjNkYzNjNTc4L2ltYWdlLXkEYXkFqcGdeQXVyNjU0OTQ0OTY@._v1',
  actors: [
    {
      actorID: '0000206',
      name: 'Keanu Reeves'
    }
  ]
}
```

Проверка индексации поля `_id` (команда отличается для текущей версии mongo)

```
> db.getCollection('movies').getIndexes()
< [ { v: 2, key: { _id: 1 }, name: '_id_' } ]
test> |
```

Проверка добавления людей в коллекцию `persons`

```
> db.persons.find().count()
< 4
```

Проверка ввода недопустимого документа

```
> db.persons.insertOne (
  {
    "id: 0000113,
    "name": "Invalid Actor"
  })
✖ SyntaxError: Unterminated string constant. (3:0)

1 | db.persons.insertOne (
2 | {
> 3 | "id: 0000113,
   | ^
4 | "name": "Invalid Actor"
5 | })
```

Запрос документов с помощью селектора запросов

Проверка мультивставки

```
test> > db.movies.find().count()  
50
```

Получить все Family фильмы

```
> db.movies.find({"genres": "Family"})  
< {  
  _id: ObjectId('68bddd00591415381d4809ac'),  
  id: '0038650',  
  title: "It's a Wonderful Life",  
  genres: [  
    'Drama',  
    'Family',  
    'Fantasy'  
  ],  
  year: 1946,  
  rating: 8.6,  
  rank: 25  
}  
{  
  _id: ObjectId('68bddd00591415381d4809ae'),  
  id: '0245429',  
  title: 'Spirited Away',  
  genres: [  
    'Animation',  
    'Adventure',  
    'Family',  
    'Fantasy',  
    'Mystery'
```

Все фильмы, которые были опубликованы в 2010 году и позже

```
> db.movies.find({"genres": "Action", "year": { $gte : 2010 } })  
< {  
  _id: ObjectId('68bddd00591415381d48099f'),  
  id: '4154796',  
  title: 'Avengers: Endgame',  
  genres: [  
    'Action',  
    'Adventure',  
    'Fantasy',  
    'Sci-Fi'  
  ],  
  year: 2019,  
  rating: 8.8,  
  rank: 11  
}  
{  
  _id: ObjectId('68bddd00591415381d4809a3'),  
  id: '1375666',  
  title: 'Inception',  
  genres: [  
    'Action',  
    'Adventure',  
    'Sci-Fi',  
    'Thriller'
```

Все фильмы, которые not относятся к жанру Drama

```
> db.movies.find({"genres": { $ne: "Drama" } })
< {
  _id: ObjectId('68bdd72d6b8a24866165e3e8'),
  id: '0133093',
  title: 'The Matrix',
  year: 1999,
  runtime: 136,
  languages: [
    'en'
  ],
  rating: 8.7,
  votes: 1496538,
  genres: [
    'Action',
    'Sci-Fi'
  ],
  plotOutline: "Thomas A. Anderson is a man living two lives. By day he is an average computer programmer and by night a hacker known as Nec
  coverUrl: 'https://m.media-amazon.com/images/M/MV5BNzQzOTk3OTAtNDQ0Zi00ZTVkLWI0MTETMDllZjNkVzNjNTc4L2ltYWdlLXkEYXkFqcGdeQXVyNjU0OTQ0OTY0._v
  actors: [
    {
      actorID: '0000206',
      name: 'Keanu Reeves'
    },
    {
      actorID: '0000401',
```

Оператор \$exists для проверки наличия или отсутствия поля

```
> db.movies.find({ "plotOutline": { $exists: true } })
< {
  _id: ObjectId('68bdd72d6b8a24866165e3e8'),
  id: '0133093',
  title: 'The Matrix',
  year: 1999,
  runtime: 136,
  languages: [
    'en'
  ],
  rating: 8.7,
  votes: 1496538,
  genres: [
    'Action',
    'Sci-Fi'
  ],
  plotOutline: "Thomas A. Anderson is a man living two lives. By day he is an average computer programmer and by night a hacker known as Nec
  coverUrl: 'https://m.media-amazon.com/images/M/MV5BNzQzOTk3OTAtNDQ0Zi00ZTVkLWI0MTETMDllZjNkVzNjNTc4L2ltYWdlLXkEYXkFqcGdeQXVyNjU0OTQ0OTY0._v
  actors: [
    {
      actorID: '0000206',
      name: 'Keanu Reeves'
    },
    {
      actorID: '0000401',
      name: 'Laurence Fishburne'
```

Оператор \$in для сопоставления одного из нескольких значений, передаваемых как массив

```
> db.movies.find({ "genres": { $in: ['Family', 'Mystery']} })
< {
  _id: ObjectId('68bdd00591415381d4809ac'),
  id: '0038650',
  title: "It's a Wonderful Life",
  genres: [
    'Drama',
    'Family',
    'Fantasy'
  ],
  year: 1946,
  rating: 8.6,
  rank: 25
}
{
  _id: ObjectId('68bdd00591415381d4809ae'),
  id: '0245429',
  title: 'Spirited Away',
  genres: [
    'Animation',
    'Adventure',
    'Family',
    'Fantasy',
    'Mystery'
  ],
}
```

Все фильмы жанра Music OR которые были выпущены в 2012 году или позже

```
> db.movies.find({ $or: [ { "genres": "Music" }, { "year": { $gte : 2012 } } ] })
< {
  _id: ObjectId('68bddd00591415381d48099f'),
  id: '4154796',
  title: 'Avengers: Endgame',
  genres: [
    'Action',
    'Adventure',
    'Fantasy',
    'Sci-Fi'
  ],
  year: 2019,
  rating: 8.8,
  rank: 11
}
{
  _id: ObjectId('68bddd00591415381d4809b3'),
  id: '0816692',
  title: 'Interstellar',
  genres: [
    'Adventure',
    'Drama',
    'Sci-Fi'
  ],
  year: 2014,
  rating: 8.5,
  rank: 32
}
```

Все фильмы жанра Action AND которые были выпущены в 2010 году или позже OR имеют рейтинг выше 8.8

```
> db.movies.find( { "genres": "Action", $or: [ { "year": { $gte : 2010 } }, { "rating": { $gt : 8.8 } } ] })
< {
  _id: ObjectId('68bddd00591415381d480999'),
  id: '0468569',
  title: 'The Dark Knight',
  genres: [
    'Action',
    'Crime',
    'Drama',
    'Thriller'
  ],
  year: 2008,
  rating: 9,
  rank: 4
}
{
  _id: ObjectId('68bddd00591415381d48099f'),
  id: '4154796',
  title: 'Avengers: Endgame',
  genres: [
    'Action',
    'Adventure',
    'Fantasy',
    'Sci-Fi'
  ],
  year: 2019,
```

Поиск по _id

```

> db.movies.find( {_id: ObjectId("68bddd00591415381d4809a3")})
< {
  _id: ObjectId('68bddd00591415381d4809a3'),
  id: '1375666',
  title: 'Inception',
  genres: [
    'Action',
    'Adventure',
    'Sci-Fi',
    'Thriller'
  ],
  year: 2010,
  rating: 8.7,
  rank: 15
}
test>

```

Обновление документов

Изменение рейтинга фильма

```

> db.movies.updateOne ( {title: 'Fight Club'} , { $set: {rating: 9} } )
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
test>

```

Изменение голосов

```

> db.movies.find( {title: 'The Matrix'}, {"votes":1})
< {
  _id: ObjectId('68bdd72d6b8a24866165e3e8'),
  votes: 1496538
}
{
  _id: ObjectId('68bdd8b5591415381d480991'),
  votes: 1496538
}
> db.movies.updateOne( {title: 'The Matrix'} , { $inc: {votes: 1} } )
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
> db.movies.find( {title: 'The Matrix'}, {"votes":1})
< {
  _id: ObjectId('68bdd72d6b8a24866165e3e8'),
  votes: 1496539
}

```

Оптимизация производительности с помощью индексов

```
> db.movies.createIndex( {title: 1} );
< title_1
> db.movies.find ( {title: "The Matrix"} );
< {
  _id: ObjectId('68bdd72d6b8a24866165e3e8'),
  id: '0133093',
  title: 'The Matrix',
  year: 1999,
  runtime: 136,
  languages: [
    'en'
  ],
  rating: 8.7,
  votes: 1496539,
  genres: [
    'Action',
    'Sci-Fi'
  ],
}
```

```
> db.movies.find ( {title: "The Matrix"} ).explain();
< {
  queryPlanner: {
    plannerVersion: 1,
    namespace: 'test.movies',
    indexFilterSet: false,
    parsedQuery: {
      title: {
        '$eq': 'The Matrix'
      }
    },
    queryHash: '6E0D6672',
    planCacheKey: 'B1CDA929',
    winningPlan: {
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: {
          title: 1
        },
        indexName: 'title_1',
        isMultiKey: false,
        multiKeyPaths: {
          title: []
        },
      },
    },
  },
}
```

Уникальные индексы:

```
> db.movies.createIndex( {id: 1}, {unique: true} );
< id_1
test> |
```

Все индексы

```
> db.movies.getIndexes()
< [
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { title: 1 }, name: 'title_1' },
  { v: 2, unique: true, key: { id: 1 }, name: 'id_1' }
]
test>
```

Удаление индексов

```
> db.movies.dropIndex( {title: 1} );
< { nIndexesWas: 3, ok: 1 }
```

Текстовый поиск

Разрешение текстового поиска

```
> db.movies.createIndex ( { title: "text", plotOutline: "text" }
< title_text_plotOutline_text
> db.movies.find( { $text: { $search: "fight" } } )
< {
  _id: ObjectId('68bddd00591415381d48099e'),
  id: '0137523',
  title: 'Fight Club',
  genres: [
    'Drama'
  ],
  year: 1999,
  rating: 9,
  rank: 10
}
```

Поиск

```
> db.movies.find( { $text: { $search: "fight terrorist" } } )
< {
  _id: ObjectId('68bddd00591415381d48099e'),
  id: '0137523',
  title: 'Fight Club',
  genres: [
    'Drama'
  ],
  year: 1999,
  rating: 9,
  rank: 10
}
{
  _id: ObjectId('68bdd72d6b8a24866165e3e8'),
  id: '0133093',
  title: 'The Matrix',
  year: 1999,
  runtime: 136,
  languages: [
    'en'
  ],
  rating: 8.7,
  votes: 1496539,
  genres: [
    'Action',
  ],
}
```

Агрегация данных

Сколько фильмов есть для разных рейтингов

```
> db.movies.aggregate( [{ $group: { _id: '$rating', total: { $sum: 1 } } } ] )
< {
  _id: 8.7,
  total: 7
}
{
  _id: 8.5,
  total: 23
}
{
  _id: 8.8,
  total: 3
}
{
  _id: 8.4,
  total: 1
}
{
  _id: 9.2,
  total: 2
}
{
  _id: 9,
  total: 3
}
{
  _id: 8.9,
  total: 3
}
}
```

Группируем по genres и подсчитываем количество фильмов для каждого жанра

```
> db.movies.aggregate([
  { $match: { year: { $gt: 2000 } } },
  { $unwind: "$genres" },
  { $group: { _id: '$genres',
    number: { $sum: 1 },
    minRating: { $min: '$rating' },
    maxRating: { $max: '$rating' },
    avgRating: { $avg: '$rating' }
  } },
  { $sort: { number: -1 } } ])
< {
  _id: 'Drama',
  number: 11,
  minRating: 8.5,
  maxRating: 9,
  avgRating: 8.636363636363637
}
{
  _id: 'Adventure',
  number: 7,
  minRating: 8.5,
  maxRating: 8.9,
  avgRating: 8.7
}
{
  _id: 'Fantasy',
  number: 5,
  minRating: 8.5,
  maxRating: 8.9,
}
```

Удаление документов

```
> db.movies.deleteOne( { "title": "Fight Club" } )
< {
  acknowledged: true,
  deletedCount: 1
}
> db.movies.deleteMany( { "plotOutline": { $exists: false } } )
< {
  acknowledged: true,
  deletedCount: 47
}
```

Задания для работы с MongoDB

Вариант 6. Ресторан - меню и заказы

Создание базы и коллекции

Create Database ✕

Database Name

Collection Name

☐ **Time-Series**
Time-series collections efficiently store sequences of measurements over a period of time. [Learn More](#)

➤ **Additional preferences** (e.g. Custom collation, Clustered collections)

Создание заказа

```
function createOrder(orderData) {

    // Автоматическая генерация order_id

    var lastOrder = db.orders.find().sort({order_id: -1}).limit(1);

    var nextOrderId = lastOrder.hasNext() ?
        parseInt(lastOrder.next().order_id) + 1 : 1001;

    var order = {

        order_id: nextOrderId.toString(),
```



```

        table: orderData.table,

        customer_name: orderData.customer_name,

        items: orderData.items,

        total: calculateTotal(orderData.items),

        status: "принят",

        order_time: new Date(),

        notes: orderData.notes || ""

    };

    return db.orders.insertOne(order);
}

function calculateTotal(items) {

    return items.reduce(function(total, item) {

        return total + (item.price * item.quantity);

    }, 0);

}

```

Обновление статуса заказа

```

function updateOrderStatus(orderId, newStatus) {

    var validStatuses = ["принят", "готовится", "готов",
"выдан"];

    if (!validStatuses.includes(newStatus)) {

        throw "Неверный статус заказа";

    }

    return db.orders.updateOne(

        { order_id: orderId },

        { $set: { status: newStatus } }
    );
}

```

```
);  
}
```

Получение активных заказов

```
function getActiveOrders() {  
    return db.orders.find({  
        status: { $in: ["принят", "готовится"] }  
    }).sort({ order_time: 1 }).toArray();  
}
```

Поиск заказов по клиенту

```
function findOrdersByCustomer(customerName) {  
    return db.orders.find({  
        customer_name: { $regex: customerName, $options: "i" }  
    }).sort({ order_time: -1 }).toArray();  
}
```

Статистика продаж по блюдам

```
function getDishStatistics(startDate, endDate) {  
    return db.orders.aggregate([  
        {  
            $match: {  
                order_time: {  
                    $gte: new Date(startDate),  
                    $lte: new Date(endDate)  
                }  
            }  
        },
```

```

    { $unwind: "$items" },
    {
      $group: {
        _id: "$items.dish",
        total_quantity: { $sum: "$items.quantity" },
        total_revenue: { $sum: { $multiply:
["$items.quantity", "$items.price"] } } },
        order_count: { $sum: 1 }
      }
    },
    { $sort: { total_revenue: -1 } }
  ]).toArray();
}

```

Примеры использования

Создание заказа

```

> var newOrder = {
  table: 5,
  customer_name: "Иван Иванов",
  items: [
    { dish: "Пицца Маргарита", quantity: 1, price: 450 },
    { dish: "Салат Цезарь", quantity: 2, price: 250 },
    { dish: "Кока-Кола", quantity: 3, price: 100 }
  ],
  notes: "Без лука в салате"
};

createOrder(newOrder);
< {
  acknowledged: true,
  insertedId: ObjectId('68bdeabb782e624d71d02317')
}

```

Обновление статуса

```

> updateOrderStatus("1001", "готовится");
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
restaurant> |

```

Получение активных заказов

```

> var activeOrders = getActiveOrders();
  print("Активные заказы:");
  activeOrders.forEach(function(order) {
    print("Заказ #" + order.order_id + " - " + order.status);
  });
< Активные заказы:
< Заказ #1001 - готовится

```

Статистика за сегодня

```

> var today = new Date();
  today.setHours(0, 0, 0, 0);
  var tomorrow = new Date(today);
  tomorrow.setDate(tomorrow.getDate() + 1);

  var stats = getDishStatistics(today, tomorrow);
  print("Статистика продаж за сегодня:");
  stats.forEach(function(item) {
    print(item._id + ": " + item.total_quantity + " шт., " + item.total_revenue + " руб.");
  });
< Статистика продаж за сегодня:
< Салат Цезарь: 2 шт., 500 руб.
< Пицца Маргарита: 1 шт., 450 руб.
< Кока-Кола: 3 шт., 300 руб.

```

Дополнительные функции:

Расчет общей суммы заказа

```

function calculateOrderTotal(items) {

  return items.reduce(function(total, item) {

    return total + (item.price * item.quantity);

```

```
    }, 0);  
}
```

```
> var orderItems = [  
  { dish: "Пицца Маргарита", quantity: 1, price: 450 },  
  { dish: "Салат Цезарь", quantity: 2, price: 250 },  
  { dish: "Кока-Кола", quantity: 3, price: 100 }  
];  
  
var totalAmount = calculateOrderTotal(orderItems);  
print("Общая сумма заказа: " + totalAmount + " руб.");  
< Общая сумма заказа: 1250 руб.
```

Статистика продаж по блюдам

```
function getDishStatistics(startDate, endDate) {  
  return db.orders.aggregate([  
    {  
      $match: {  
        order_time: {  
          $gte: new Date(startDate),  
          $lte: new Date(endDate)  
        }  
      }  
    },  
    { $unwind: "$items" },  
    {  
      $group: {  
        _id: "$items.dish",  
        total_quantity: { $sum: "$items.quantity" },  

```

```

        total_revenue: { $sum: { $multiply:
["$items.quantity", "$items.price"] } },

        average_price: { $avg: "$items.price" }

    }

},

{ $sort: { total_revenue: -1 } }

]).toArray();

}

```

```

> var today = new Date();
today.setHours(0, 0, 0, 0);
var tomorrow = new Date(today);
tomorrow.setDate(tomorrow.getDate() + 1);

var dishStats = getDishStatistics(today, tomorrow);

print("Статистика продаж за сегодня:");
print("Блюдо".padEnd(20) + "Кол-во".padEnd(10) + "Выручка".padEnd(12) + "Ср.цена");
print("-".repeat(50));

dishStats.forEach(function(dish) {
    print(dish._id.padEnd(20) +
        dish.total_quantity.toString().padEnd(10) +
        dish.total_revenue.toFixed(2).padEnd(12) +
        dish.average_price.toFixed(2) + " руб.");
});
< Статистика продаж за сегодня:
< Блюдо          Кол-во    Выручка    Ср.цена
< -----
< Салат Цезарь   2          500.00     250.00 руб.
< Пицца Маргарита 1          450.00     450.00 руб.
< Кока-Кола      3          300.00     100.00 руб.

```

Поиск заказов по клиенту

```

> var customerOrders = findOrdersByCustomer("Иван");

print("Заказы клиента Иван:");
customerOrders.forEach(function(order, index) {
    print((index + 1) + ". Заказ #" + order.order_id +
        " - " + order.order_time.toLocaleDateString() +
        " - Стол " + order.table +
        " - " + order.status +
        " - " + order.total + " руб.");

    print("    Блюда: " + order.items.map(item =>
        item.dish + " (" + item.quantity + " шт.)").join(", "));
    print("---");
});
< Заказы клиента Иван:
< 1. Заказ #1001 - 9/7/2025 - Стол 5 - готовится - 1250 руб.
<    Блюда: Пицца Маргарита (1 шт.), Салат Цезарь (2 шт.), Кока-Кола (3 шт.)
< ---

```

Redis

Запуск

```
ngpu@ngpu-vm: ~/Downloads/idb/nosql-workshop/01-environment/docker$ docker run -it --rm --network nosql-platform bitnami/redis redis-cli -h redis-1 -p 6379
Unable to find image 'bitnami/redis:latest' locally
latest: Pulling from bitnami/redis
ce4f4d45400a: Pull complete
Digest: sha256:25bf63f3caf75af4628c0dfcf39059ad1ac8abe135be85e99699f9637b16dc28
Status: Downloaded newer image for bitnami/redis:latest
redis 21:12:26.81 INFO ==>
redis 21:12:26.81 INFO ==> Welcome to the Bitnami redis container
redis 21:12:26.81 INFO ==> Subscribe to project updates by watching https://github.com/bitnami/containers
redis 21:12:26.82 INFO ==> NOTICE: Starting August 28th, 2025, only a limited subset of images/charts will remain available for free. Backup will be available for some time at the 'Bitnami Legacy' repository. More info at https://github.com/bitnami/containers/issues/83267
redis 21:12:26.82 INFO ==>

redis-1:6379> █
```

Проверка запуска

```
redis-1:6379> ping
PONG
```

```
redis-1:6379> help
redis-cli 8.2.1
To get help about Redis commands type:
  "help @<group>" to get a list of commands in <group>
  "help <command>" for help on <command>
  "help <tab>" to get a list of possible help topics
  "quit" to exit
```

```
To set redis-cli preferences:
  ":set hints" enable online hints
  ":set nohints" disable online hints
Set your preferences in ~/.redisclirc
```

help @string

```
redis-1:6379> help @string

APPEND key value
summary: Appends a string to the value of a key. Creates the key if it doesn't exist.
since: 2.0.0

DECR key
summary: Decrements the integer value of a key by one. Uses 0 as initial value if the key doesn't exist.
since: 1.0.0

DECRBY key decrement
summary: Decrements a number from the integer value of a key. Uses 0 as initial value if the key doesn't exist.
since: 1.0.0

GET key
summary: Returns the string value of a key.
since: 1.0.0

GETDEL key
summary: Returns the string value of a key after deleting the key.
since: 6.2.0

GETEX key [EX seconds|PX milliseconds|EXAT unix-time-seconds|PXAT unix-time-milliseconds|PERSIST]
summary: Returns the string value of a key after setting its expiration time.
since: 6.2.0

GETRANGE key start end
summary: Returns a substring of the string stored at a key.
since: 2.4.0

GETSET key value
summary: Returns the previous string value of a key after setting it to a new value.
since: 1.0.0

INCR key
summary: Increments the integer value of a key by one. Uses 0 as initial value if the key doesn't exist.
since: 1.0.0
```

Использование Redis Commander

Работа с ключами

Сохранение значения по ключу и проверка сохранения данных

```
SET server:name "redis-server"
"OK"
GET server:name
"redis-server"
EXISTS server:name
1
KEYS server*
1) "server:name"
KEYS *
1) "server:name"
```

Операции Get и Set

Работа со значениями, ключами и установка нескольких пар «ключ-значение»

```
SET connections 10
"OK"
GET connections
"10"
SET connections 20
"OK"
GET connections
"20"
SETNX connections 30
0
GET connections
"20"
SETNX newkey 30
1
MSET key1 10 key2 20 key3 30
"OK"
MGET key1 key3
1) "10"
2) "30"
```

Операции инкремента и декремента

Значение как счетчик, работа с командами


```
SET connections 10
"OK"
INCR connections
11
INCRBY connections 10
21
DECR connections
20
DECRBY connections 10
10
DEL connections
1
EXISTS connections
0
INCR connections
1
```

Срок действия (Expiration) и время жизни (TTL)

Существование ключа в течение определенного времени

```
SET resource:lock "Redis Demo"
"OK"
EXPIRE resource:lock 120
1
TTL resource:lock
110
TTL resource:lock
41
TTL resource:lock
-2
EXISTS resource:lock
0
SET resource:lock "Redis Demo 1" EX 120
"OK"
TTL resource:lock
118
SET resource:lock "Redis Demo 2"
"OK"
TTL resource:lock
-1
```

Структуры данных "Список" (List)

```

RPUSH skills "Oracle RDBMS"
1
RPUSH skills "Redis"
2
GET skills
"WRONGTYPE Operation against a key holding the wrong kind of value"
LRANGE skills 0 -1
1) "Oracle RDBMS"
2) "Redis"
LPUSH skills "SQL Server"
3
LRANGE skills 0 -1
1) "SQL Server"
2) "Oracle RDBMS"
3) "Redis"
LRANGE skills 0 -1
1) "SQL Server"
2) "Oracle RDBMS"
3) "Redis"
LRANGE skills 0 1
1) "SQL Server"
2) "Oracle RDBMS"
LRANGE skills 1 2
1) "Oracle RDBMS"
2) "Redis"
LLEN skills
3
LPOP skills
"SQL Server"
RPOP skills
"Redis"
LLEN skills
1
LRANGE skills 0 -1
1) "Oracle RDBMS"

```

Структуры данных "Множество" (Set)

```

SADD nosql:products "Cassandra"
1
SADD nosql:products "Redis"
1
SADD nosql:products "MongoDB"
1
SMEMBERS nosql:products
1) "Cassandra"
2) "Redis"
3) "MongoDB"
SREM nosql:products "MongoDB"
1
SMEMBERS nosql:products
1) "Cassandra"
2) "Redis"
SISMEMBER nosql:products "Cassandra"
1
SISMEMBER nosql:products "MongoDB"
0
SADD rdbms:products "Oracle"
1
SADD rdbms:products "SQL Server"
1
SUNION rdbms:products nosql:products
1) "Oracle"
2) "SQL Server"
3) "Cassandra"
4) "Redis"
SUNIONSTORE database:products rdbms:products nosql:products
4
SMEMBERS database:products
1) "Oracle"
2) "SQL Server"
3) "Cassandra"
4) "Redis"
SADD favorite:products "Cassandra"
1
SADD favorite:products "Oracle"
1
SINTER database:products favorite:products
1) "Cassandra"
2) "Oracle"

```

Структуры данных "Упорядоченное множество" (Sorted Set)

```
ZADD pioneers 1940 "Alan Kay"
1
ZADD pioneers 1906 "Grace Hopper"
1
ZADD pioneers 1953 "Richard Stallman"
1
ZADD pioneers 1965 "Yukihiro Matsumoto"
1
ZADD pioneers 1916 "Claude Shannon"
1
ZADD pioneers 1969 "Linus Torvalds"
1
ZADD pioneers 1957 "Sophie Wilson"
1
ZADD pioneers 1912 "Alan Turing"
1
ZRANGE pioneers 2 4
1) "Claude Shannon"
2) "Alan Kay"
3) "Richard Stallman"
ZRANGE pioneers 2 4 WITHSCORES
1) "Claude Shannon"
2) "1916"
3) "Alan Kay"
4) "1940"
5) "Richard Stallman"
6) "1953"
ZREVRANGE pioneers 0 2
1) "Linus Torvalds"
2) "Yukihiro Matsumoto"
3) "Sophie Wilson"
```

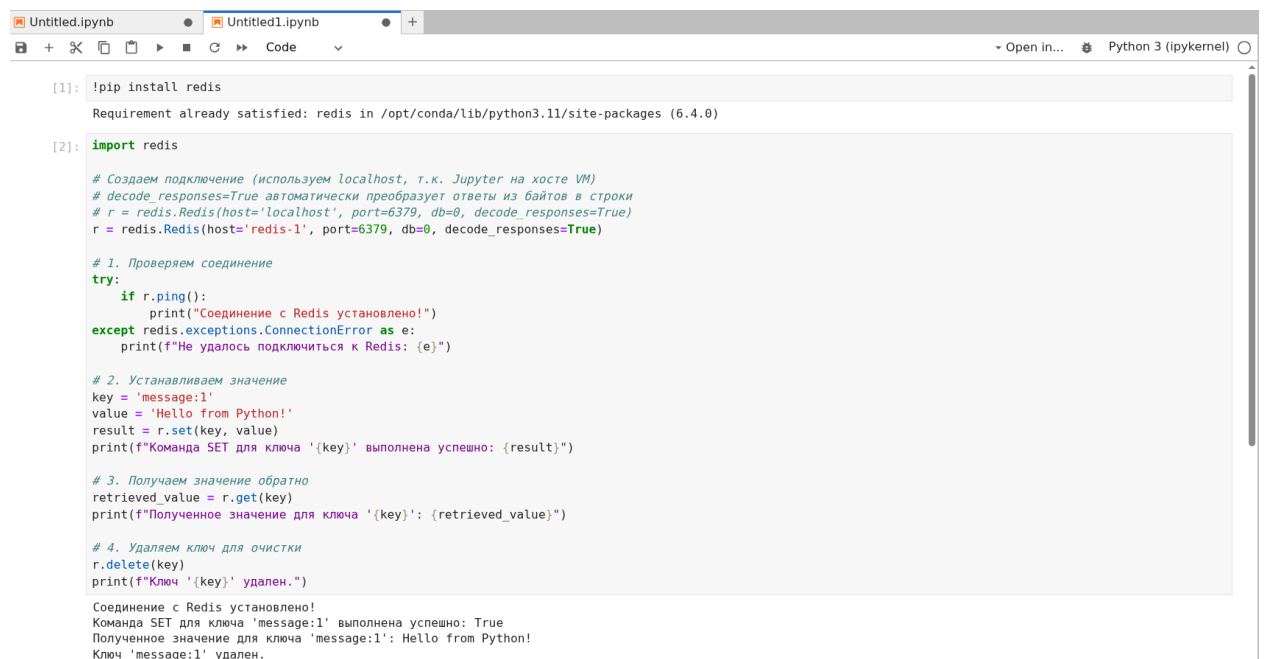
Структуры данных "Хэш" (Hash)

```
HSET user:1000 name "John Smith"
1
HSET user:1000 email "john.smith@example.com"
1
HSET user:1000 password "s3cret"
1
HGETALL user:1000
{
  "name": "John Smith",
  "email": "john.smith@example.com",
  "password": "s3cret"
}
HMSET user:1001 name "Mary Jones" password "hidden" email "mjones@example.com"
"OK"
HGETALL user:1001
{
  "name": "Mary Jones",
  "password": "hidden",
  "email": "mjones@example.com"
}
HGET user:1001 name
"Mary Jones"
HSET user:1000 visits 10
1
HINCRBY user:1000 visits 1
11
HINCRBY user:1000 visits 10
21
HDEL user:1000 visits
1
```

Дополнительные возможности Redis. Геопространственные данные

```
GEOADD cities:russia 37.6176 55.7558 "Moscow"
1
GEOADD cities:russia 30.3351 59.9311 "Saint Petersburg"
1
GEOADD cities:russia 82.9346 55.0084 "Novosibirsk"
1
GEORADIUS cities:russia 37.6176 55.7558 500 km WITHDIST
1) ["Moscow", "0.0002"]
```

Работа с Python



```
Untitled.ipynb • Untitled1.ipynb +
[1]: !pip install redis
Requirement already satisfied: redis in /opt/conda/lib/python3.11/site-packages (6.4.0)

[2]: import redis

# Создаем подключение (используем localhost, т.к. Jupyter на хосте VM)
# decode_responses=True автоматически преобразует ответы из байтов в строки
# r = redis.Redis(host='localhost', port=6379, db=0, decode_responses=True)
r = redis.Redis(host='redis-1', port=6379, db=0, decode_responses=True)

# 1. Проверяем соединение
try:
    if r.ping():
        print("Соединение с Redis установлено!")
except redis.exceptions.ConnectionError as e:
    print(f"Не удалось подключиться к Redis: {e}")

# 2. Устанавливаем значение
key = 'message:1'
value = 'Hello from Python!'
result = r.set(key, value)
print(f"Команда SET для ключа '{key}' выполнена успешно: {result}")

# 3. Получаем значение обратно
retrieved_value = r.get(key)
print(f"Полученное значение для ключа '{key}': {retrieved_value}")

# 4. Удаляем ключ для очистки
r.delete(key)
print(f"Ключ '{key}' удален.")

Соединение с Redis установлено!
Команда SET для ключа 'message:1' выполнена успешно: True
Полученное значение для ключа 'message:1': Hello from Python!
Ключ 'message:1' удален.
```

Задание для индивидуального выполнения на Python

Вариант 6, 16 Лог активности пользователей

Выполнено в файле `redis.ipynb`. Проверка работоспособности

```
2025-09-08 12:11:23,464 - __main__ - INFO - Успешное подключение к Redis на redis-1:6379
2025-09-08 12:11:23,471 - __main__ - INFO - Записано действие пользователя user123: login
2025-09-08 12:11:23,475 - __main__ - INFO - Записано действие пользователя user123: view_page
2025-09-08 12:11:23,477 - __main__ - INFO - Записано действие пользователя user123: purchase
2025-09-08 12:11:23,478 - __main__ - INFO - Получено 0 действий пользователя user123
2025-09-08 12:11:23,479 - __main__ - INFO - Получено 0 действий пользователя user123
2025-09-08 12:11:23,483 - __main__ - INFO - Подсчитана активность пользователя user123: {}
Соединение с Redis установлено!
```

Тестируем запись действий:

Получаем историю действий:

Подсчитываем статистику:

Просматриваем данные в Redis:

Найдено ключей: `[b'user_activity:user123']`

Ключ `b'user_activity:user123'`: 3 записей

```
[9]: # Проверяем данные напрямую
if redis_manager.redis_client.ping():
    print("Прямой доступ к данным:")

    # Все ключи активности
    activity_keys = redis_manager.redis_client.keys("user_activity:*")
    print(f"Ключи активности: {activity_keys}")

    # Данные конкретного пользователя
    user_key = "user_activity:user123"
    if redis_manager.redis_client.exists(user_key):
        all_actions = redis_manager.redis_client.zrange(user_key, 0, -1)
        print(f"\nВсе действия user123:")
        for action in all_actions:
            print(f"  - {action}")

    # Информация о Redis
    info = redis_manager.redis_client.info()
    print(f"\nRedis информация: версия {info['redis_version']], используемая память: {info['used_memory_human']}")
```

Прямой доступ к данным:
Ключи активности: [b'user_activity:user123']

Все действия user123:

- b'{"user_id": "user123", "action_type": "login", "timestamp": "2025-09-08T11:50:52.927776", "details": {"browser": "Chrome", "ip": "192.168.1.100"}}'
- b'{"user_id": "user123", "action_type": "view_page", "timestamp": "2025-09-08T11:50:52.939071", "details": {"page": "/products", "duration": "30s"}}'
- b'{"user_id": "user123", "action_type": "purchase", "timestamp": "2025-09-08T11:50:52.941471", "details": {"item": "laptop", "price": "999.99"}}'

Redis информация: версия 7.2.5, используемая память: 1.56М

Очистка данных

```
[22]: # Очищаем тестовые данные
if redis_manager.redis_client.ping():
    # Удаляем все тестовые ключи
    test_keys = redis_manager.redis_client.keys("user_activity:*")
    if test_keys:
        redis_manager.redis_client.delete(*test_keys)
        print(f"Удалено {len(test_keys)} тестовых ключей")
    else:
        print("Нет тестовых ключей для удаления")
```

Удалено 1 тестовых ключей

Neo4J

Подключение к Cypher Shell

```
mgpu@mgpu-vm:~/Downloads/idb/nosql-workshop/01-environment/docker$ sudo docker exec -ti neo4j-1 ./bin/cypher-shell -u neo4j -p abc123abc123
[sudo] password for mgpu:
Connected to Neo4j using Bolt protocol version 5.4 at neo4j://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j@neo4j> :help

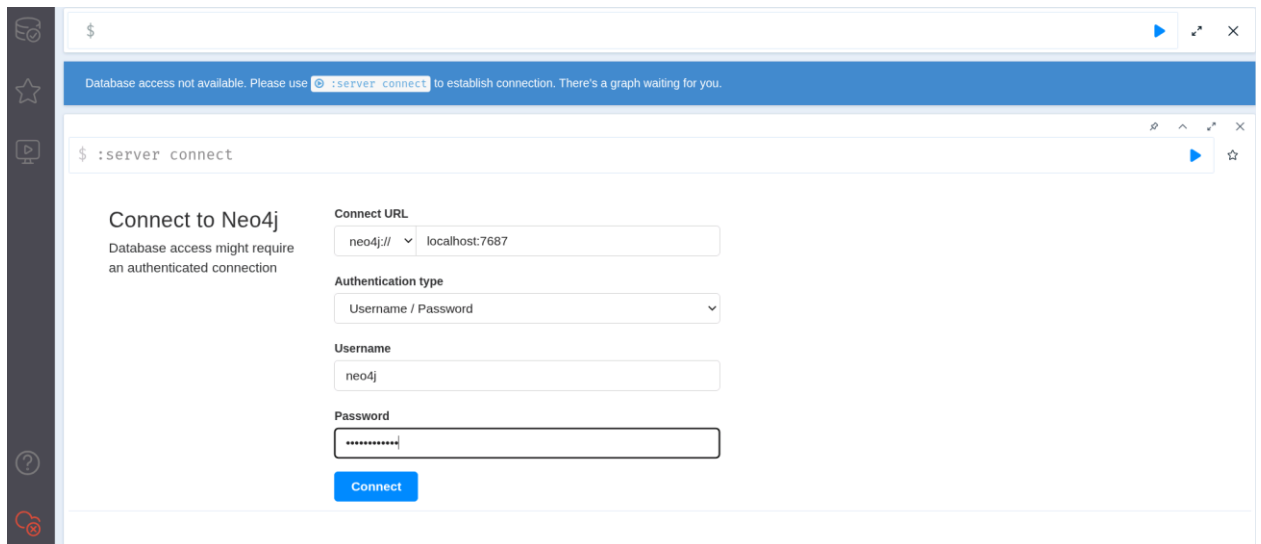
Available commands:
:begin      Open a transaction
:commit     Commit the currently open transaction
:connect    Connects to a database
:disconnect Disconnects from database
:exit       Exit the logger
:help       Show this help message
:history    Statement history
:impersonate Impersonate user
:param      Set the value of a query parameter
:rollback   Rollback the currently open transaction
:source     Executes Cypher statements from a file
:sysinfo    Neo4j system information
:use        Set the active database

For help on a specific command type:
:help command

Keyboard shortcuts:
Up and down arrows to access statement history.
Tab for autocompletion of commands, hit twice to select suggestion from list using arrow keys.

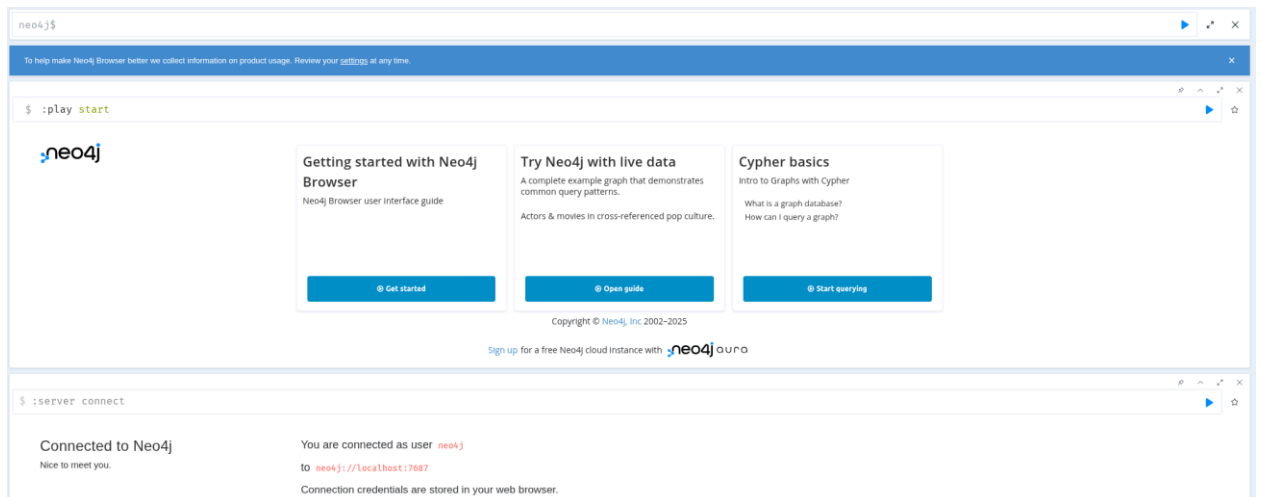
For help on cypher please visit:
https://neo4j.com/docs/cypher-manual/current/
```

Подключение через Neo4J Browser



The image shows the Neo4j Browser web interface. At the top, a blue banner states "Database access not available. Please use :server connect to establish connection. There's a graph waiting for you." Below this, the command ":server connect" is entered in the terminal. The main area is titled "Connect to Neo4j" and includes a note: "Database access might require an authenticated connection". The connection form contains the following fields:

- Connect URL:** A dropdown menu showing "neo4j://" and a text input field containing "localhost:7687".
- Authentication type:** A dropdown menu set to "Username / Password".
- Username:** A text input field containing "neo4j".
- Password:** A password input field with masked characters "*****".
- Connect:** A blue button to initiate the connection.



The image shows the Neo4j Browser main dashboard after a successful connection. The terminal at the top shows the command ":play start". The dashboard features the Neo4j logo and three main sections:

- Getting started with Neo4j Browser:** A section titled "Neo4j Browser user interface guide" with a "Get started" button.
- Try Neo4j with live data:** A section titled "A complete example graph that demonstrates common query patterns" with a sub-header "Actors & movies in cross-referenced pop culture" and an "Open guide" button.
- Cypher basics:** A section titled "Intro to Graphs with Cypher" with sub-headers "What is a graph database?" and "How can I query a graph?", and a "Start querying" button.

At the bottom, a status bar indicates "Connected to Neo4j" and "You are connected as user neo4j to neo4j://localhost:7687". It also notes "Connection credentials are stored in your web browser." and includes a "Sign up for a free Neo4j cloud Instance with neo4j aura" link.

Граф фильмов

```
neo4j$ :play movie graph
```

Movie Graph

Pop-cultural connections between actors and movies

The Movie Graph is a mini graph application containing actors and directors that are related through the movies they've collaborated on.

This guide will show you how to:

- Create: Insert movie data into the graph
- Find: retrieve individual movies and actors
- Query: discover related actors and directors
- Solve: the Bacon Path

WARNING: This guide will modify the data in the currently active database.

Загрузка графа фильмов

```
neo4j$ :play movie graph
```

The Movie Graph

Create

To the right is a giant code block containing a single Cypher query statement composed of multiple CREATE clauses. This will create the movie graph.

Click on the code block
Notice it gets copied to the editor above
Click the editor's play button to execute
Wait for the query to finish
WARNING: This adds data to the current database, each time it is run!

[help](#) [cypher](#) [CREATE](#)

```
@ CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real World'})
CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})
CREATE (Carrie:Person {name:'Carrie-Anne Moss', born:1967})
CREATE (Laurence:Person {name:'Laurence Fishburne', born:1961})
CREATE (Hugo:Person {name:'Hugo Weaving', born:1960})
CREATE (LillyW:Person {name:'Lilly Wachowski', born:1967})
CREATE (LanaW:Person {name:'Lana Wachowski', born:1965})
CREATE (JoelS:Person {name:'Joel Silver', born:1952})
CREATE
(Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrix),
(Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrix),
```

Database Information

Use database

neo4j

Node labels

Movie (19) Person (9)

Relationship types

ACTED_IN DIRECTED

Property keys

born name rating released roles summary tagline title

Connected as

Username: neo4j Roles: - Disconnect: server disconnect

DBMS

Cluster role: primary Version: 5.19.0 Edition: Community Name: neo4j Databases: dbx Information: system Query List: queries

```
neo4j$ CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real World'}) CREATE
```

Overview

Node labels

Movie (19) Person (9)

Relationship types

ACTED_IN (10) DIRECTED (18)

Displaying 19 nodes, 0 relationships.

```
neo4j$ :help create
```

CREATE

Insert graph data

The **CREATE** clause is used to create data by specifying named nodes and relationships with inline properties.

Use **MATCH** clauses for reading data, and **CREATE** or **MERGE** for writing data.

Reference [CREATE manual page](#)

Related [help SET](#) [help MERGE](#) [help Cypher](#)

Примеры запросов и их выполнение

Поиск актёра по имени "Том Хэнкс"

```
neo4j$ MATCH (tom {name: "Tom Hanks"}) RETURN tom
```

Node properties

Person

<elementid> 4:ae181c8c-9161-4092-96ff-7f15a426cf53:71

<id> 71

born 1956

name Tom Hanks

Поиск фильма с названием "Cloud Atlas" и все связанные узлы

neo4j\$ MATCH (cloudAtlas {title: "Cloud Atlas"}) RETURN cloudAtlas

Graph

Table

Text

Code

Overview

Node labels

* (1)

Movie (1)

Person (10)

Relationship types

* (10)

ACTED_IN (4)

DIRECTED (3)

WROTE (1)

PRODUCED (1)

REVIEWED (1)

Displaying 1 nodes, 0 relationships.

Другие запросы

neo4j\$ MATCH (people:Person) RETURN people.name LIMIT 10

Table

Text

Code

people.name
"Keanu Reeves"
"Carrie-Anne Moss"
"Laurence Fishburne"
"Hugo Weaving"
"Lilly Wachowski"
"Lana Wachowski"

neo4j\$ MATCH (nineties:Movie) WHERE nineties.released >= 1990 AND nineties.released < 2000 RETURN nineties.title

Table

Text

Code

nineties.title
"The Matrix"
"The Devil's Advocate"
"A Few Good Men"
"As Good as It Gets"
"What Dreams May Come"
"Snow Falling on Cedars"

neo4j\$ MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors) RETURN coActors.name

Table

Text

Code

coActors.name
"Ed Harris"
"Gary Sinise"
"Kevin Bacon"
"Bill Paxton"
"Parker Posey"

neo4j\$ MATCH p=shortestPath((bacon:Person {name:"Kevin Bacon"})-[*]-(meg:Person {name:"Meg Ryan"})) RETURN p

Graph

Table

Text

Warn

Code

Overview

Node labels

* (5)

Person (3)

Movie (2)

Relationship types

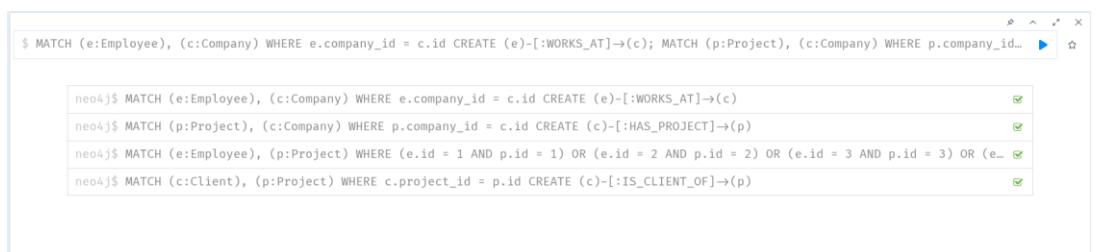
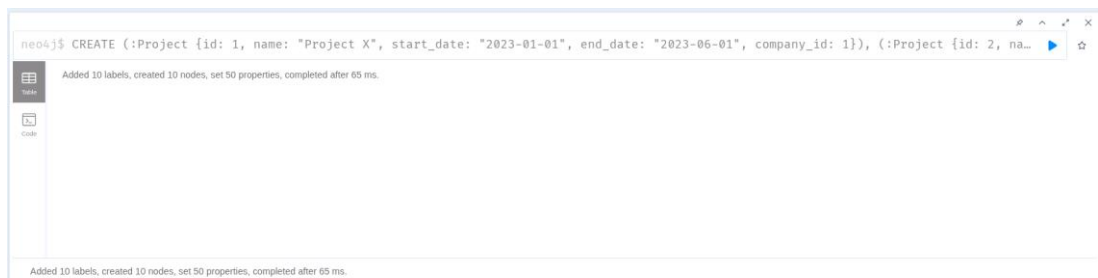
* (4)

ACTED_IN (4)

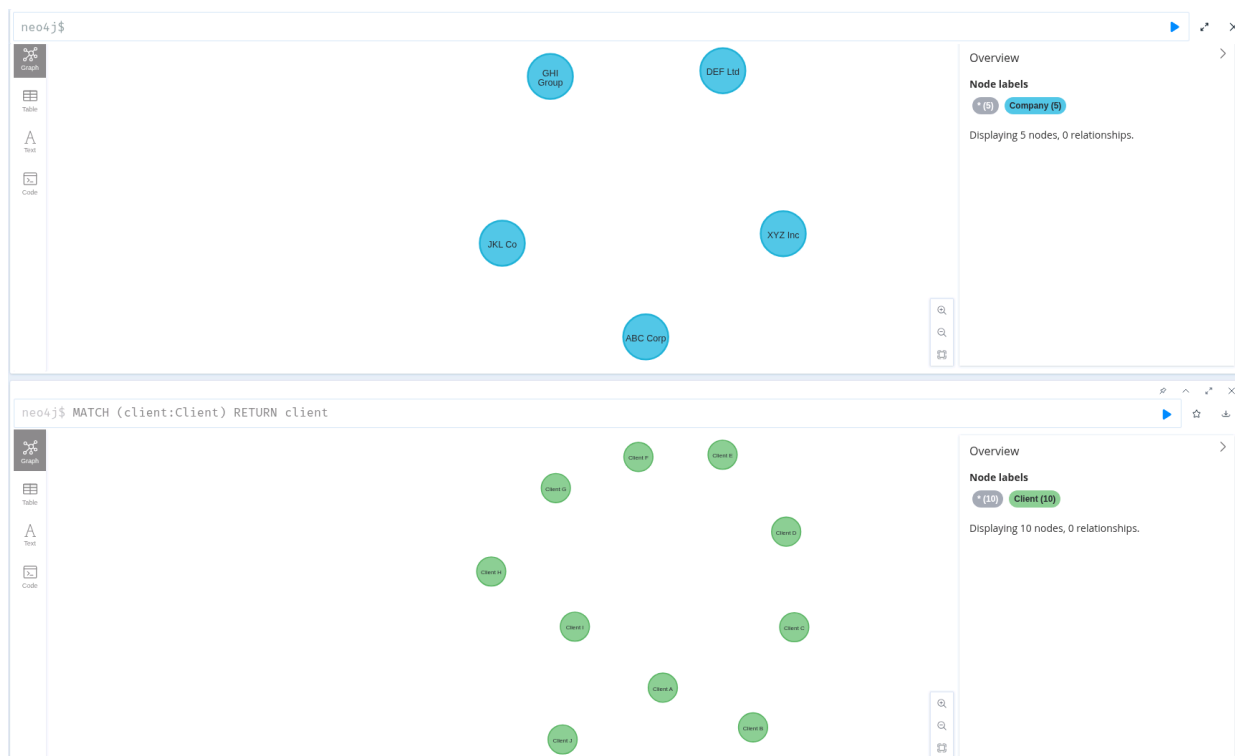
Displaying 5 nodes, 4 relationships.

Дополнительное задание Neo4j

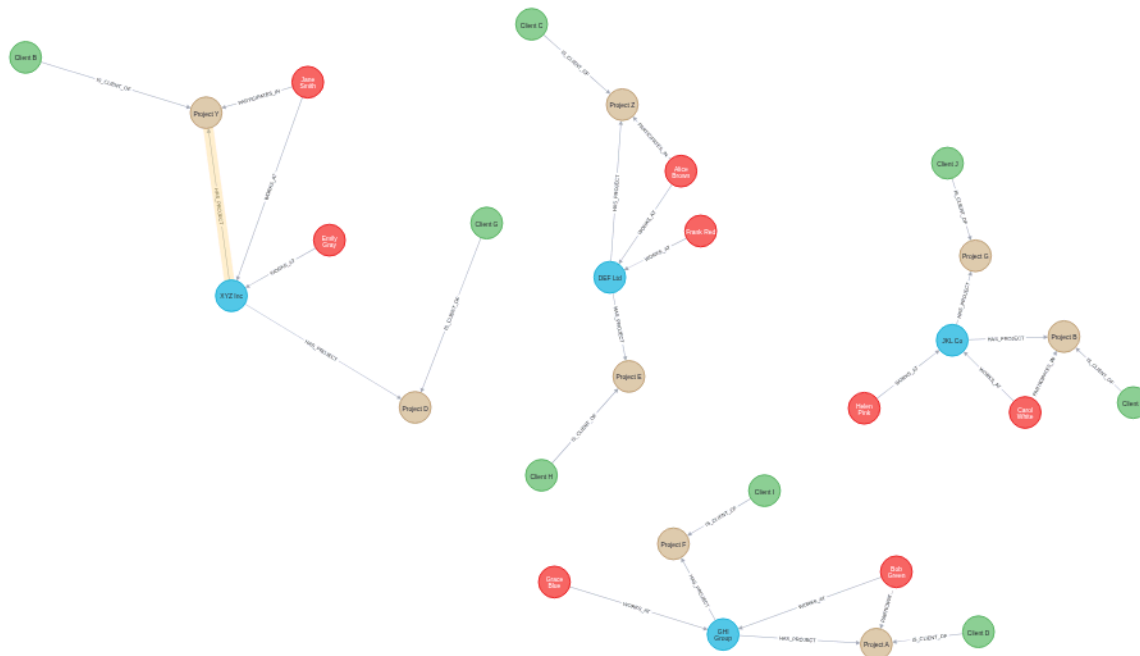
Создание узлов и отношений:



Примеры узлов



Вид графа



Индивидуальный вариант 6.

Задание 1:

Найдите все компании из здравоохранения.

Решение:

Для графического представления:

```
MATCH (c:Company {industry: "Healthcare"})
```

```
RETURN c
```

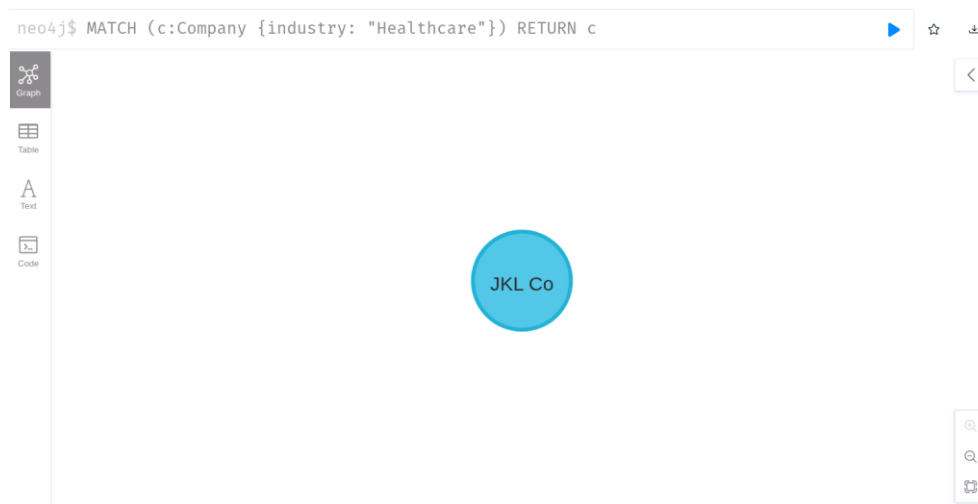
Для табличного представления:

```
MATCH (c:Company {industry: "Healthcare"})
```

```
RETURN c.id AS ID, c.name AS Company, c.industry AS Industry
```

Результат:

Графическое представление:



Табличное представление:

The image shows the Neo4j Cypher query interface with the same query as above. The interface is set to 'Table' view. The main area displays a table with the following data:

ID	Company	Industry
5	"JKL Co"	"Healthcare"

Задание 2:

Найдите всех дизайнеров в базе данных.

Решение:

Для графического представления:

```
MATCH (e:Employee {position: "Designer"})
```

```
RETURN e
```

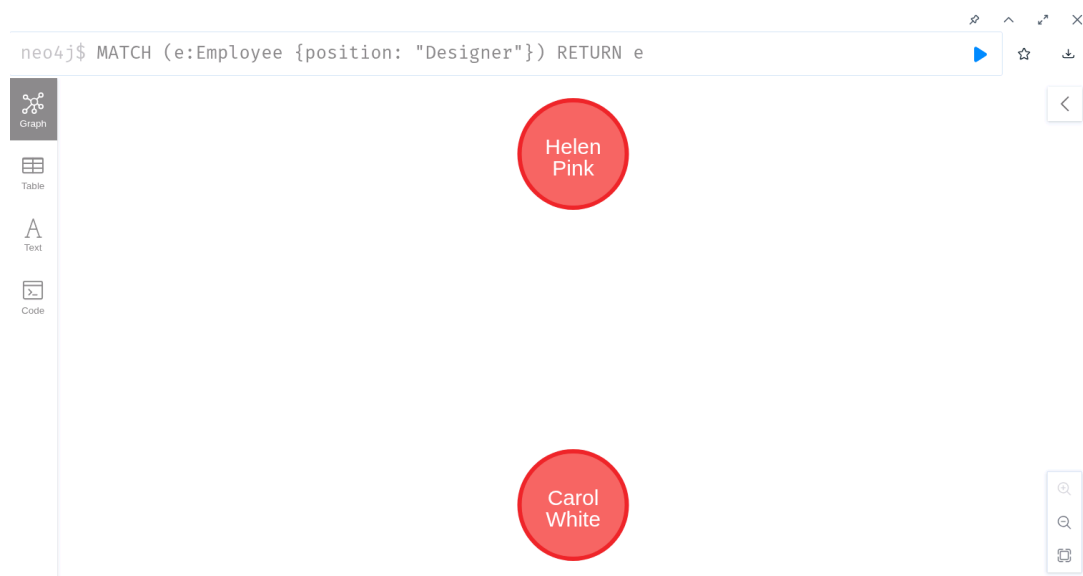
Для табличного представления:

```
MATCH (e:Employee {position: "Designer"})
```

```
RETURN e.id AS ID, e.name AS Name, e.position AS Position
```

Результат:

Графическое представление:



Табличное представление:

The screenshot shows the Neo4j Cypher query editor with the query: `neo4j$ MATCH (e:Employee {position: "Designer"}) RETURN e.id AS ID, e.name AS Name, ...`. The interface includes a sidebar with icons for Table, Text, and Code. The main area displays a table visualization with the following data:

	ID	Name	Position
1	5	"Carol White"	"Designer"
2	10	"Helen Pink"	"Designer"

Задание 3:

Найдите все проекты, завершившиеся в 2024 году.

Решение:

Для графического представления:

```
MATCH (p:Project)
```

```
WHERE date(p.end_date).year = 2024
```

```
RETURN p
```

Для табличного представления:

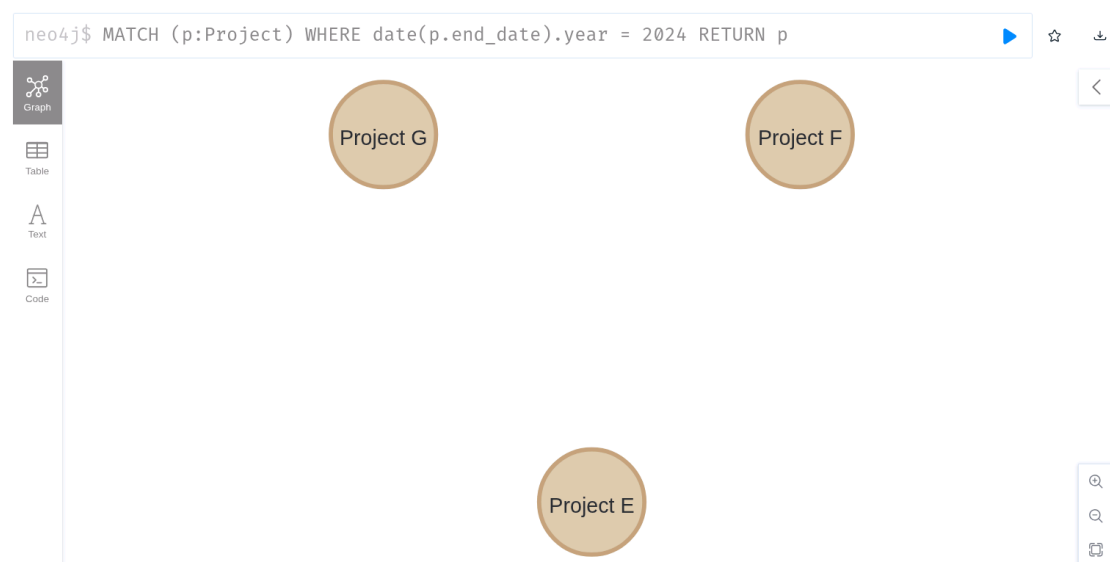
```
MATCH (p:Project)
```

```
WHERE date(p.end_date).year = 2024
```

```
RETURN p.id AS ID, p.name AS Project, p.end_date AS EndDate
```

Результат:

Графическое представление:



Табличное представление:

neo4j\$ MATCH (p:Project) WHERE date(p.end_date).year = 2024 RETURN p.id AS ID, p.name AS Project, p.end_date AS EndDate

	ID	Project	EndDate
1	8	"Project E"	"2024-01-01"
2	9	"Project F"	"2024-02-01"
3	10	"Project G"	"2024-03-01"

Started streaming 3 records after 26 ms and completed after 33 ms.

Задание 4:

Найдите всех клиентов проекта "Project A".

Решение:

Для графического представления:

```
MATCH (p:Project {name: "Project A"})<-[:IS_CLIENT_OF]-(client:Client)
```

```
RETURN p, client
```

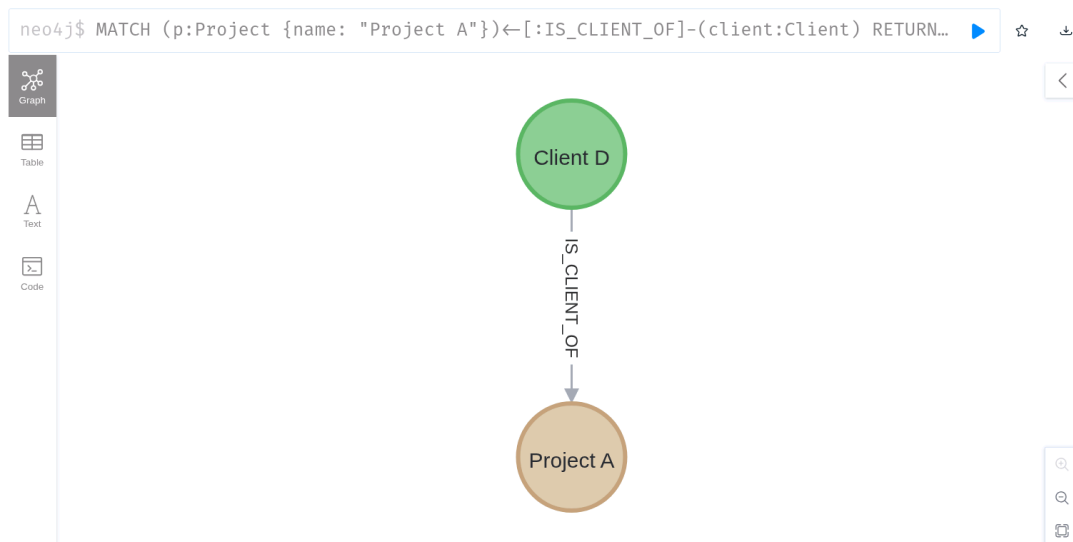
Для табличного представления:

```
MATCH (p:Project {name: "Project A"})<-[:IS_CLIENT_OF]-(client:Client)
```

```
RETURN client.id AS ID, client.name AS Client, p.name AS Project
```

Результат:

Графическое представление:



Табличное представление:

```
neo4j$ MATCH (p:Project {name: "Project A"})←[:IS_CLIENT_OF]-(client:Client) RETURN...
```

	ID	Client	Project
1	4	"Client D"	"Project A"

Задание 5:

Найдите все проекты, в которых участвует сотрудник "Bob Green".

Решение:

Для графического представления:

```
MATCH (e:Employee {name: "Bob Green"})-[:PARTICIPATES_IN]->(p:Project)
```

```
RETURN e, p
```

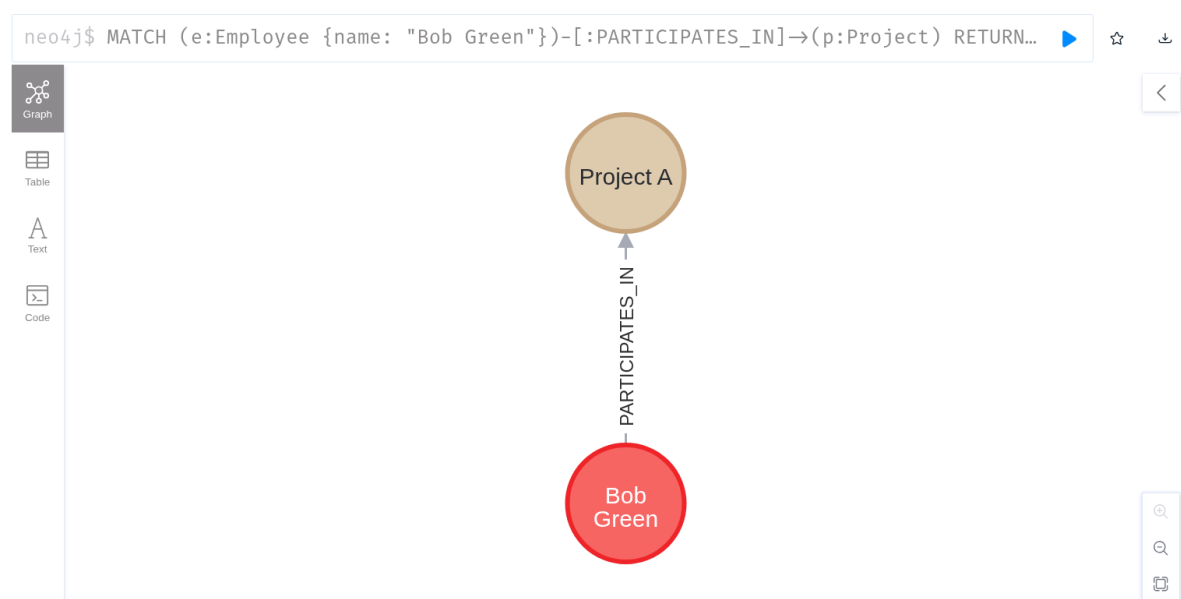
Для табличного представления:

```
MATCH (e:Employee {name: "Bob Green"})-[:PARTICIPATES_IN]->(p:Project)
```

```
RETURN e.name AS Employee, p.name AS Project, p.id AS ProjectID
```

Результат:

Графическое представление:



Табличное представление:

neo4j\$ MATCH (e:Employee {name: "Bob Green"})-[:PARTICIPATES_IN]→(p:Project) RETURN...   

 Table	Employee	Project	ProjectID
 Text	1 "Bob Green"	"Project A"	4
 Code			

Индивидуальное задание для самостоятельной работы

Вариант 6.

Задание 1 (MongoDB). Найти все фильмы жанров "Family" или "Mystery" (\$in) и добавить им в массив genres новый тег "classic" (\$addToSet).

Найти все фильмы жанров:

```
> db.movies.find({ "genres": { $in: ['Family', 'Mystery']} })
< {
  _id: ObjectId('68bdee847c6e678c2a7ca7fb'),
  id: '0038650',
  title: "It's a Wonderful Life",
  genres: [
    'Drama',
    'Family',
    'Fantasy'
  ],
  year: 1946,
  rating: 8.6,
  rank: 25
}
{
  _id: ObjectId('68bdee847c6e678c2a7ca7fd'),
  id: '0245429',
  title: 'Spirited Away',
  genres: [
    'Animation',
    'Adventure',
    'Family',
    'Fantasy',
    'Mystery'
  ],
  year: 2001,
  rating: 8.5,
  rank: 27
}
```

Добавление тега (одной командой):

```
> db.movies.updateMany({ "genres": { $in: ["Family", "Mystery"] } }, { $addToSet: { "genres": "classic" } })
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 10,
  modifiedCount: 10,
  upsertedCount: 0
}
filmbd>
```

Проверка результата:

```
> db.movies.find({ "genres": { $in: ['Family', 'Mystery']} })
< {
  _id: ObjectId('68bdee847c6e678c2a7ca7fb'),
  id: '0038650',
  title: "It's a Wonderful Life",
  genres: [
    'Drama',
    'Family',
    'Fantasy',
    'classic'
  ],
  year: 1946,
  rating: 8.6,
  rank: 25
}
{
  _id: ObjectId('68bdee847c6e678c2a7ca7fd'),
  id: '0245429',
  title: 'Spirited Away',
  genres: [
    'Animation',
    'Adventure',
    'Family',
    'Fantasy',
    'Mystery',
    'classic'
  ],
  year: 2001,
```

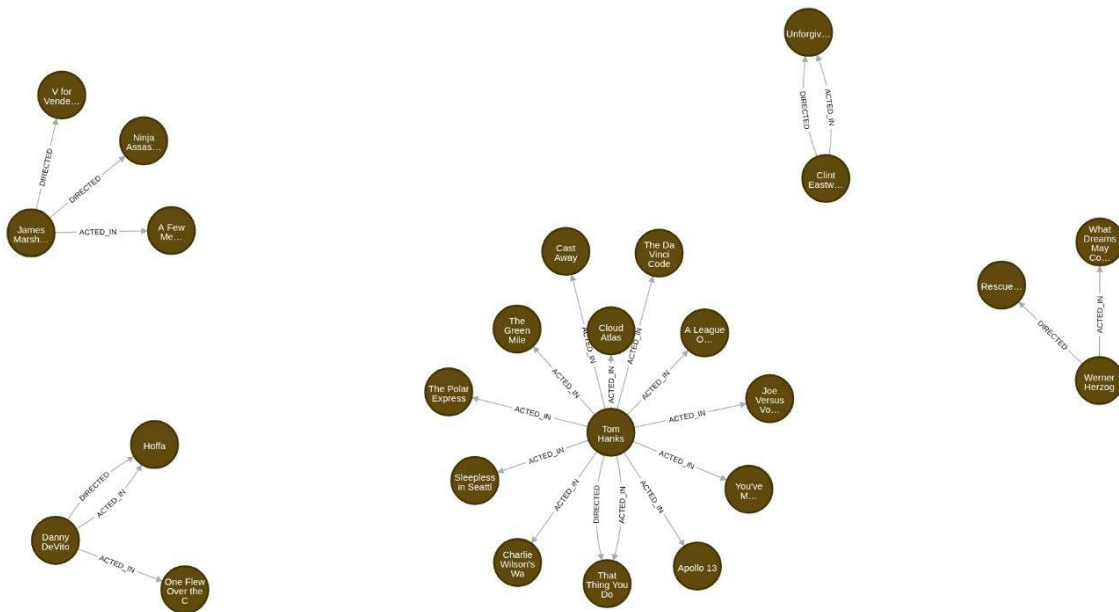
Задание 2 (Neo4j). Найти всех режиссеров, которые также снимались в фильмах в качестве актеров.

Режиссеры-актеры и их связи с фильмами:

MATCH (p:Person)-[:DIRECTED]->(dm:Movie)

MATCH (p)-[:ACTED_IN]->(am:Movie)

RETURN p, dm, am



Табличное представление:

MATCH (p:Person)-[:DIRECTED]->(:Movie)

WHERE EXISTS {

(p)-[:ACTED_IN]->(:Movie)

}

RETURN DISTINCT p.name AS DirectorActor

ORDER BY DirectorActor

neo4j\$ MATCH (p:Person)-[:DIRECTED]→(:Movie) WHERE EXISTS { (p)-[:ACTED_IN]→(:Movi...

☆ ⬇

Table	DirectorActor
1	"Clint Eastwood"
2	"Danny DeVito"
3	"James Marshall"
4	"Tom Hanks"
5	"Werner Herzog"

Started streaming 5 records after 270 ms and completed after 290 ms.

Задание 3 (Redis). Смоделировать корзину покупок: для пользователя user:205 в хэш cart:205 добавить 2 товара (product_id и quantity). Увеличить количество одного из товаров на 2 (HINCRBY).

Добавление товаров в корзину:

Первый товар (ID: 123, количество: 1) - HSET cart:205 product_123 1

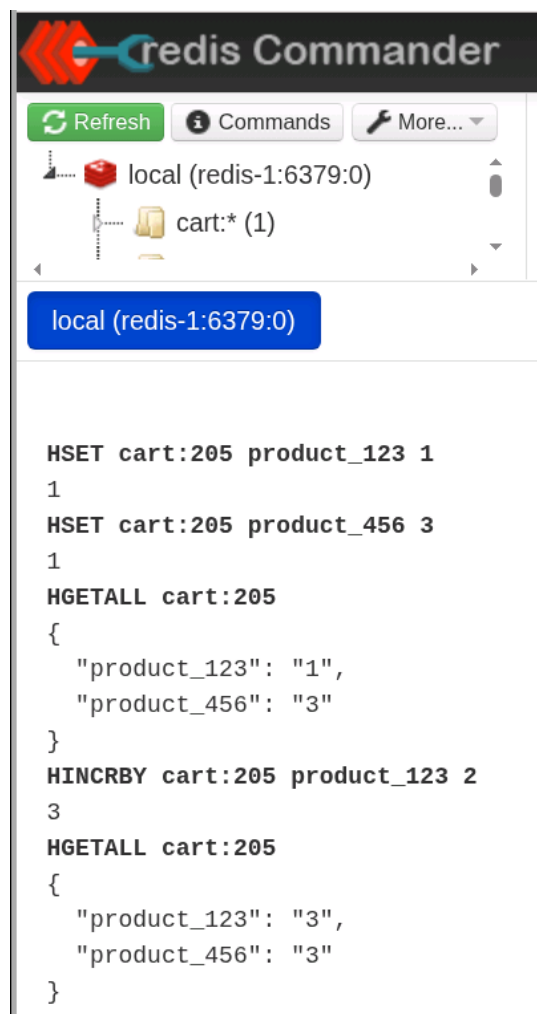
Второй товар (ID: 456, количество: 3) - HSET cart:205 product_456 3

Увеличение количества товара на 2:

HINCRBY cart:205 product_123 2

Проверка результата:

HGETALL cart:205



Добавление нескольких товаров сразу и изменение после:

```
HMSET cart:205 product_1 1 product_2 3
"OK"
```

```
HGETALL cart:205
```

```
{
  "product_123": "3",
  "product_456": "3",
  "product_1": "1",
  "product_2": "3"
}
```

```
HINCRBY cart:205 product_1 2
```

```
3
```

```
HGETALL cart:205
```

```
{
  "product_123": "3",
  "product_456": "3",
  "product_1": "3",
  "product_2": "3"
}
```

Шаг 4. Программное взаимодействие с базами данных (Python)

4.1. Работа с MongoDB через pymongo

```
from pymongo import MongoClient

from bson import ObjectId

# 1. Задайте параметры подключения

mongo_uri = "mongodb://admin:pass@mongo-
1:27017/admin?authSource=admin"

db_name = "test"

collection_name = "movies"

try:

    # 2. Подключение к MongoDB

    client = MongoClient(mongo_uri)

    client.admin.command('ping')

    print("Подключение к MongoDB установлено успешно!")

    # 3. Выбор базы данных и коллекции

    db = client[db_name]

    collection = db[collection_name]

    # Удаляем проблемный индекс перед началом работы

    try:

        collection.drop_index("id_1")

        print("Проблемный индекс 'id_1' удален")

    except Exception as e:

        print(f"Индекс не найден или не может быть удален: {e}")

    # Очистка коллекции перед началом работы

    collection.delete_many({})
```

```
# 4. Вставка данных (Create) - убираем поле 'id' если оно
есть

test_data = [

    {"lab_name": "Lab 1", "subject": "Physics", "score":
85},

    {"lab_name": "Lab 2", "subject": "Chemistry", "score":
90},

    {"lab_name": "Lab 3", "subject": "Biology", "score":
88},

]

result = collection.insert_many(test_data)

print(f"\nВставлено документов: {len(result.inserted_ids)}")

# 5. Чтение данных (Read)

print("\nСодержимое коллекции:")

for doc in collection.find():

    print(doc)

# 6. Обновление данных (Update)

collection.update_one({"subject": "Physics"}, {"$set":
{"score": 95}})

print("\nДокумент после обновления:")

print(collection.find_one({"subject": "Physics"}))

# 7. Удаление данных (Delete)

collection.delete_one({"subject": "Chemistry"})

print(f"\nКоличество документов после удаления:
{collection.count_documents({})}")

# 8. Удаление коллекции для очистки

db.drop_collection(collection_name)
```



```

        print(f"\nКоллекция '{collection_name}' удалена.")

except Exception as e:

    print(f"Ошибка: {e}")

finally:

    # 9. Заккрытие подключения

    if 'client' in locals() and client:

        client.close()

        print("\nПодключение к MongoDB закрыто.")

```

Результат выполнения:

Подключение к MongoDB установлено успешно!
Проблемный индекс 'id_1' удален

Вставлено документов: 3

Содержимое коллекции:

```

{'_id': ObjectId('68befcaab540de80af8108b7'), 'lab_name': 'Lab 1', 'subject': 'Physics', 'score': 85}
{'_id': ObjectId('68befcaab540de80af8108b8'), 'lab_name': 'Lab 2', 'subject': 'Chemistry', 'score': 90}
{'_id': ObjectId('68befcaab540de80af8108b9'), 'lab_name': 'Lab 3', 'subject': 'Biology', 'score': 88}

```

Документ после обновления:

```

{'_id': ObjectId('68befcaab540de80af8108b7'), 'lab_name': 'Lab 1', 'subject': 'Physics', 'score': 95}

```

Количество документов после удаления: 2

Коллекция 'movies' удалена.

Подключение к MongoDB закрыто.

4.2. Работа с Redis через redis-py

```

import redis

try:

    # 1. Подключение к Redis (корректные параметры для VM)

    r = redis.Redis(host='redis-1', port=6379, db=0,
decode_responses=True)

    r.ping()

    print("Соединение с Redis успешно установлено.")

    # 2. Работа со строками (String)

    r.set('user:1:name', 'Alice')

```

```
user_name = r.get('user:1:name')

print(f"\nСтрока: user:1:name = {user_name}")

# 3. Работа с хэшами (Hash) - для хранения объектов

r.hset('user:2', mapping={'name': 'Bob', 'email':
'bob@example.com'})

user_info = r.hgetall('user:2')

print(f"Хэш: user:2 = {user_info}")

# 4. Работа со списками (List) - для очередей, логов

r.lpush('tasks', 'task1', 'task2', 'task3')

tasks = r.lrange('tasks', 0, -1)

print(f"Список: tasks = {tasks}")

# 5. Работа с множествами (Set) - для уникальных элементов

r.sadd('online_users', 'user_a', 'user_b', 'user_c',
'user_a')

online_count = r.scard('online_users')

print(f"Множество: {r.smembers('online_users')}, количество
онлайн: {online_count}")

# 6. Очистка созданных ключей

keys_to_delete = ['user:1:name', 'user:2', 'tasks',
'online_users']

r.delete(*keys_to_delete)

print(f"\nТестовые ключи удалены.")

except redis.ConnectionError as e:

    print(f"Ошибка подключения к Redis: {e}")
```

Результат выполнения:

Соединение с Redis успешно установлено.

Строка: user:1:name = Alice

Хэш: user:2 = {'name': 'Bob', 'email': 'bob@example.com'}

Список: tasks = ['task3', 'task2', 'task1']

Множество: {'user_a', 'user_c', 'user_b'}, количество онлайн: 3

Тестовые ключи удалены.

4.3. Работа с Neo4j через neo4j-driver

```
from neo4j import GraphDatabase
```

```
# 1. Параметры подключения
```

```
uri = "bolt://localhost:7687" # Правильный URI для подключения  
к Neo4j
```

```
user = "neo4j"
```

```
password = "abc123abc123" # Пароль, установленный при первом  
запуске Neo4j Browser
```

```
# 2. Функция для выполнения запроса (Transaction Function)
```

```
def get_movies(tx, movie_limit): # 'tx' - объект транзакции
```

```
    # Cypher-запрос для получения названий фильмов
```

```
    # Используется параметризованный запрос ($limit) для  
    безопасности
```

```
    query = """
```

```
    MATCH (m:Movie)
```

```
    RETURN m.title AS title
```

```
    LIMIT $limit
```

```
    """
```

```
    # Выполнение запроса с передачей параметра
```

```
    result = tx.run(query, limit=movie_limit)
```

```

        # 3. Обработка результата: извлечение значения 'title' из
каждой записи

        return [record["title"] for record in result]

try:

    with GraphDatabase.driver(uri, auth=(user, password)) as
driver:

        with driver.session() as session:

            # Запрос выполняется в одну строку с помощью lambda-
функции

            movies = session.execute_read(

                lambda tx, limit: tx.run("MATCH (m:Movie) RETURN
m.title LIMIT $limit", limit=limit).value("title"),

                5 # Аргумент для lambda-функции

            )

            print("Фильмы:", movies)

except Exception as e:

    print(f"Ошибка: {e}")

```

Результат выполнения:

Выводы

В ходе данной лабораторной работы были изучены и практически применены три различных типа NoSQL баз данных: документо-ориентированная MongoDB, графовая Neo4j и хранилище типа «ключ-значение» Redis.

Освоены ключевые операции: запросы в MongoDB, работа с языком Cypher в Neo4j для анализа связей и манипуляции различными структурами данных в Redis.

Основные трудности включали первоначальную настройку окружения и освоение нового синтаксиса запросов Cypher, которые были решены через изучение документации и примеров.

Приобретены практические навыки администрирования (запуск СУБД в Docker), взаимодействия с базами через графические клиенты и Python-библиотеки, а также основы моделирования данных для разных нереляционных подходов.