Проектування високонавантажених систем

Лабораторна робота 5

Торгало Ігнатій ФБ-51мп

1. Розгортання Cassandra у Docker та підключення через cqlsh

Запущено контейнер з Cassandra та виконано підключення до СУБД через cqlsh.

Рисунок 1 — Запущений контейнер Cassandra та підключення через cqlsh

2. Створення keyspace shop та таблиці items

Створено keyspace з SimpleStrategy та replication_factor = 1. Таблиця items спроєктована так, щоб швидко виконувати запити в межах категорії: category використано як partition key, а price (та id) — як clustering key для сортування за ціною. Для додаткових властивостей товару використано тип map (props), а для пошуку по вмісту map створено індекс.

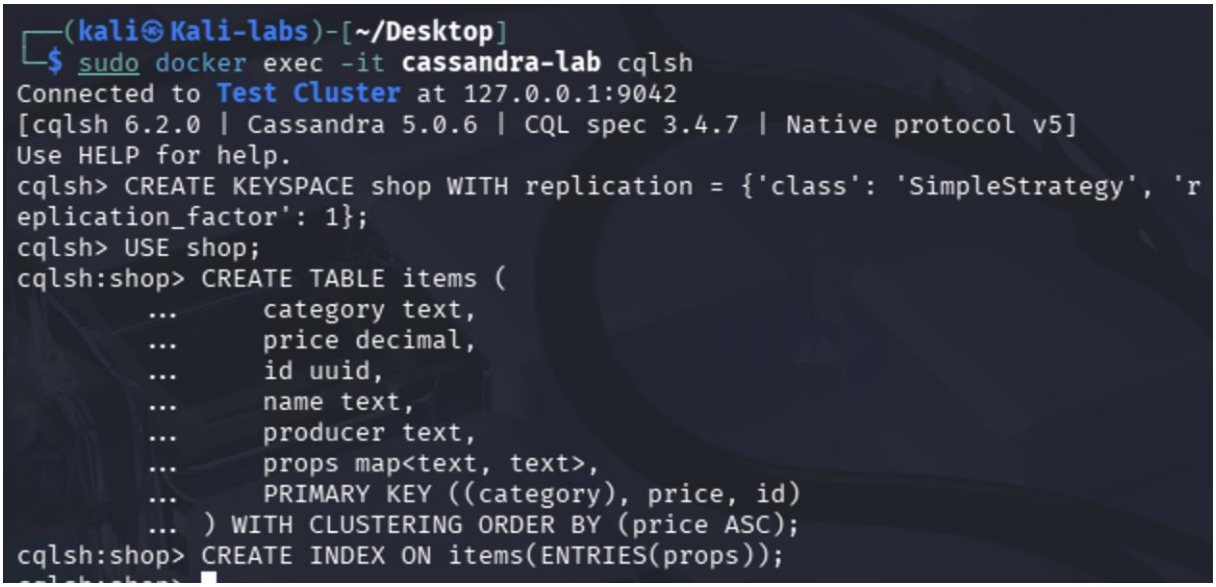

Рисунок 2 — Створення keyspace shop та таблиці items з індексом на map

3. Наповнення таблиці items тестовими даними

Додано тестові товари різних категорій (Electronics, Books) з різними наборами властивостей у props.

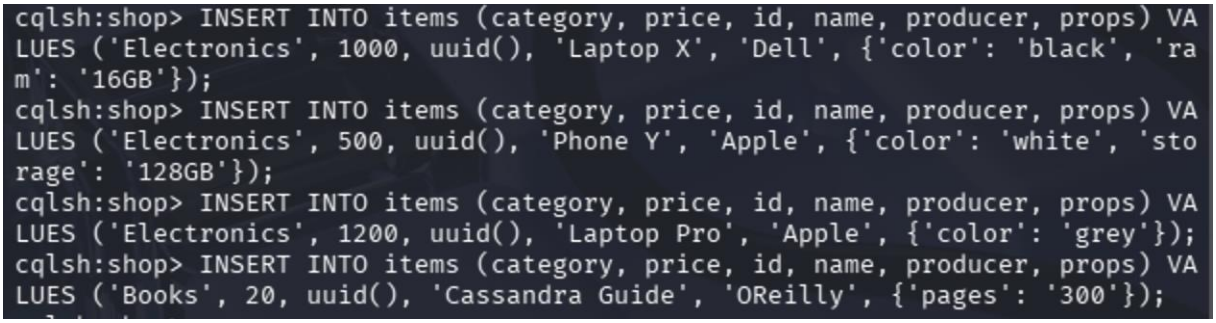

Рисунок 3 — Додавання тестових даних у items

4. Перевірка структури items та вибірка товарів у категорії

Використано DESCRIBE для перегляду структури таблиці та SELECT для вибірки всіх товарів у категорії. У запитах ALLOW FILTERING не використовувалось.

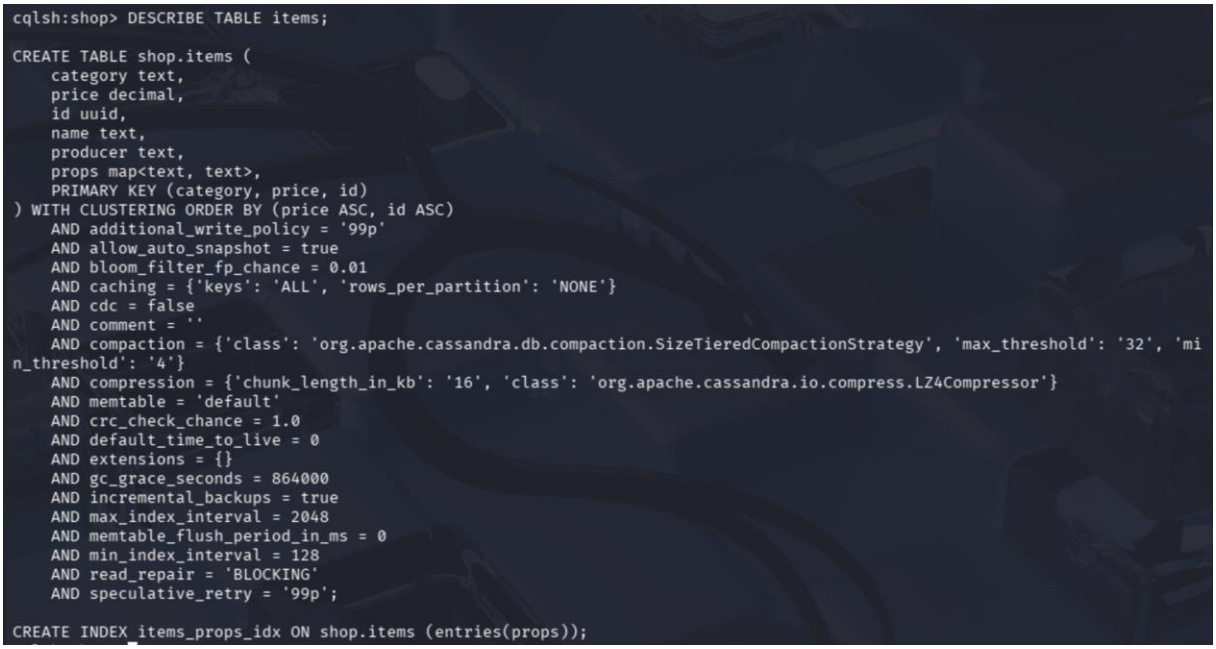

Рисунок 4 — Результат DESCRIBE TABLE items

Рисунок 5 — Вибірка товарів у категорії Electronics (відсортовано за price)

5. Створення таблиці orders та запити до замовлень

Створено таблицю orders для зберігання замовлень користувачів. Для швидкого доступу до замовлень конкретного замовника customer_name використано як partition key, a order_time — як clustering key з сортуванням за часом (DESC). Виконано вибірку замовлень клієнта, підрахунок суми та отримання часу запису ціни через WRITETIME.

Рисунок 6 — Таблиця orders та запити (вибірка, sum, WRITETIME)

6. Розгортання кластеру Cassandra з 3 нод та перевірка конфігурації

Розгорнуто кластер з трьох нод. Перевірено стан та склад кластеру командою nodetool status.

Для розгортання кластеру використано docker-compose.yml з трьома сервісами (cassandra-1/2/3), seed-нодою cassandra-1 та параметрами обмеження heap (MAX_HEAP_SIZE, HEAP_NEWSIZE). Додано healthcheck через cqlsh, щоб наступні ноди стартували лише після готовності seed-ноди.

```yaml
services:
  cassandra-1:
    image: cassandra:latest
    container_name: cassandra-1
    environment:
      - CASSANDRA_SEEDS=cassandra-1
      - CASSANDRA_CLUSTER_NAME=MyCluster
      - CASSANDRA_ENDPOINT_SNITCH=GossipingPropertyFileSnitch
      - CASSANDRA_DC=DC1
      # Ограничиваем память!
      - MAX_HEAP_SIZE=512M
      - HEAP_NEWSIZE=100M
    ports:
      - 9042:9042
    healthcheck:
      test: ["CMD", "cqlsh", "-e", "describe keyspaces"]
      interval: 30s
      timeout: 10s
      retries: 5

  cassandra-2:
    image: cassandra:latest
    container_name: cassandra-2
    environment:
      - CASSANDRA_SEEDS=cassandra-1
      - CASSANDRA_CLUSTER_NAME=MyCluster
      - CASSANDRA_ENDPOINT_SNITCH=GossipingPropertyFileSnitch
      - CASSANDRA_DC=DC1
      - MAX_HEAP_SIZE=512M
      - HEAP_NEWSIZE=100M
    depends_on:
      cassandra-1:
        condition: service_healthy

  cassandra-3:
    image: cassandra:latest
    container_name: cassandra-3
    environment:
      - CASSANDRA_SEEDS=cassandra-1
      - CASSANDRA_CLUSTER_NAME=MyCluster
      - CASSANDRA_ENDPOINT_SNITCH=GossipingPropertyFileSnitch
      - CASSANDRA_DC=DC1
      - MAX_HEAP_SIZE=512M
      - HEAP_NEWSIZE=100M
    depends_on:
      cassandra-1:
        condition: service_healthy

```

Рисунок 7 — Фрагмент docker-compose.yml для 3 нод Cassandra

```
┌──(kali㉿Kali-labs)-[~/Desktop/cassandra]
└─$ sudo docker exec -it cassandra-1 nodetool status
Datacenter: DC1
═══════════════
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address     Load       Tokens  Owns (effective)  Host ID                               Rack
UN  172.19.0.4  80.03 KiB  16      59.3%             0e0c2d4f-47c5-4b15-b0fa-2128351ce9e8  rack1
UN  172.19.0.2  119.83 KiB 16      64.7%             765176ec-2644-4c69-8ee1-97a678b0e898  rack1
UN  172.19.0.3  147.5 KiB  16      76.0%             a529d3c1-6def-4b95-9e3b-183cae4afbf5  rack1
```

Рисунок 8 — nodetool status (3 ноди у статусі UN)

7. Keyspace з replication factor 1/2/3 та тестові таблиці

Створено три keyspace (rf_1, rf_2, rf_3) з SimpleStrategy та replication_factor 1, 2, 3. У кожному keyspace створено просту таблицю test та вставлено тестовий рядок.

```
└─$ sudo docker exec -it cassandra-1 cqlsh
Connected to MyCluster at 127.0.0.1:9042
[cqlsh 6.2.0 | Cassandra 5.0.6 | CQL spec 3.4.7 | Native protocol v5]
Use HELP for help.
cqlsh> CREATE KEYSPACE rf_1 WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
cqlsh> CREATE KEYSPACE rf_2 WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 2};
cqlsh> CREATE KEYSPACE rf_3 WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 3};
cqlsh>
cqlsh> CREATE TABLE rf_1.test (id int PRIMARY KEY, val text);
cqlsh> CREATE TABLE rf_2.test (id int PRIMARY KEY, val text);
cqlsh> CREATE TABLE rf_3.test (id int PRIMARY KEY, val text);
cqlsh>
cqlsh> INSERT INTO rf_1.test (id, val) VALUES (1, 'data_rf1');
cqlsh> INSERT INTO rf_2.test (id, val) VALUES (1, 'data_rf2');
cqlsh> INSERT INTO rf_3.test (id, val) VALUES (1, 'data_rf3');
```

Рисунок 9 — Створення rf_1/rf_2/rf_3, таблиць test та вставка даних

8. Відключення ноди та перевірка читання/запису з різними рівнями Consistency

Відключено одну ноду (cassandra-3) та повторно перевірено стан кластеру. Далі виконано спроби читання з різних keyspace при різних рівнях Consistency. Спостереження:

• Для rf_1 (RF=1) при недоступності ноди, яка зберігала єдину репліку, читання може завершуватися помилкою навіть на CONSISTENCY ONE.

• Для rf_3 (RF=3) читання на CONSISTENCY ONE залишається доступним (потрібна відповідь хоча б від однієї з живих реплік).

• Для rf_3 при CONSISTENCY ALL потрібні відповіді від усіх реплік; при недоступності однієї ноди отримуємо помилку Unavailable.

```
┌──(kali㉿Kali-labs)-[~/Desktop/cassandra]
└─$ sudo docker stop cassandra-3
cassandra-3

┌──(kali㉿Kali-labs)-[~/Desktop/cassandra]
└─$ sudo docker exec -it cassandra-1 nodetool status
Datacenter: DC1
═══════════════
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address     Load        Tokens  Owns  Host ID                               Rack
UN  172.19.0.4  80.75 KiB   16      ?     0e0c2d4f-47c5-4b15-b0fa-2128351ce9e8  rack1
UN  172.19.0.2  120.62 KiB  16      ?     765176ec-2644-4c69-8ee1-97a678b0e898  rack1
DN  172.19.0.3  145.73 KiB  16      ?     a529d3c1-6def-4b95-9e3b-183cae4afbf5  rack1

Note: Non-system keyspaces don't have the same replication settings, effective ownership information is meaningless
```

Рисунок 10 — Відключення ноди cassandra-3 та стан кластера (DN)

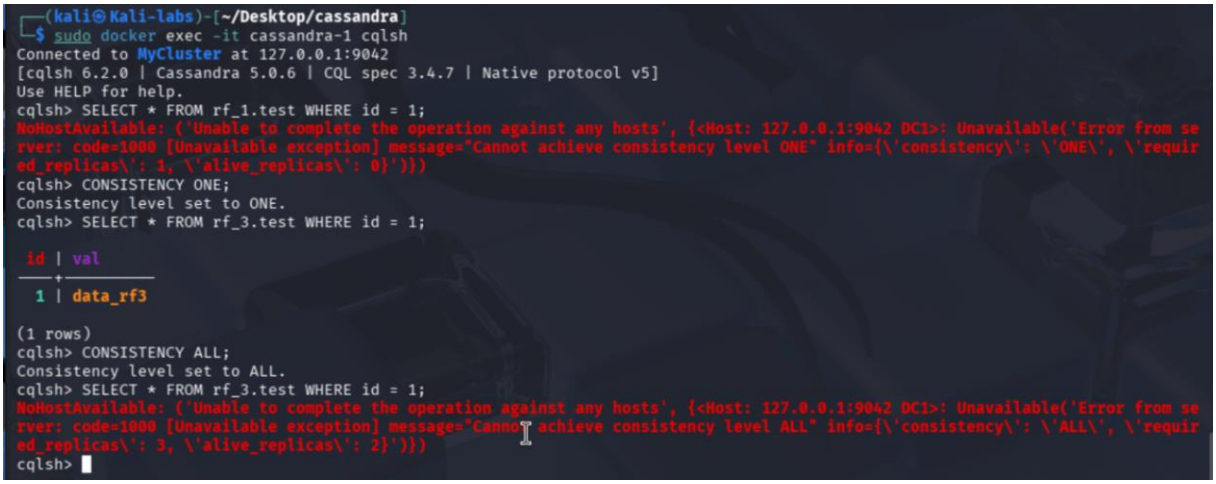

Рисунок 11 — Перевірка CONSISTENCY (ONE працює, ALL повертає Unavailable)

9. Аналіз продуктивності counter: ONE vs QUORUM

Таблиця з counter має бути створена у keyspace з RF=3. Далі 10 клієнтів паралельно виконують інкремент по 10_000 разів. Необхідно порівняти час виконання та кінцеве значення лічильника для Consistency ONE та QUORUM (очікувано 100000).

Результати:

• Consistency ONE: час виконання ≈ 15.86 сек, кінцеве значення = 100000 (очікуване)

• Consistency QUORUM: час виконання ≈ 11.42 сек, кінцеве значення = 100000 (очікуване)

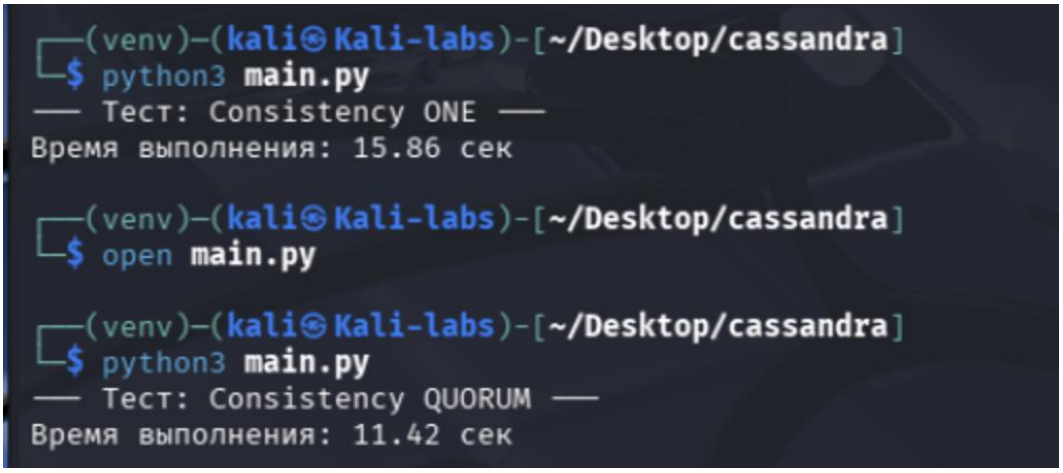

Рисунок 12 — Порівняння часу виконання тесту для Consistency ONE та QUORUM

10. Висновок

У ході лабораторної роботи було розгорнуто Cassandra у Docker та виконано моделювання даних у форматі column family. Було створено таблиці items та orders з правильно підібраними ключами для швидких запитів у межах partition key (category для items та customer_name для orders) без використання ALLOW FILTERING. Також було розгорнуто кластер з 3 нод та досліджено вплив replication factor і Consistency Level на доступність читання. Експеримент показав, що RF=1 робить систему вразливою до відмови ноди з даними, тоді як RF=3 дозволяє читати з CONSISTENCY ONE при відмові однієї ноди, але CONSISTENCY ALL стає недоступним. Додатково виконано тест продуктивності для counter (RF=3): Consistency ONE — 15.86 сек, Consistency QUORUM — 11.42 сек (значення можуть відрізнятися через прогрів кешу та навантаження системи).