**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ**

**ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ**

**ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ**

**Факультет программной инженерии и компьютерной техники**

ПРОЕКТ

«Разработка модели кластера при помощи NoSQL хранилищ»

Студент: Пудов Никита Алексеевич

Группа: P4135

Санкт-Петербург

2024

* 1. **Предметная область**

В рамках данного проекта была использована предметная область ресторан. Предметная область была разбита на три NoSQL базы: Mongo, Cassandra, NEO4J

В Mongo хранилось меню ресторана, данные были следующими:

* \_id – идентификатор (ключ)
* dish\_name – наименование блюда
* price - цена
* category – категория (первое блюдо, второе блюдо, напитки, десерт)

В Cassandra хранились данные о клиентах ресторана (программа лояльности) и данные о работниках ресторана. Пространство ключей у них одно – restaurants. Получилось две таблицы:

Table Clients:

* client\_id – идентификатор клиента
* client\_lastname - фамилия клиента
* client\_firstname – имя клиента
* phone - телефон
* balance – сумма на бонусном счёте
* status – статус в программе лояльности (бронзовый, серебряный, золотой)

PRIMARY KEY ((client\_id), client\_lastname, client\_firstname)

Table Employers:

* employer\_id – идентификатор работника
* employer\_lastname – фамилия работника
* employer\_firstname – имя работника
* position – позиция
* salary – зарплата

PRIMARY KEY ((employer\_id), employer\_lastname, employer\_firstname)

В Neo4j хранилась информация о связях между клиентами, официантами и столами. Официанты обслуживали столы, клиенты резервировали столы.

В базе содержалось три узла: Clients, Officiants, Tables

У Clients свойства: id-ключ, client\_name – фамилия + имя.

У Officiants свойства: id-ключ, officiant\_name- фамилия + имя.

У Tables свойства:id-ключ, table\_number – номер стола

Связь между Clients и Tables – reserved (зарезирвировано)

Связь между Officiants и Tables – serve (обслуживается)

* 1. **Развертывание кластера**

Все три кластера (каждый вид базы – свой кластер) были развернуты при помощи докера. У каждого кластера были свои особенности при развертывании, однако общие моменты можно выделить: задание внешних и внутренних портов контейнера, выделение ресурсов для узла кластера, создание соединения между узлами кластера. Также для каждого кластера необходимо написать docker-compose, в котором прописать все узлы кластера, затем осуществить команды docker-compose build и docker-compose up для поднятия и запуска кластера. Примеры кластеров в докере:

Листинг 1. Mongo docker-compose

|  |
| --- |
| version: "3.8"  services:  mongo1:  image: mongo:6  container\_name: mongo1  command: ["--replSet", "myReplicaSet", "--bind\_ip\_all", "--port", "49005"] #команда для инициирования replicaSet  volumes:  - ./data/mongo-1:/data/db,size=200m  ports:  - 49005:49005  healthcheck: # команда для инициирования членов replica set и запуска узла через mongosh  test: test $$(echo "rs.initiate({\_id:'myReplicaSet',members:[{\_id:0,host:\"mongo1:49005\"},{\_id:1,host:\"mongo2:49006\"},{\_id:2,host:\"mongo3:49007\"}]}).ok || rs.status().ok" | mongosh --port 49005 --quiet) -eq 1  interval: 10s  start\_period: 30s  deploy:  resources:  limits: # лимит выделяемых ресурсов для узла кластера  memory: 512m  cpus: '1'  reservations: # количество ресурсов, которое минимум должно быть выделено узлу кластера  memory: 256m  cpus: '0.5'  mongo2:  image: mongo:6  container\_name: mongo2  command: ["--replSet", "myReplicaSet", "--bind\_ip\_all", "--port", "49006"] #команда для инициирования replicaSet  volumes:  - ./data/mongo-2:/data/db,size=200m  ports:  - 49006:49006  deploy:  resources:  limits: # лимит выделяемых ресурсов для узла кластера  memory: 512m  cpus: '1'  reservations: # количество ресурсов, которое минимум должно быть выделено узлу кластера  memory: 256m  cpus: '0.5'  mongo3:  image: mongo:6  container\_name: mongo3  command: ["--replSet", "myReplicaSet", "--bind\_ip\_all", "--port", "49007"] #команда для инициирования replicaSet  volumes:  - ./data/mongo-3:/data/db,size=200m  ports:  - 49007:49007  deploy:  resources:  limits: # лимит выделяемых ресурсов для узла кластера  memory: 512m  cpus: '1'  reservations: # количество ресурсов, которое минимум должно быть выделено узлу кластера  memory: 256m  cpus: '0.5' |

Листинг 2. Cassandra docker-compose

|  |
| --- |
| version: "3.8"  services:  cassandra-first:  image: cassandra:5.0  container\_name: cassandra-first  volumes:  - ./cassandra-1:/data/database  environment:  - MAX\_HEAP\_SIZE=1024M #максимальный размер Java кучи  - HEAP\_NEWSIZE=128M # минимально выделяемый размер Java кучи  - CASSANDRA\_SEEDS=cassandra-first,cassandra-second,cassandra-third # задание связи между узлами кластера  - CASSANDRA\_CLUSTER\_NAME=myCluster #имя кластера  - CASSANDRA\_ENDPOINT\_SNITCH=GossipingPropertyFileSnitch # протокол общения между узлами gossip  - CASSANDRA\_AUTO\_BOOTSTRAP=true #автоматическое подключение узла к кластеру при его обнаружении  ports:  - "9042:9042" # Порт для внешних подключений к Cassandra  networks:  - database-network  deploy:  resources:  limits: # лимит выделяемых ресурсов для узла кластера  memory: 1.5G  cpus: "1.0"  reservations: # количество ресурсов, которое минимум должно быть выделено узлу кластера  memory: 512M  cpus: "0.5"  cassandra-second:  image: cassandra:5.0  container\_name: cassandra-second  volumes:  - ./cassandra-2:/data/database  environment:  - MAX\_HEAP\_SIZE=1024M #максимальный размер Java кучи  - HEAP\_NEWSIZE=128M # минимально выделяемый размер Java кучи  - CASSANDRA\_SEEDS=cassandra-first,cassandra-second,cassandra-third # задание связи между узлами кластера  - CASSANDRA\_CLUSTER\_NAME=myCluster #имя кластера  - CASSANDRA\_ENDPOINT\_SNITCH=GossipingPropertyFileSnitch # протокол общения между узлами gossip  - CASSANDRA\_AUTO\_BOOTSTRAP=true #автоматическое подключение узла к кластеру при его обнаружении  ports:  - "9043:9042" # Порт для внешних подключений к Cassandra  networks:  - database-network  deploy:  resources:  limits: # лимит выделяемых ресурсов для узла кластера  memory: 1.5G  cpus: "1.0"  reservations: # количество ресурсов, которое минимум должно быть выделено узлу кластера  memory: 512M  cpus: "0.5"  cassandra-third:  image: cassandra:5.0  container\_name: cassandra-third  volumes:  - ./cassandra-3:/data/database  environment:  - MAX\_HEAP\_SIZE=1024M #максимальный размер Java кучи  - HEAP\_NEWSIZE=128M # минимально выделяемый размер Java кучи  - CASSANDRA\_SEEDS=cassandra-first,cassandra-second,cassandra-third # задание связи между узлами кластера  - CASSANDRA\_CLUSTER\_NAME=myCluster #имя кластера  - CASSANDRA\_ENDPOINT\_SNITCH=GossipingPropertyFileSnitch # протокол общения между узлами gossip  - CASSANDRA\_AUTO\_BOOTSTRAP=true #автоматическое подключение узла к кластеру при его обнаружении  ports:  - "9044:9042" # Порт для внешних подключений к Cassandra  networks:  - database-network  deploy:  resources:  limits: # лимит выделяемых ресурсов для узла кластера  memory: 1.5G  cpus: "1.0"  reservations: # количество ресурсов, которое минимум должно быть выделено узлу кластера  memory: 512M  cpus: "0.5"  networks:  database-network:  driver: bridge |

Листинг 3. Neo4j docker-compose

|  |
| --- |
| version: '3.8'  # Custom top-level network  networks:  neo4j-internal:  services:  server1:  # Docker image to be used  image: neo4j:3.5-enterprise  # Hostname  hostname: server1  # Service-level network, which specifies the networks, from the list of the top-level networks (in this case only neo4j-internal), that the server will connect to.  # Adds a network alias (used in neo4j.conf when configuring the discovery members)  networks:  neo4j-internal:  aliases:  - neo4j-network  # The ports that will be accessible from outside the container - HTTP (7474) and Bolt (7687).  ports:  - "7474:7474" #порт HTTP для neo4j  - "7687:7687" #порт, необходимый для связи с узлом кластера по протоколу Bolt  # Uncomment the volumes to be mounted to make them accessible from outside the container.  volumes:  - ./neo4j.conf:/conf/neo4j.conf # This is the main configuration file. Конфигурационный файл  - ./data/server1:/data #папка для данных  - ./logs/server1:/logs #папка для логов  - ./conf/server1:/conf #папка для хранения конфигурационного файла  - ./import/server1:/import #папка для хранения файлов для импорта в базу  # Passes the following environment variables to the container  environment:  - NEO4J\_ACCEPT\_LICENSE\_AGREEMENT=yes #принятие лицензионного соглашения  - NEO4J\_AUTH=neo4j/changeme #задаем данные для аутентификации: пользователь, пароль  - EXTENDED\_CONF=yes #задаем что у нас расширенная конфигурация (расширенные настройки узла кластера)  - NEO4J\_EDITION=docker\_compose #конфигурация узла адаптирована под docker-compose  - NEO4J\_initial\_server\_mode\_\_constraint=PRIMARY #режим запуска узла в режиме Master  # Simple check testing whether the port 7474 is opened.  # If so, the instance running inside the container is considered as "healthy".  # This status can be checked using the "docker ps" command.  healthcheck:  test: ["CMD-SHELL", "wget --no-verbose --tries=1 --spider localhost:7474 || exit 1"] #команда, выполняющаяся в контейнере и проверяющая доступность узла кластера. Режим spider нужен, чтобы wget не загружал файлы  deploy:  resources:  limits: # лимит выделяемых ресурсов для узла кластера  memory: 1.5G  cpus: "1.0"  reservations: # количество ресурсов, которое минимум должно быть выделено узлу кластера  memory: 512M  cpus: "0.5"  # Set up the user  server2:  image: neo4j:3.5-enterprise  hostname: server2  networks:  neo4j-internal:  aliases:  - neo4j-network  ports:  - "7475:7474" #порт HTTP для neo4j  - "7688:7687" #порт, необходимый для связи с узлом кластера по протоколу Bolt  volumes:  - ./neo4j.conf:/conf/neo4j.conf #конфигурационный файл  - ./data/server2:/data #папка для данных  - ./logs/server2:/logs #папка для логов  - ./conf/server2:/conf #папка для хранения конфигурационного файла  - ./import/server2:/import #папка для хранения файлов для импорта в базу  environment:  - NEO4J\_ACCEPT\_LICENSE\_AGREEMENT=yes #принятие лицензионного соглашения  - NEO4J\_AUTH=neo4j/changeme #задаем данные для аутентификации: пользователь, пароль  - EXTENDED\_CONF=yes #задаем что у нас расширенная конфигурация (расширенные настройки узла кластера)  - NEO4J\_EDITION=docker\_compose #конфигурация узла адаптирована под docker-compose  - NEO4J\_initial\_server\_mode\_\_constraint=SECONDARY #режим запуска узла кластера в Slave  healthcheck:  test: ["CMD-SHELL", "wget --no-verbose --tries=1 --spider localhost:7474 || exit 1"] #команда, выполняющаяся в контейнере и проверяющая доступность узла кластера. Режим spider нужен, чтобы wget не загружал файлы  deploy:  resources:  limits: # лимит выделяемых ресурсов для узла кластера  memory: 1.5G  cpus: "1.0"  reservations: # количество ресурсов, которое минимум должно быть выделено узлу кластера  memory: 512M  cpus: "0.5"  server3:  image: neo4j:3.5-enterprise  hostname: server3  networks:  neo4j-internal:  aliases:  - neo4j-network  ports:  - "7476:7474" #порт HTTP для neo4j  - "7689:7687" #порт, необходимый для связи с узлом кластера по протоколу Bolt  volumes:  - ./neo4j.conf:/conf/neo4j.conf #конфигурационный файл  - ./data/server3:/data #папка для данных  - ./logs/server3:/logs #папка для логов  - ./conf/server3:/conf #папка для хранения конфигурационного файла  - ./import/server3:/import #папка для хранения файлов для импорта в базу  environment:  - NEO4J\_ACCEPT\_LICENSE\_AGREEMENT=yes #принятие лицензионного соглашения  - NEO4J\_AUTH=neo4j/changeme #задаем данные для аутентификации: пользователь, пароль  - EXTENDED\_CONF=yes #задаем что у нас расширенная конфигурация (расширенные настройки узла кластера)  - NEO4J\_EDITION=docker\_compose #конфигурация узла адаптирована под docker-compose  - NEO4J\_initial\_server\_mode\_\_constraint=SECONDARY #режим запуска узла кластера в Slave  healthcheck:  test: ["CMD-SHELL", "wget --no-verbose --tries=1 --spider localhost:7474 || exit 1"] #команда, выполняющаяся в контейнере и проверяющая доступность узла кластера. Режим spider нужен, чтобы wget не загружал файлы  deploy:  resources:  limits: # лимит выделяемых ресурсов для узла кластера  memory: 1.5G  cpus: "1.0"  reservations: # количество ресурсов, которое минимум должно быть выделено узлу кластера  memory: 512M  cpus: "0.5"  server4:  image: neo4j:3.5-enterprise  hostname: server4  networks:  neo4j-internal:  aliases:  - neo4j-network  ports:  - "7477:7474" #порт HTTP для neo4j  - "7690:7687" #порт, необходимый для связи с узлом кластера по протоколу Bolt  volumes:  - ./neo4j.conf:/conf/neo4j.conf #конфигурационный файл  - ./data/server4:/data #папка для данных  - ./logs/server4:/logs #папка для логов  - ./conf/server4:/conf #папка для хранения конфигурационного файла  - ./import/server4:/import #папка для хранения файлов для импорта в базу  environment:  - NEO4J\_ACCEPT\_LICENSE\_AGREEMENT=yes #принятие лицензионного соглашения  - NEO4J\_AUTH=neo4j/changeme #задаем данные для аутентификации: пользователь, пароль  - EXTENDED\_CONF=yes #задаем что у нас расширенная конфигурация (расширенные настройки узла кластера)  - NEO4J\_EDITION=docker\_compose #конфигурация узла адаптирована под docker-compose  - NEO4J\_initial\_server\_mode\_\_constraint=SECONDARY #режим запуска узла кластера в Slave  healthcheck:  test: ["CMD-SHELL", "wget --no-verbose --tries=1 --spider localhost:7474 || exit 1"] #команда, выполняющаяся в контейнере и проверяющая доступность узла кластера. Режим spider нужен, чтобы wget не загружал файлы  deploy:  resources:  limits: # лимит выделяемых ресурсов для узла кластера  memory: 1.5G  cpus: "1.0"  reservations: # количество ресурсов, которое минимум должно быть выделено узлу кластера  memory: 512M  cpus: "0.5" |

* 1. **Написание API запросов к БД**

Для каждой из баз был написан свой API для запросов. Все API Были написаны на Python. Примеры каждого API:

Листинг 4. API Mongo

|  |
| --- |
| def generate\_random\_data(num\_records):  fake = Faker()  categories = ["первое блюдо", "второе блюдо", "напитки", "десерт"]    data = []  for \_ in range(num\_records):  dish\_name = fake.text(max\_nb\_chars=30)  price = random.randint(1000, 5000)  category = random.choice(categories)  data.append([dish\_name, price, category])    return data  # Создание DataFrame из сгенерированных данных  def create\_dataframe(num\_records):  data = generate\_random\_data(num\_records)  df = pd.DataFrame(data, columns=["dish\_name", "price", "category"])  return df  start = time.time()  collection.insert\_many(df.to\_dict("records"))  end = time.time() - start  # Выборка всех записей из коллекции  start = time.time()  result = collection.find({})  end = time.time() – start  def get\_dishes\_by\_category(collection, category):  query = {"category": category}  result = collection.find(query)  return result  def get\_dishes\_below\_price(collection, price\_threshold):  query = {"price": {"$lt": price\_threshold}}  result = collection.find(query)  return result  def get\_dishes\_in\_price\_range(collection, min\_price, max\_price):  query = {"price": {"$gte": min\_price, "$lte": max\_price}}  result = collection.find(query)  return result  # Получение всех блюд определенной категории  category = "первое блюдо"  start = time.time()  dishes\_by\_category = get\_dishes\_by\_category(collection, category)  end = time.time() - start  print(f"All dishes in category '{category}':")  for dish in dishes\_by\_category:  print(dish)  # Получение всех блюд с ценой ниже заданного порога  price\_threshold = 3000  start = time.time()  dishes\_below\_price = get\_dishes\_below\_price(collection, price\_threshold)  end = time.time() - start  print(f"All dishes below price {price\_threshold}:")  for dish in dishes\_below\_price:  print(dish)  # Получение всех блюд с ценой в диапазоне между заданными значениями  min\_price = 2000  max\_price = 4000  start = time.time()  dishes\_in\_price\_range = get\_dishes\_in\_price\_range(collection, min\_price, max\_price)  end = time.time() - start  print(f"All dishes in price range {min\_price}-{max\_price}:")  for dish in dishes\_in\_price\_range:  print(dish)  def delete\_all\_records(collection):  result = collection.delete\_many({})  print(f"Удалено {result.deleted\_count} записей.") |

Листинг 5. API Cassandra

|  |
| --- |
| # Создание пространства ключей  session.execute("CREATE KEYSPACE IF NOT EXISTS restaurant WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication\_factor' : '3' }")  # Использование созданного пространства ключей  session.set\_keyspace('restaurant')  # Создание таблицы Employer\_by\_Name  session.execute("""  CREATE TABLE IF NOT EXISTS Employer\_by\_Name (  employer\_id UUID,  employer\_lastname TEXT,  employer\_firstname TEXT,  position TEXT,  salary DECIMAL,  PRIMARY KEY ((employer\_id), employer\_lastname, employer\_firstname)  )  """)  # Создание таблицы Client\_by\_Bonus  session.execute("""  CREATE TABLE IF NOT EXISTS Client\_by\_Bonus (  client\_id UUID,  client\_lastname TEXT,  client\_firstname TEXT,  status TEXT,  balance DECIMAL,  PRIMARY KEY ((client\_id), client\_lastname, client\_firstname)  )  """)  def generate\_data\_employers(num\_records):  faker = Faker()  data = []  # Генерация и вставка данных в таблицу  for \_ in range(num\_records):  employer\_id = uuid.uuid4()  employer\_lastname = faker.last\_name()  employer\_firstname = faker.first\_name()  position = faker.job()  salary = random.randint(40000, 200000)  data.append((employer\_id, employer\_lastname, employer\_firstname, position, salary))  return data  def generate\_data\_clients(num\_records):  faker = Faker()  data = []  for \_ in range(num\_records):  client\_id = uuid.uuid4()  client\_lastname = faker.last\_name()  client\_firstname = faker.first\_name()  status = faker.random\_element(elements=('bronse', 'serebro', 'gold'))  balance = random.randint(0, 10000)  data.append((client\_id, client\_lastname, client\_firstname, status, balance))  return data  # Функция для заполнения таблицы Employer\_by\_Name рандомными данными  def fill\_employer\_table(session, data):  # Параллельная вставка данных  query = """  INSERT INTO Employer\_by\_Name (  employer\_id, employer\_lastname, employer\_firstname, position, salary  ) VALUES (%s, %s, %s, %s, %s)  """  execute\_concurrent\_with\_args(session, query, data, concurrency=100)  # Функция для заполнения таблицы Client\_by\_Bonus рандомными данными  def fill\_client\_table(session, data):  query = """  INSERT INTO Client\_by\_Bonus (  client\_id, client\_lastname, client\_firstname, status, balance  ) VALUES (%s, %s, %s, %s, %s)  """  execute\_concurrent\_with\_args(session, query, data, concurrency=100)  # Функция для выборки записей из таблицы Employer\_by\_Name  def select\_employers(session):  rows = session.execute("SELECT \* FROM Employer\_by\_Name")  for row in rows:  print(row)  # Функция для выборки записей из таблицы Client\_by\_Bonus  def select\_clients(session):  rows = session.execute("SELECT \* FROM Client\_by\_Bonus")  for row in rows:  print(row)  def get\_employees\_by\_position(session, position):  query = f"SELECT \* FROM Employer\_by\_Name WHERE position = '{position}' ALLOW FILTERING"  result = session.execute(query)  return result  def get\_clients\_by\_status(session, status):  query = f"SELECT \* FROM Client\_by\_Bonus WHERE status = '{status}' ALLOW FILTERING"  result = session.execute(query)  return result  def get\_employees\_above\_salary(session, salary\_threshold):  query = f"SELECT \* FROM Employer\_by\_Name WHERE salary > {salary\_threshold} ALLOW FILTERING"  result = session.execute(query)  return result  def get\_clients\_below\_balance(session, balance\_threshold):  query = f"SELECT \* FROM Client\_by\_Bonus WHERE balance < {balance\_threshold} ALLOW FILTERING"  result = session.execute(query)  return result  def truncate\_tables(session):  # Очистка таблицы Employer\_by\_Name  session.execute("TRUNCATE TABLE Employer\_by\_Name")  # Очистка таблицы Client\_by\_Bonus  session.execute("TRUNCATE TABLE Client\_by\_Bonus") |

Листинг 6. API Neo4j

|  |
| --- |
| # Функция для очистки базы данных  def clear\_database(driver):      # Создание сессии      with driver.session() as session:          # Удаление всех узлов и отношений          session.run("MATCH (n) DETACH DELETE n")  # Функция для вывода информации о узлах  def print\_node\_info(driver):      try:          # Получение информации об узлах          with driver.session() as session:              result = session.run("MATCH (n) RETURN n")              print("Информация об узлах:")              for record in result:                  node\_properties = record["n"]                  print(dict(node\_properties.items()))      except Exception as e:          print(f"Произошла ошибка: {e}")  def generate\_clients\_csv(num\_records):      fake = Faker()      names = []      phones = []      # Заполняем DataFrame данными о клиентах      for \_ in tqdm(range(num\_records), desc="Генерация данных"):          client\_name = fake.name()          phone = fake.phone\_number()          names.append(client\_name)          phones.append(phone)        d = {"client\_name": names, "phone": phones}      clients\_df = pd.DataFrame(d)      # Записываем DataFrame в CSV файл с отслеживанием прогресса      with tqdm(total=len(clients\_df), desc="Запись в файл") as pbar:          clients\_df.to\_csv("clients.csv", index=False)          pbar.update(len(clients\_df))  def generate\_officiants(num\_records):      fake = Faker()        officiants\_data = []      for \_ in range(num\_records):          officiant\_name = fake.name()          officiants\_data.append({"officiant\_name": officiant\_name})      return officiants\_data  def generate\_tables(num\_records):      tables\_data = []      for i in range(num\_records):          table\_number = i + 1          tables\_data.append({"table\_number": table\_number})      return tables\_data  def load\_clients\_into\_neo4j(driver):      query\_string = '''      USING PERIODIC COMMIT 1000      LOAD CSV WITH HEADERS FROM 'file:///clients.csv' AS row      CREATE (c:Client {client\_name: row.client\_name, phone: row.phone})      '''      with driver.session() as session:          session.run(query\_string)  def create\_clients\_relationships(driver):      query\_string = '''      USING PERIODIC COMMIT 1000      LOAD CSV WITH HEADERS FROM 'file:///clients.csv' AS row      WITH row ORDER BY rand() LIMIT 1000      MATCH (t:Table)      WITH row, t ORDER BY rand() LIMIT 1000      CREATE (c:Client {client\_name: row.client\_name, phone: row.phone})-[:reserved]->(t)      '''      with driver.session() as session:          session.run(query\_string)  def create\_nodes\_and\_relationships(driver, officiants\_data, tables\_data):      try:          with driver.session() as session:              session.run("UNWIND $officiants\_data AS data "                          "CREATE (:Officiant {officiant\_name: data.officiant\_name})", officiants\_data=officiants\_data)          with driver.session() as session:              session.run("UNWIND $tables\_data AS data "                          "CREATE (:Table {table\_number: data.table\_number})", tables\_data=tables\_data)          num\_officiants = len(officiants\_data)          num\_tables = len(tables\_data)          with driver.session() as session:              session.run("MATCH (o:Officiant), (t:Table) "                          "WHERE id(o) % $num\_officiants = id(t) % $num\_tables "                          "CREATE (o)-[:serve]->(t)", num\_officiants=num\_officiants, num\_tables=num\_tables)      except Exception as e:          print(f"Произошла ошибка: {e}")  # Функция для вывода информации о узлах и отношениях  def print\_node\_and\_relationship\_info(driver):      try:          with driver.session() as session:              # Получение информации об узлах              print("Узлы:")              nodes\_result = session.run("MATCH (n) RETURN n")              for record in nodes\_result:                  node = record["n"]                  print("Узел:")                  for key, value in dict(node.items()).items():                      print(f"{key}: {value}")              # Получение информации об отношениях              print("\nОтношения:")              relationships\_result = session.run("MATCH ()-[r]->() RETURN r")              for record in relationships\_result:                  relationship = record["r"]                  print("Отношение:")                  for key, value in dict(relationship.items()).items():                      print(f"{key}: {value}")      except Exception as e:          print(f"Произошла ошибка: {e}")  # Вывод информации об узлах клиентов  with driver.session() as session:      result = session.run("MATCH (c:Client) RETURN c")      print("Узлы клиентов:")      for record in result:          print(record["c"])  def get\_all\_clients(driver):      with driver.session() as session:          result = session.run("MATCH (c:Client) RETURN c")          return [record["c"] for record in result]  def get\_all\_tables(driver):      with driver.session() as session:          result = session.run("MATCH (t:Table) RETURN t")          return [record["t"] for record in result]  def get\_all\_officiants(driver):      with driver.session() as session:          result = session.run("MATCH (o:Officiant) RETURN o")          return [record["o"] for record in result]  def get\_clients\_reserved\_for\_table(driver, table\_number):      with driver.session() as session:          result = session.run(              "MATCH (c:Client)-[:reserved]->(t:Table) WHERE t.table\_number = $table\_number RETURN c",              table\_number=table\_number          )          return [record["c"] for record in result]  def get\_tables\_served\_by\_officiant(driver, officiant\_name):      with driver.session() as session:          result = session.run(              "MATCH (o:Officiant)-[:serve]->(t:Table) WHERE o.officiant\_name = $officiant\_name RETURN t",              officiant\_name=officiant\_name          )          return [record["t"] for record in result] |

* 1. **Проведение измерений скорости вставки данных**

Для каждого из кластеров были проведены нагрузочные тесты по вставке данных в него. Генерировалось 2 млн записей для каждой из баз. Для генерации осмысленных данных использовалась библиотека Faker. В процессе вставки использовались возможности параллельной обработки каждой базы. Сама генерация данных занимает 5 минут.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Mongo | Cassandra | Neo4j |
| Время вставки данных | 1 минута | 16 минут | <1 минуты |

* 1. **Отказоустойчивость кластера**

В каждом из кластеров были осуществлены попытки сделать отказ кластера под нагрузкой. Для cassandra это достигается уменьшением памяти в Java кучи. Например, при максимальном размере кучи 256 Мбайт кассандра падала при вставке данных

Для Mongo и Neo4j достигается уменьшением количества выделяемых ресурсов на узел. Если резко ограничить лимит выделяемых ресурсов до условного значения 10 Мбайт, то мы уроним кластер.

Также возможно принудительно отключать ноду кластера через docker.

**Выводы**

В ходе выполнения проекта была выбрана предметная область ресторан. Для данной предметной области определен набор хранимых данных и распределен по трём NoSQL базам – Mongo, Cassandra, Neo4j. Для каждой базы был создан кластер из минимум трёх узлов. Для создания кластера применялись средства контейнеризации Docker. Все базы были заполнены сгенированными данными в размере 2 млн записей. Для генерации осмысленных данных использовалась библиотека Faker. Для взаимодействия с базами и выполнение запросов был написан API. В ходе работы выяснилось, что самая долгая вставка данных выполнялась в Cassandra. Также проведены эксперименты по тестированию отказоустойчивости кластера (изменение выделяемых ресурсов при заполнении баз большим массивом данных).