Проєктування високонавантаженних систем. Лабораторна 5, звіт

Михайло Голуб

17 грудня 2025 р.

1 Завдання лабораторної роботи

1.1 Частина 1. Робота зі структурами даних у Cassandra

1. Ознайомитись з особливістю моделювання даних у Cassandra;
2. Створити `keyspace` з найпростішою стратегією реплікації;
3. Створити дві таблиці: `items` та `orders`;
4. Таблиця `items` містить різноманітні товари (тобто у яких різний набір властивостей). Для набору властивостей товару виберіть базові характеристики однакові для всіх товарів (`name, category, price, producer,` ...), а для властивостей які відрізняються використовуйте тип map (створивши індекс для можливості пошуку по її вмісту) Необхідно, щоб пошук швидко працював для категорії (`category`) товарів. Ця вимога має бути врахована при створенні ключа для таблиці (тобто, `category` має бути `partition key`);
5. Наповнити таблицю тестовими данними;
6. Написати запит, який показує структуру створеної таблиці (команда `DESCRIBE`);
7. Написати запит, який виводить усі товари в певній категорії відсортовані за ціною;
8. Напишіть запити, які вибирають товари за різними критеріями в межах певної категорії (тут де треба замість індексу використайте Matirialized view): назва, ціна (в проміжку), ціна та виробник
9. Створіть таблицю orders в якій міститься ім'я замовника і інформація про замовлення: перелік id-товарів у замовленні, вартість замовлення, дата замовлення, ... Для кожного замовника повинна бути можливість швидко шукати його замовлення і виконувати по них запити. Ця вимога має бути врахована при створенні ключа для таблиці (аналогічно як для items);

10. Напишіть запит, який показує структуру створеної таблиці (команда DESCRIBE);
11. Для замовника виведіть всі його замовлення відсортовані за часом коли вони були зроблені;
12. Для кожного замовників підрахуйте загальну суму на яку були зроблені усі його замовлення;
13. Для кожного замовлення виведіть час коли його ціна були занесена в базу (SELECT WRITETIME);

!!! У запитах заборонено використовувати ALLOW FILTERING !!!

1.2 Частина 2. Налаштування реплікації у Cassandra

1. Сконфігурувати кластер з 3-х нод;
2. Перевірити правильність конфігурації за допомогою `nodetool status`;
3. Використовуючи `cqlsh`, створити три `keyspace` з `replication factor` 1, 2, 3 з `SimpleStrategy`;
4. В кожному з кейспейсів створити прості таблиці;
5. Спробувати писати і читати в ці таблиці підключаючись до різних нод через `cqlsh`;
6. Вставте дані в створені таблиці і подивіться на їх розподіл по вузлах кластера окремо; для кожного з кейспейсів (команда `nodetool status`) – має бути видно відсоток даних який зберігається на ноді;
7. Для якогось запису з кожного з кейспейсу виведіть ноди на яких зберігаються дані – має бути видно ip-адреси вузлів на яких зберігається даний рядок;
8. Відключити одну з нод. Для кожного з кейспейсів перевірити з якими рівнями consistency можемо читати та писати: для `replication factor` 1 – `CONSISTENCY ONE`, для 2 – `CONSISTENCY ONE/TWO`, для 3 – `CONSISTENCY ONE/TWO/THREE`;
9. Зробити так щоб три ноди працювали, але не бачили одна одну по мережі (заблокувати чи відключити зв'язок між ними);
10. Для кейспейсу з replication factor 3 задати рівень consistency рівним 1. Виконати по черзі запис значення з однаковим primary key, але різними іншими значенням окремо на кожну з нод (тобто створіть конфлікт);
11. Відновити зв'язок між нодами, перевірити що вони знову об'єднались у кластер. Визначити яким чином був вирішений конфлікт даних та яке значення було прийнято кластером та за яким принципом.

1.3 Частина 3. Аналіз продуктивності та перевірка цілісності

Аналогічно попереднім завданням, необхідно, для кластеру налаштованому у попередній частині, створити таблицю з каунтером лайків. Далі з 10 окремих клієнтів одночасно запустити інкерементацію каунтеру лайків по 10_000 на кожного клієнта з різними опціями взаємодії з Cassandra. Таблиця має бути створена у Keyspace з replication factor 3. Для створення каунтеру використовуйте спеціальний тип колонки – counter (цей тип буде підтримувати операції increment/decrement in-place):

- Вказавши у параметрах запиту `Consistency Level One` (це буде означати, що запис відбувається синхронно тільки на одну ноду), запустіть 10 клієнтів з інкрементом по 10_000 на кожному з них. Виміряйте час виконання та перевірте чи кінцеве значення буде дорівнювати очікуваному – 100K
- Вказавши у параметрах запиту `Consistency Level QUORUM` (це буде означати, що запис відбувається синхронно на більшість нод), запустіть 10 клієнтів з інкрементом по 10_000 на кожному з них. Виміряйте час виконання та перевірте чи кінцеве значення буде дорівнювати очікуваному – 100K

1.4 Вимоги до оформлення протоколу

Завдання здається особисто без протоколу, або надсилається протокол який має містити:

- команди та результати їх виконання у вигляді скріншотів
- аналіз отриманих результатів

2 Хід роботи

2.1 Кластер cassandra

Кластер запускається на Docker з наступним docker-compose:

Лістинг 1: docker-compose.yml

```
services:
  cassandra1:
    image: cassandra:4.1
    container_name: cassandra1
    hostname: cassandra1
    networks:
      - cassandra-net
    ports:
```

```
      - "9042:9042"
    environment:

      CASSANDRA_CLUSTER_NAME: "TestCluster"
      CASSANDRA_SEEDS: "cassandra1"
      CASSANDRA_DC: "dc1"
      CASSANDRA_RACK: "rack1"
      CASSANDRA_ENDPOINT_SNITCH:
  GossipingPropertyFileSnitch
      CASSANDRA_NUM_TOKENS: 256
      CASSANDRA_RPC_ADDRESS: 0.0.0.0
      CASSANDRA_BROADCAST_RPC_ADDRESS: "cassandra1"

    volumes:
      - cassandra1-data:/var/lib/cassandra
    restart: unless-stopped

  cassandra2:
    image: cassandra:4.1
    container_name: cassandra2
    hostname: cassandra2
    networks:
      - cassandra-net
    ports:
    - "9043:9042"
    depends_on:
      - cassandra1
    environment:
      CASSANDRA_CLUSTER_NAME: "TestCluster"
      CASSANDRA_SEEDS: "cassandra1"
      CASSANDRA_DC: "dc1"
      CASSANDRA_RACK: "rack1"
      CASSANDRA_ENDPOINT_SNITCH:
  GossipingPropertyFileSnitch
      CASSANDRA_NUM_TOKENS: 256
      CASSANDRA_RPC_ADDRESS: 0.0.0.0
      CASSANDRA_BROADCAST_RPC_ADDRESS: "cassandra1"
    volumes:
      - cassandra2-data:/var/lib/cassandra
    restart: unless-stopped

  cassandra3:
    image: cassandra:4.1
    container_name: cassandra3
    hostname: cassandra3
    networks:
```

```
      - cassandra-net
    ports:
    - "9044:9042"
    depends_on:
      - cassandra1
    environment:
      CASSANDRA_CLUSTER_NAME: "TestCluster"
      CASSANDRA_SEEDS: "cassandra1"
      CASSANDRA_DC: "dc1"
      CASSANDRA_RACK: "rack1"
      CASSANDRA_ENDPOINT_SNITCH:
   GossipingPropertyFileSnitch
      CASSANDRA_NUM_TOKENS: 256
      CASSANDRA_RPC_ADDRESS: 0.0.0.0
      CASSANDRA_BROADCAST_RPC_ADDRESS: "cassandra1"
    volumes:
      - cassandra3-data:/var/lib/cassandra
    restart: unless-stopped

networks:
  cassandra-net:
    driver: bridge

volumes:
  cassandra1-data:
  cassandra2-data:
  cassandra3-data:
```

2.2 Частина 1. Робота зі структурами даних у Cassandra

Для повторюванності виконання запитів до Cassandra, запити виконуються pyton скриптом. Також, це дозволяє створити генератор даних для таблиць. Окрім вказаного в завданні функціоналу, скрипт видаляє keyspace, якщо він вже існує та генерує тестові дані.

Лістинг 2: part_1.py

```python
from cassandra.cluster import Cluster
from cassandra.query import SimpleStatement
import string
import numpy as np
import random
from datetime import datetime, timezone

KEYSPACE = "lab_keyspace"

try:
    print("Trying to reach Cassandra node 1")
    cluster = Cluster(contact_points=["127.0.0.1"], port=9042)
```

```
    session = cluster.connect()
except:
    try:
        print("Trying to reach Cassandra node 2")
        cluster = Cluster(contact_points=["127.0.0.1"], port=9043)
        session = cluster.connect()
    except:
        try:
            print("Trying to reach Cassandra node 3")
            cluster = Cluster(contact_points=["127.0.0.1"], port
    =9044)
            session = cluster.connect()
        except:
            raise Exception("Cannot find cluster")

session.execute(f"DROP KEYSPACE IF EXISTS {KEYSPACE};", timeout
    =30.0)
print(f"Keyspace '{KEYSPACE}' dropped")


session.execute(f"""
    CREATE KEYSPACE {KEYSPACE}
    WITH replication = {{
        'class': 'SimpleStrategy',
        'replication_factor': 3
    }};
""", timeout=30.0)
print(f"Keyspace '{KEYSPACE}' created")

session.set_keyspace(KEYSPACE)


session.execute("""
    CREATE TABLE items (
        category text,
        item_id uuid,
        name text,
        price decimal,
        producer text,
        attributes map<text, text>,
        PRIMARY KEY ((category), price, item_id)
    ) WITH CLUSTERING ORDER BY (price ASC, item_id ASC);
""", timeout=30.0)
print("Table 'items' created")

session.execute("""
    CREATE INDEX items_attributes_idx
    ON items (ENTRIES(attributes));
""", timeout=30.0)
print("Index on attributes created")

categories = ["steel", "aluminium", "concrete",
              "composite", "wood", "plastic", "other"]
def get_category():
    return categories[int(random.random()*len(categories))]
producers = ["IPT Inc.", "FizTech co.", "KPI and others",
               "Siko R Sky", "Mann. co.", "Aperture laboratories",
```

```
                "Walter White chemistry site", "Jack Horner
    Industrial pies"]
def get_producer():
    return producers[int(random.random()*len(producers))]

def get_name():
    name = ""
    name += string.ascii_uppercase[int(random.random()*len(string.
    ascii_uppercase))]
    for i in range(int(np.random.uniform(3, 16))):
        name += string.ascii_lowercase[int(random.random()*len(
    string.ascii_lowercase))]
    return name

def get_price():
    return round(np.random.uniform(0.5, 10**4), 2)

attributes = ["weight", "length", "width", "height"]

def get_atributes():
    att = dict()
    for i in attributes:
        if random.random() > 0.5:
            att[i] = str(round(np.random.uniform(1,1001),2))
    return att

for i in range(100):
    session.execute(
    "INSERT INTO items (category, item_id, name, price, producer,
    attributes) VALUES (%s, uuid(), %s, %s, %s, %s)",
    (get_category(), get_name(), get_price(), get_producer(),
    get_atributes(),), timeout=30.0)
print("Test data into items inserted")


session.execute("""
    CREATE TABLE orders (
        customer_name text,
        order_date timestamp,
        order_id uuid,
        item_ids list<uuid>,
        total_price decimal,
        PRIMARY KEY ((customer_name), order_date, order_id)
    ) WITH CLUSTERING ORDER BY (order_date DESC);
""", timeout=30.0)
print("Table 'orders' created")

customers = ["RED", "BLU", "Baldwin locomotive company",
             "Depressing coal mines",
             "Slightly less depressing coal mines",
             "Construction company that constructs offices for
    construction companies",
             "O.W.C.A",
             "Doofenshmirtz Evil Inc.",
             "This company does not build secret goverment progects
     Inc."]

```

```
118 def get_customer():
119     return customers[int(random.random()*len(customers))]
120 
121 def get_order_date(min = 2015, max = 2025):
122     start = datetime(min, 1, 1, 0, 0, 0, tzinfo=timezone.utc)
123     end = datetime(max, 12, 31, 23, 59, 59, tzinfo=timezone.utc)
124 
125     start_ts = int(start.timestamp())
126     end_ts = int(end.timestamp())
127 
128     random_ts = random.randint(start_ts, end_ts)
129 
130     return datetime.fromtimestamp(random_ts, tz=timezone.utc)
131 
132 def get_all_items_ids_and_prices():
133     ids = []
134     prices = []
135     for category in categories:
136         rows = session.execute("SELECT item_id, price FROM items
        WHERE category=%s", (category,), timeout=30.0)
137         for row in rows:
138             ids.append(row.item_id)
139             prices.append(row.price)
140     return ids, prices
141 
142 ids, prices = get_all_items_ids_and_prices()
143 
144 def get_price(uuid):
145     try:
146         i = ids.index(uuid)
147     except:
148         return False
149     return prices[i]
150 
151 def get_items_and_price(ids):
152     items = np.random.choice(ids, size=int(np.random.uniform(1, 16)
        ), replace=False).tolist()
153     price = sum([get_price(uuid) for uuid in items])
154     return items, price
155 
156 for i in range(100):
157     items, price = get_items_and_price(ids)
158     session.execute(
159     "INSERT INTO orders (customer_name, order_date, order_id,
        item_ids, total_price) VALUES (%s, %s, uuid(), %s, %s)",
160     (get_customer(), get_order_date(), items, price,), timeout
        =30.0)
161 print("Test data into orders inserted")
162 
163 
164 input("Waiting for manual DESCRIBE")
165 
166 print()
167 print("Category with sorting by price:")
168 rows = session.execute("SELECT * FROM items WHERE category=%s", ('
        aluminium',), timeout=30.0)
169 for row in rows:
```

```
    print(row)


print()
try:
    session.execute("CREATE INDEX name_index ON items(name)")
except:
    print("Failed to create index on name (May already exist)")
print("Category with name constrains:")
selected_name = input("Input name: ")
rows = session.execute("SELECT * FROM items WHERE category=%s AND
    name = %s", ('aluminium',selected_name,), timeout=30.0)
for row in rows:
    print(row)

print()
print("Category with price constrains (1000,5000):")
rows = session.execute("SELECT * FROM items WHERE category=%s AND
    price>1000 AND price<5000", ('aluminium',), timeout=30.0)
for row in rows:
    print(row)

print()
try:
    session.execute("CREATE INDEX producer_index ON items(producer)
    ", timeout=30.0)
except:
    print("Failed to create index on producer (May already exist)")
print("Category with price (1000,9000) and producer constrains :")


for producer in producers:
    rows = session.execute("SELECT * FROM items WHERE category=%s
    AND price>1000 AND price<9000 AND producer = %s", ('aluminium',
    producer,), timeout=30.0)
    producer_found = False
    for row in rows:
        producer_found = True
        print(row)
    if producer_found:
        break


print()
print("Orders for RED with sorting by date:")
rows = session.execute("SELECT * FROM orders WHERE customer_name=%s
    ", ('RED',), timeout=30.0)
for row in rows:
    print(row)

print()
print("All orders total price for every customer:")
rows = session.execute("SELECT DISTINCT customer_name FROM orders",
     timeout=30.0)
customer_names = [row.customer_name for row in rows]
for customer in customer_names:
```

```
    rows = session.execute("SELECT total_price FROM orders WHERE
    customer_name=%s", (customer,), timeout=30.0)
    total = sum([row.total_price for row in rows])
    print(customer, total)

print()
print("WRITETIME:")
rows = session.execute("""SELECT order_id, customer_name, WRITETIME
    (total_price) AS price_write_time
        FROM orders WHERE customer_name = %s""", ('RED',), timeout
    =30.0)
for row in rows:
    print(row)

cluster.shutdown()

input("End")
```

DESCRIBE:

```
Windows PowerShell
[cqlsh 6.1.0 | Cassandra 4.1.10 | CQL spec 3.4.6 | Native protocol v5]
Use HELP for help.
cqlsh> USE lab_keyspace;
cqlsh:lab_keyspace> DESCRIBE TABLES;

items  orders

cqlsh:lab_keyspace> DESCRIBE TABLE items;

CREATE TABLE lab_keyspace.items (
    category text,
    price decimal,
    item_id uuid,
    name text,
    producer text,
    attributes map<text, text>,
    PRIMARY KEY (category, price, item_id)
) WITH CLUSTERING ORDER BY (price ASC, item_id ASC)
    AND additional_write_policy = '99p'
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND cdc = false
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND memtable = 'default'
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    AND extensions = {}
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair = 'BLOCKING'
    AND speculative_retry = '99p';

CREATE INDEX items_attributes_idx ON lab_keyspace.items (entries(attributes));
cqlsh:lab_keyspace> DESCRIBE TABLE orders;

CREATE TABLE lab_keyspace.orders (
    customer_name text,
    order_date timestamp,
    order_id uuid,
    total_price decimal,
    item_ids list<uuid>,
    PRIMARY KEY (customer_name, order_date, order_id)
) WITH CLUSTERING ORDER BY (order_date DESC, order_id ASC)
    AND additional_write_policy = '99p'
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND cdc = false
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND memtable = 'default'
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    AND extensions = {}
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair = 'BLOCKING'
    AND speculative_retry = '99p';
cqlsh:lab_keyspace>
```

Рис. 1: Ініціалізація кластеру

Результат роботи коду (перший рядок виводу SELECT з ентерами, для демонстрації всіх полів. Інші рядки – виходять за межі сторінки і обрізаються, але так цей вивід хоча б можна дочитати до останнього рядка):

```
Trying to reach Cassandra node 1
Keyspace 'lab_keyspace' dropped
Keyspace 'lab_keyspace' created
Table 'items' created
Index on attributes created
Test data into items inserted
Table 'orders' created
Test data into orders inserted
Waiting for manual DESCRIBE

Category with sorting by price:
Row(category='aluminium', price=Decimal('124.95'),
item_id=UUID('dad7d4cf-9aa5-477b-a374-229e0b4a5835'),
attributes=OrderedMapSerializedKey([('height', '310.11'),
('length', '373.82')]), name='Kpfrugtqsokbl',
producer='FizTech co.')
Row(category='aluminium', price=Decimal('260.15'), item_id=UUID('680df2e5-e7b9-4210-8ad9-d23de98eb076'
Row(category='aluminium', price=Decimal('413.91'), item_id=UUID('4b9e9cc3-90d1-4779-9d5e-5584e05c8829'
Row(category='aluminium', price=Decimal('511.15'), item_id=UUID('f45ec388-6a01-4295-b5ca-4a4d341ec3b1'
Row(category='aluminium', price=Decimal('523.68'), item_id=UUID('b928eef1-0595-432d-a941-c11aea715254'
Row(category='aluminium', price=Decimal('809.32'), item_id=UUID('69ab1b67-d537-4ce8-83fa-e923fa51b01c'
Row(category='aluminium', price=Decimal('1623.66'), item_id=UUID('0d67b2b9-f434-4020-8d02-9be70ca4ce97
Row(category='aluminium', price=Decimal('2709.36'), item_id=UUID('5c09a956-c6c7-4afc-b645-fe971c4e2f0f
Row(category='aluminium', price=Decimal('3051.55'), item_id=UUID('7dfc937d-3b64-43fb-8d0e-ec91b0fe540c
Row(category='aluminium', price=Decimal('4355.98'), item_id=UUID('c14a532c-844e-43b4-8da8-498d7e7829a6
Row(category='aluminium', price=Decimal('4611.29'), item_id=UUID('c20486a5-caf6-47cc-a24f-67e866978646
Row(category='aluminium', price=Decimal('4622.4'), item_id=UUID('e561a306-2fd2-44ae-9f7e-46d11eb60a84'
Row(category='aluminium', price=Decimal('5614.39'), item_id=UUID('1f5e14ec-7701-4ef1-8cf8-d501f22eaf7b
Row(category='aluminium', price=Decimal('8239.14'), item_id=UUID('f8da65f4-a7f5-4cdb-89c1-4530032ffc58
Row(category='aluminium', price=Decimal('8831.77'), item_id=UUID('11bb7583-312e-4346-94ef-e39619a2142b
Row(category='aluminium', price=Decimal('9439.91'), item_id=UUID('22fbcf85-fae0-4e78-b98d-c4ed7f6cf7a5
Row(category='aluminium', price=Decimal('9582.46'), item_id=UUID('7f24a4e7-c70a-45c2-9211-250c7f387d1a

Category with name constrains:
Input name: Mybye
Row(category='aluminium', price=Decimal('4355.98'),
item_id=UUID('c14a532c-844e-43b4-8da8-498d7e7829a6'),
attributes=None, name='Mybye',
producer='Aperture laboratories')

Category with price constrains (1000,5000):
Row(category='aluminium', price=Decimal('1623.66'),
item_id=UUID('0d67b2b9-f434-4020-8d02-9be70ca4ce97'),
attributes=OrderedMapSerializedKey([('height', '2.46'), ('weight', '559.62')]),
name='Tnrxnsp', producer='Siko R Sky')
Row(category='aluminium', price=Decimal('2709.36'), item_id=UUID('5c09a956-c6c7-4afc-b645-fe971c4e2f0f
```

```
Row(category='aluminium', price=Decimal('3051.55'), item_id=UUID('7dfc937d-3b64-43fb-8d0e-ec91b0fe540c
Row(category='aluminium', price=Decimal('4355.98'), item_id=UUID('c14a532c-844e-43b4-8da8-498d7e7829a6
Row(category='aluminium', price=Decimal('4611.29'), item_id=UUID('c20486a5-caf6-47cc-a24f-67e8669786460
Row(category='aluminium', price=Decimal('4622.4'), item_id=UUID('e561a306-2fd2-44ae-9f7e-46d11eb60a84'

Category with price (1000,9000) and producer constrains :
Row(category='aluminium', price=Decimal('4611.29'), item_id=UUID('c20486a5-caf6-47cc-a24f-67e8669786460
Row(category='aluminium', price=Decimal('4622.4'), item_id=UUID('e561a306-2fd2-44ae-9f7e-46d11eb60a84'

Orders for RED with sorting by date:
Row(customer_name='RED',
order_date=datetime.datetime(2023, 8, 7, 22, 48, 14),
order_id=UUID('cd9139a1-0775-4968-90e8-b3a607df2b8a'),
item_ids=[UUID('11bb7583-312e-4346-94ef-e39619a2142b'),
UUID('0d67b2b9-f434-4020-8d02-9be70ca4ce97'),
UUID('0061f320-9b73-455b-86ee-1a1d61f16428'),
UUID('81916927-d302-4bbb-a0a2-c631d6b40970'),
UUID('45d21142-be36-45e6-b808-eef380a7ead4'),
UUID('79ea5061-07cb-4fbd-99e8-f9914c97b388'),
UUID('866ed99c-f551-4c41-8589-686c34f29e88'),
UUID('1f5e14ec-7701-4ef1-8cf8-d501f22eaf7b'),
UUID('e1a24dac-91fd-4da0-8015-ead385b01982'),
UUID('c2abed94-ba5e-47bb-a676-8e1780faca58')],
total_price=Decimal('58830.08'))
Row(customer_name='RED', order_date=datetime.datetime(2023, 1, 27, 8, 29), order_id=UUID('6402adbe-632
Row(customer_name='RED', order_date=datetime.datetime(2022, 3, 4, 22, 46, 44), order_id=UUID('54d4c68b
Row(customer_name='RED', order_date=datetime.datetime(2022, 1, 8, 4, 29, 50), order_id=UUID('78331eab-
Row(customer_name='RED', order_date=datetime.datetime(2020, 4, 7, 0, 18, 37), order_id=UUID('e79fecdd-
Row(customer_name='RED', order_date=datetime.datetime(2019, 4, 1, 22, 49, 17), order_id=UUID('1c8ec1b0
Row(customer_name='RED', order_date=datetime.datetime(2018, 7, 27, 18, 42, 54), order_id=UUID('72ec133
Row(customer_name='RED', order_date=datetime.datetime(2016, 12, 1, 7, 32, 45), order_id=UUID('c2fe058b
Row(customer_name='RED', order_date=datetime.datetime(2016, 10, 21, 3, 57, 14), order_id=UUID('58cc5fc
Row(customer_name='RED', order_date=datetime.datetime(2016, 1, 31, 22, 18, 53), order_id=UUID('327b8ee
Row(customer_name='RED', order_date=datetime.datetime(2015, 11, 15, 13, 45, 19), order_id=UUID('2259b3
Row(customer_name='RED', order_date=datetime.datetime(2015, 10, 21, 15, 14, 40), order_id=UUID('8c15e8
Row(customer_name='RED', order_date=datetime.datetime(2015, 7, 11, 9, 21, 50), order_id=UUID('c53710a7

All orders total price for every customer:
Doofenshmirtz Evil Inc. 327056.09
O.W.C.A 514832.90
This company does not build secret goverment progects Inc. 389293.64
Baldwin locomotive company 381497.00
Construction company that constructs offices for construction companies 607143.54
RED 694261.42
BLU 284720.39
Depressing coal mines 337731.01
Slightly less depressing coal mines 509513.92

WRITETIME:
Row(order_id=UUID('cd9139a1-0775-4968-90e8-b3a607df2b8a'),
```

```
customer_name='RED', price_write_time=1766000896582849)
Row(order_id=UUID('6402adbe-632f-4c58-b1e4-782e7f7a79c6'), customer_name='RED', price_write_time=17660
Row(order_id=UUID('54d4c68b-d05b-4157-b972-6e9c79a8f695'), customer_name='RED', price_write_time=17660
Row(order_id=UUID('78331eab-bf10-4453-a52e-fb6fd4b347f2'), customer_name='RED', price_write_time=17660
Row(order_id=UUID('e79fecdd-7865-467d-b579-8e611b56fadd'), customer_name='RED', price_write_time=17660
Row(order_id=UUID('1c8ec1b0-f9c6-4a1d-adad-9bca1a2fd7f4'), customer_name='RED', price_write_time=17660
Row(order_id=UUID('72ec1337-c11b-461d-945d-7192792c7506'), customer_name='RED', price_write_time=17660
Row(order_id=UUID('c2fe058b-2e32-4ca3-a305-574dda9aa561'), customer_name='RED', price_write_time=17660
Row(order_id=UUID('58cc5fc4-b486-4ff1-bf46-fbe1a24e9da6'), customer_name='RED', price_write_time=17660
Row(order_id=UUID('327b8ee0-ecb0-4ba5-9556-ee7e51bca6c7'), customer_name='RED', price_write_time=17660
Row(order_id=UUID('2259b3f8-e80b-4fb1-b607-6e2b925403ae'), customer_name='RED', price_write_time=17660
Row(order_id=UUID('8c15e88e-e524-4882-80fe-21301e734316'), customer_name='RED', price_write_time=17660
Row(order_id=UUID('c53710a7-c43d-4d66-9ccb-d61b41294bfc'), customer_name='RED', price_write_time=17660
End
```

2.3 Частина 2. Налаштування реплікації у Cassandra

3 Результати

Варіант	Час роботи	Пропускна здатність	Значення лічильника
1	26.37	3792.65	100к
majority	76.2	1312.37	100к
1 та падіння	23.05	4338.28	100к
majority та падіння	59.51	1680.42	100k