Лабораторная работа 2.1. Изучение методов хранения данных на основе NoSQL

Цель работы: изучение и практическое применение трех различных типов NoSQL баз данных: документо-ориентированной (MongoDB), графовой (Neo4j) и ключ-значение (Redis). Студенты должны 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.

Оборудование и программное обеспечение

- Операционная система Ubuntu. Скачать по ссылке (https://disk.yandex.ru/d/91q8DWFBV7ILMQ).
- Установленные пакеты для работы с NoSQL базами данных: MongoDB, Redis, Neo4j.
- Язык программирования Python (с библиотеками pymongo, redis, neo4j).
- CSV файлы с данными.

Краткая теоретическая справка

NoSQL — это подход к проектированию баз данных, которые не являются реляционными и могут использовать различные модели данных. Они оптимизированы для конкретных сценариев использования и обеспечивают высокую масштабируемость и гибкость.

1. MongoDB: документо-ориентированная СУБД

MongoDB хранит данные в гибких, JSON-подобных документах. Это означает, что поля могут варьироваться от документа к документу и структура данных может быть изменена со временем.

- **Основные концепции:**
  - **База данных (Database):** контейнер для коллекций.
  - **Коллекция (Collection):** аналог таблицы в реляционных СУБД. Хранит группу связанных документов, но не требует от них одинаковой схемы.
  - **Документ (Document):** аналог строки (записи) в реляционных СУБД. Представляет собой структуру данных в формате BSON (бинарный JSON).
  - **Поле (Field):** аналог столбца, пара ключ-значение в документе.
- **Ключевая особенность:** гибкая схема, позволяющая хранить в одной коллекции документы с разным набором полей.

2. Neo4j: графовая СУБД

Neo4j — это графовая база данных, разработанная для хранения и обработки связанных данных. Она идеально подходит для анализа сложных взаимосвязей,

таких как социальные сети, рекомендательные системы и управление зависимостями.

- **Основные компоненты графовой модели:**
  - **Узлы (Nodes):** представляют сущности (например, "Человек", "Фильм"). Аналогичны записям в реляционной таблице.
  - **Отношения (Relationships):** представляют связи между узлами (например, ACTED_IN, DIRECTED). Отношения всегда имеют направление и тип.
  - **Свойства (Properties):** пары ключ-значение, которые хранятся внутри узлов и отношений (например, name: 'Tom Hanks').
- **Язык запросов:** Cypher — декларативный язык для запросов к графу.

3. Redis: хранилище типа "ключ-значение"

Redis (Remote Dictionary Server) — это высокопроизводительное хранилище данных в памяти, работающее по принципу "ключ-значение". Оно часто используется для кэширования, управления сессиями, очередей сообщений и в качестве брокера сообщений.

- **Основные структуры данных:**
  - **Строки (Strings):** простейший тип, где одному ключу соответствует одно строковое значение.
  - **Списки (Lists):** последовательности строк, упорядоченные по порядку вставки.
  - **Множества (Sets):** неупорядоченные коллекции уникальных строк.
  - **Хэши (Hashes):** структуры для хранения объектов, состоящие из полей и их значений.
  - **Упорядоченные множества (Sorted Sets):** множества, где каждый элемент связан с числовым значением (оценкой), которое используется для сортировки.

Порядок выполнения работы

1. Ознакомьтесь с инструкциями по настройке окружения, загрузке данных и базовым операциям для каждой СУБД в официальной директории курса на GitHub: https://github.com/BosenkoTM/nosql-workshop/tree/main/01-environment
2. Выберите вариант задания из таблицы ниже в соответствии с вашим номером в списке группы.
3. Выполните все три комплексных задания для вашего варианта, документируя каждый шаг (код запроса/скрипта и результат его выполнения).
4. Подготовьте и оформите отчет в соответствии с требованиями, указанными в разделе "Оформление отчета".

Шаг 1. Подготовка рабочего окружения на виртуальной машине

Все необходимые базы данных запускаются как Docker-контейнеры с помощью одного файла конфигурации.

1. Запустите виртуальную машину и войдите в систему.
2. Откройте **Терминал** (Terminal).
3. Перейдите в директорию с файлами для запуска Docker:
   ```
   cd /home/mgpu/Downloads/ibd/nosql-workshop/01-environment/docker
   ```
4. Запустите все контейнеры с базами данных в фоновом режиме:
   ```
   sudo docker compose up -d
   ```
   После выполнения этой команды все три базы данных (MongoDB, Redis, Neo4j) будут запущены и готовы к работе.
5. Для остановки контейнеров используйте команду sudo docker compose stop. Для полного удаления — sudo docker compose down.

Шаг 2. Доступ к инструментам управления

1. **MongoDB.** Запустите **MongoDB Compass** из меню приложений. Создайте новое подключение, используя строку URI (остальные поля заполнятся автоматически):
   ```
   mongodb://mongoroot:mongopass@localhost:27017/
   ```
2. **Redis.** Откройте браузер Google Chrome и перейдите по адресу http://localhost:8082. Вы увидите веб-интерфейс **Redis Commander**.
3. **Neo4j.** Откройте браузер Google Chrome и перейдите по адресу http://localhost:7474. Это **Neo4j Browser**. При первом входе используйте:
   - **Login:** neo4j
   - **Password:** password (при первом входе система попросит сменить пароль, можете установить любой).
4. **Python/Jupyter.** Запустите **JupyterLab** из меню приложений для выполнения заданий с использованием Python.

Шаг 3. Выполнение заданий

1. Ознакомьтесь с инструкциями по загрузке данных и базовым операциям в официальной директории курса на GitHub (рекомендуется для углубленного понимания):

- MongoDB Workshop (https://github.com/BosenkoTM/nosql-workshop/tree/main/04-working-with-mongodb)
- Redis Workshop (https://github.com/BosenkoTM/nosql-workshop/tree/main/02-working-with-redis)
- Neo4j Workshop(https://github.com/BosenkoTM/nosql-workshop/tree/main/06-working-with-neo4j)

2. Выберите **один вариант** из таблицы ниже в соответствии с вашим номером в списке группы.
3. Выполните все три комплексных задания для вашего варианта. **Данные для MongoDB и Neo4j уже загружены в базу данных movies (выполнить**

установку согласно практической работе на лекции). Вам не нужно создавать CSV-файлы. Для Redis задания выполняются с нуля.

4. Подготовьте и оформите отчет.

Шаг 4. Программное взаимодействие с базами данных (Python)

Этот шаг посвящен работе с MongoDB, Redis и Neo4j с использованием языка Python. Все необходимые библиотеки (pymongo, redis, neo4j) уже установлены в виртуальной машине.

Запустите JupyterLab из меню приложений и создайте новый ноутбук (New notebook...) для выполнения следующих заданий.

4.1. Работа с MongoDB через pymongo

Скопируйте и выполните следующий код в ячейке Jupyter. Он демонстрирует полный цикл CRUD-операций: подключение, создание данных, чтение, обновление и удаление.

```
from pymongo import MongoClient

# 1. Задайте параметры подключения (корректные для VM)
mongo_uri = "mongodb://mongoroot:mongopass@localhost:27017/"
db_name = "student"
collection_name = "test_labs"

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]

    # Очистка коллекции перед началом работы
    collection.delete_many({})

    # 4. Вставка данных (Create)
    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. Закрытие подключения
    if 'client' in locals() and client:
        client.close()
        print("\nПодключение к MongoDB закрыто.")
```

4.2. Работа с Redis через redis-py

Следующий код демонстрирует работу с основными типами данных в Redis. Выполните его в новой ячейке Jupyter.

```
import redis

try:
    # 1. Подключение к Redis (корректные параметры для VM)
    r = redis.Redis(host='localhost', 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}")
```

4.3. Работа с Neo4j через neo4j-driver

Данный код показывает, как выполнить Cypher-запрос к базе данных Neo4j и обработать результаты.

```
from neo4j import GraphDatabase

# 1. Задайте параметры подключения (корректные для VM)
uri = "bolt://localhost:7687"
user = "neo4j"
password = "password" # Используйте пароль, который вы установили при первом входе

def get_movies(tx, movie_limit):
    # 2. Cypher-запрос для получения названий фильмов
    query = """
    MATCH (m:Movie)
    RETURN m.title AS title
    LIMIT $limit
    """
    result = tx.run(query, limit=movie_limit)
    # 3. Извлечение данных из результата
    return [record["title"] for record in result]
```

```
try:
    # 4. Создание сессии и выполнение транзакции
    with GraphDatabase.driver(uri, auth=(user, password)) as driver:
        with driver.session() as session:
            movies = session.read_transaction(get_movies, 5) # Получаем 5 фильмов

            print("Подключение к Neo4j успешно.")
            print("5 фильмов из базы данных:")
            for movie in movies:
                print(f"- {movie}")

except Exception as e:
    print(f"Ошибка при работе с Neo4j: {e}")
```

Задания для самостоятельной работы

№	Задание 1 (MongoDB)	Задание 2 (Neo4j)	Задание 3 (Redis)
1	Найти все фильмы в жанре "Action", выпущенные после 2010 года, и увеличить ($inc) их счетчик голосов (votes) на 50.	Найти всех актеров, которые снимались в одних и тех же фильмах, что и "Tom Hanks" (коллеги по съемочной площадке).	Смоделировать сессию пользователя: с помощью **хэша** (HSET) с ключом user:101 сохранить username и email. Установить срок жизни сессии в 300 секунд (EXPIRE).
2	Найти все фильмы, относящиеся к жанру "Drama", но не относящиеся к жанру "Crime" ($ne). Для найденных фильмов установить ($set) новое поле needs_review в true.	Найти все фильмы, которые срежиссировал "Lilly Wachowski", и вывести названия этих фильмов и имена актеров, которые в них снимались.	Смоделировать систему тегов для статьи: используя **множество** (SADD) с ключом article:55:tags, добавить теги "database", "nosql", "mongodb". Проверить, входит ли тег "sql" в это множество (SISMEMBER).
3	Найти все фильмы, выпущенные в 1994 или 2008 году ($in), и удалить у них поле rank ($unset).	Найти кратчайший путь (shortestPath) в графе между актерами "Kevin Bacon" и "Meg Ryan".	Смоделировать очередь задач: с помощью **списка** (LPUSH) добавить 5 ID задач в очередь task_queue. Затем извлечь (RPOP) 2 задачи для обработки.

4	Используя агрегацию, сгруппировать фильмы по году выпуска (year) и посчитать средний рейтинг ($avg) для каждого года.	Найти всех актеров, которые снимались в фильмах, спродюсированных "Joel Silver".	Смоделировать счетчик онлайн-пользователей. Создать **множество** online_users. Добавить (SADD) 5 ID пользователей. Получить общее количество пользователей онлайн (SCARD).
5	Найти все фильмы, которые были выпущены до 1980 года ИЛИ имеют рейтинг выше 8.9 ($or). Отсортировать результат по году в порядке убывания.	Найти всех людей (актеров или режиссеров), чье имя начинается на "Tom".	Смоделировать таблицу лидеров: с помощью **упорядоченного множества** (ZADD) leaderboard:game1 добавить 4 игроков с их очками. Увеличить счет одного из игроков на 100 очков (ZINCRBY).
6	Найти все фильмы жанров "Family" или "Mystery" ($in) и добавить им в массив genres новый тег "classic" ($addToSet).	Найти всех режиссеров, которые также снимались в фильмах в качестве актеров.	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).
7	Найти все фильмы с рейтингом между 8.5 и 8.7 включительно ($gte, $lte). Для этих фильмов переименовать поле rating в imdb_rating ($rename).	Найти всех актеров, родившихся в тот же год, что и "Keanu Reeves".	Создать два **множества** group:A с ID (1,2,3,4) и group:B с ID (3,4,5,6). Найти их объединение (SUNION) и пересечение (SINTER).
8	Найти все фильмы, у которых поле plotOutline не существует ($exists: false), и установить им это поле со значением "Description pending".	Посчитать, в скольких фильмах снялся каждый актер, и вывести топ-5 актеров по количеству фильмов.	Установить ключ api_key:user7 со значением и сроком жизни 1 час (SET с EX). Проверить оставшееся время жизни ключа (TTL).
9	Используя агрегацию, найти 3 жанра с	Найти актера, сыгравшего роль	Смоделировать ленту новостей: добавить 5

	наибольшим количеством фильмов. (Подсказка: используйте $unwind, $ group, $sort).	"Neo" в фильме "The Matrix", и вывести его год рождения.	новостей в **список** feed:user9 (LPUSH). Оставить в списке только последние 3 новости (LTRIM).
10	Найти все фильмы, в названии которых есть слово "Wars". Удалить все эти фильмы (deleteMany).	Найти все фильмы, выпущенные в 1990-х годах, и их режиссеров.	Используя MSET, одновременно установить ключи config:db:host, config:db:port, config:db:user. Затем получить их значения командой MGET.
11	Найти все фильмы, где в массиве genres есть и "Action", и "Thriller" ($all).	Найти всех людей, которые связаны с фильмом "Cloud Atlas" (актеры, режиссеры и т.д.), и тип их связи.	Создать **хэш** product:77 с полями name, price, stock. Уменьшить значение stock на 5 (HINCRBY с отрицательным значением).
12	Найти 5 самых старых фильмов в коллекции и обновить у них поле is_vintage на true.	Найти всех актеров, которые не снимались в фильмах с "Tom Hanks".	Смоделировать права доступа: создать **множество** user:33:permissions и добавить в него права "read", "write". Проверить, есть ли у пользователя право "delete" (SISMEMBER).
13	Найти все фильмы, выпущенные в XXI веке (год >= 2001), с рейтингом ниже 8.5.	Найти всех режиссеров, снявших более одного фильма в базе данных.	В **упорядоченное множество** ranking:posts добавить 5 постов с их рейтингами. Найти все посты с рейтингом от 100 до 500 (ZRANGEBYSCORE).
14	Для фильма "Pulp Fiction" удалить из массива genres значение "Crime" ($pull).	Найти фильм, в котором вместе снимались "Hugo Weaving" и "Keanu Reeves".	Инкрементировать счетчик (INCR) для ключа page_views:contact 15 раз. Получить итоговое значение.
15	Используя агрегацию, посчитать количество	Найти всех актеров, которые работали с режиссерами	Создать **список** log_errors и добавить в него 3 сообщения

	фильмов, выпущенных в каждое десятилетие.	Вачовски (Lilly или Lana).	об ошибках. Получить последнее добавленное сообщение, не удаляя его из списка (LINDEX).
16	Найти все фильмы, название которых начинается с "The". Для них добавить поле last_updated с текущей датой ($currentDate).	Найти кратчайший путь между "Carrie-Anne Moss" и "Tom Hanks".	Смоделировать подписки на теги: для пользователя user:99 в **множество** user:99:tags добавить 3 тега. Удалить один из тегов (SREM).
17	Найти все фильмы, у которых в массиве genres ровно 2 элемента ($size).	Найти всех актеров, которые снимались в фильме, выпущенном в год их рождения.	Сохранить в **хэш** request:xyz данные о ip, user_agent, timestamp. Получить только ip и user_agent с помощью HMGET.
18	Найти все фильмы жанра "Comedy" с рейтингом выше 8.0, выпущенные до 2000 года.	Найти всех людей, которые являются одновременно и режиссерами, и продюсерами.	В **упорядоченное множество** players_exp добавить 5 игроков с их опытом. Найти ранг (ZRANK) конкретного игрока.
19	Обновить все документы, где year меньше 1970, установив им новое поле era со значением "Old School".	Найти актеров, которые снимались и в "The Matrix", и в "Cloud Atlas".	Используя SETNX, попытаться установить ключ lock:resource:123. Проверить результат. Попытаться установить его еще раз.
20	Найти все фильмы с рейтингом 9.0 или выше. Вывести только их названия и год выпуска, исключив поле _id.	Найти актеров, которые снимались только в одном фильме из представленных в базе.	В **список** queue:priority добавить 3 задачи. Переместить последнюю добавленную задачу в начало списка.
21	Найти фильмы жанра "Biography" и отсортировать их по рейтингу в порядке убывания.	Найти всех режиссеров, которые не снимали фильмы с "Tom Cruise".	Смоделировать голосование: для poll:4 создать **хэш** со счетчиками для option_A и option_B.

			Увеличить счетчик для option_A на 1 (HINCRBY).
22	Удалить все фильмы, у которых ранг (rank) больше 40.	Найти всех актеров, чье имя содержит 4 буквы.	Создать два **множества** tech:frontend (js, css, html) и tech:backend (python, go, js). Найти технологии, которые относятся только к бэкенду (SDIFF).
23	Для всех фильмов, выпущенных в 1999 году, увеличить рейтинг на 0.1 ($inc).	Найти самого возрастного и самого молодого человека в базе данных.	В **упорядоченное множество** events_schedule д обавить 4 события, где оценка — это timestamp. Получить событие, которое произойдет следующим.
24	Найти все фильмы, в которых есть жанр "Sci-Fi", и добавить в них жанр "Science Fiction" ($addToSet).	Найти всех людей, которые не связаны с фильмом "The Matrix" никакими отношениями.	Установить 10 разных ключей (key:1, key:2...) с любыми значениями. Получить все ключи, соответствующие шаблону key:* (KEYS).
25	Используя агрегацию, найти среднее количество голосов (votes) для фильмов в жанре "Action" и "Drama".	Найти пару актеров с наибольшим количеством совместных фильмов.	Смоделировать кеширование страницы. Сохранить HTML-код в ключ cache:page:about со сроком жизни 10 минут (SETEX).
26	Найти фильмы с рейтингом ниже 8.4 и названием, не содержащим "The".	Найти актеров, которые снимались в фильмах, где режиссером был "Clint Eastwood".	Смоделировать "лайки" под постом: в **множество** post:123:likes до бавить 10 ID пользователей, которые поставили лайк. Проверить, лайкнул ли пост пользователь с ID 5.
27	Для всех фильмов жанра "Western" установить поле country в "USA".	Найти все фильмы, в которых количество ролей (отношений ACTED _IN) меньше 3.	В **хэш** user:45 добавить поля visits и last_visit. При каждом запуске скрипта

			увеличивать visits и обновлять last_visit.
28	Найти все фильмы, где массив genres содержит "Adventure", но не содержит "Fantasy".	Найти режиссера, снявшего фильм с самым длинным названием.	Создать **список** messages. Добавить в него 10 сообщений. Получить и удалить из списка первые 3 сообщения.
29	Найти все фильмы, выпущенные в год, когда вышел "The Matrix" (1999).	Построить граф, показывающий "Tom Hanks", его 5 коллег по съемочной площадке и фильмы, которые их связывают.	В **упорядоченное множество** temperatures:city_A добавить замеры температуры за 5 дней (оценка - температура, значение - день). Найти самый холодный и самый теплый день.
30	Удалить поле summary (если оно есть) у всех фильмов, где рейтинг (imdb_rating) выше 9.0.	Найти актеров, которые снимались в фильмах, где они играли роль с именем "John".	Смоделировать черновики постов: для пользователя user:88 создать **хэш** drafts:user:88, где ключи - ID постов, а значения - текст черновика. Добавить и затем удалить один черновик (HDEL).

Оформление отчета

Отчет о выполненной работе должен быть представлен в виде документа и содержать следующие разделы:

1. **Титульный лист:** с указанием названия вуза, дисциплины, темы работы, ФИО студента и преподавателя.
2. **Цель работы:** переформулированная цель из данного методического указания.
3. **Краткое описание процесса выполнения:** шаги, которые были предприняты для выполнения заданий (например, как загружались данные, какие инструменты использовались).
4. **Результаты выполнения индивидуального задания:** для каждого из трех заданий вашего варианта необходимо предоставить:
    - **Постановка задачи:** текст задания из таблицы.

- **Код запроса:** полный текст запроса для MongoDB/Neo4j или скрипт/последовательность команд для Redis.
- **Результат:** скриншот, подтверждающий результат выполнения запроса (вывод из консоли, графическое представление графа, данные из GUI-клиента).

5. **Выводы:** в заключении необходимо кратко описать:
   - Какие основные концепции и операции для каждой из трех СУБД были изучены.
   - С какими трудностями вы столкнулись при выполнении заданий и как их решили.
   - Какие практические навыки работы с NoSQL базами данных были получены.

Критерии оценки (10-балльная система)

- **10 баллов (Отлично):** работа выполнена в полном объеме и без ошибок. Все запросы/команды корректны и оптимальны. Отчет отлично структурирован, содержит подробные пояснения, все необходимые скриншоты и глубокие, осмысленные выводы.
- **8-9 баллов (Хорошо):** работа выполнена полностью, но содержит 1-2 несущественные ошибки или неточности в запросах. Отчет хорошо оформлен, но выводы могли бы быть более развернутыми.
- **5-7 баллов (Удовлетворительно):** работа выполнена не менее чем на 70%. Некоторые запросы некорректны или не достигают поставленной цели. В отчете отсутствуют некоторые обязательные элементы (например, часть скриншотов или пояснений).
- **1-4 балла (Неудовлетворительно):** студент предпринял попытку выполнить работу, но не смог справиться с основными задачами (подключение к БД, выполнение базовых запросов). Отчет практически отсутствует или не соответствует требованиям.
- **0 баллов:** работа не сдана.