LAB - 06

Neo4j

Objective: Students will learn how to structure data in a graph format, use Cypher queries for data manipulation, and explore real-world use cases of graph databases. The objective of this assignment is to give students practical exposure to graph databases by building a social network model using Neo4j.

ASSIGNMENT:

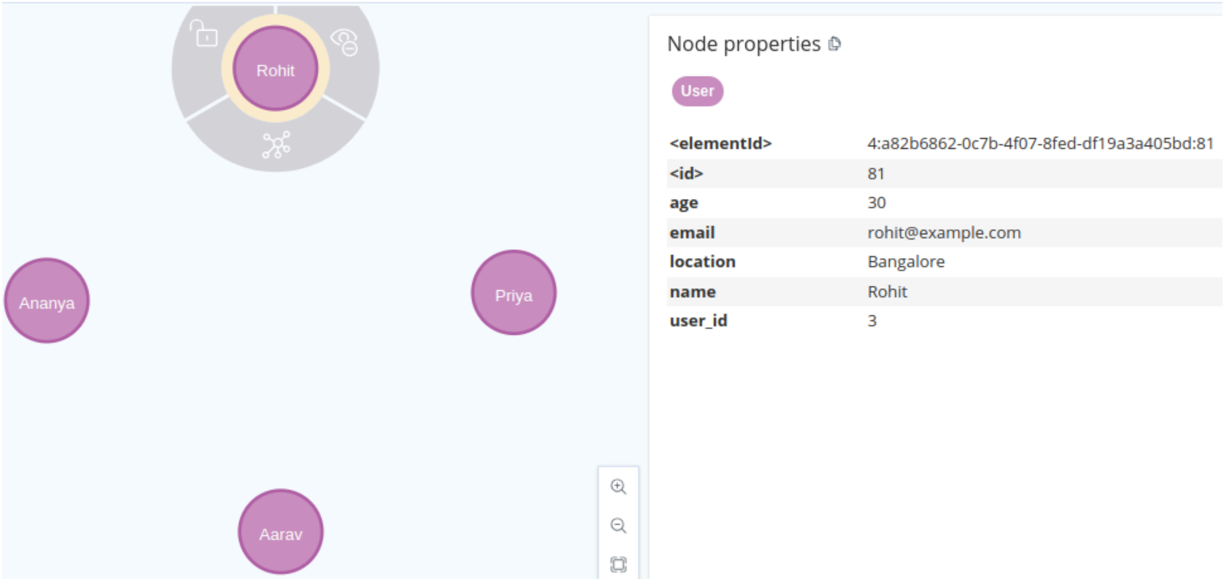
Design the database for Social networks.

1. Nodes:

● User Node: Represents the users of the social network.

○ Properties: user_id, name, age, location, email

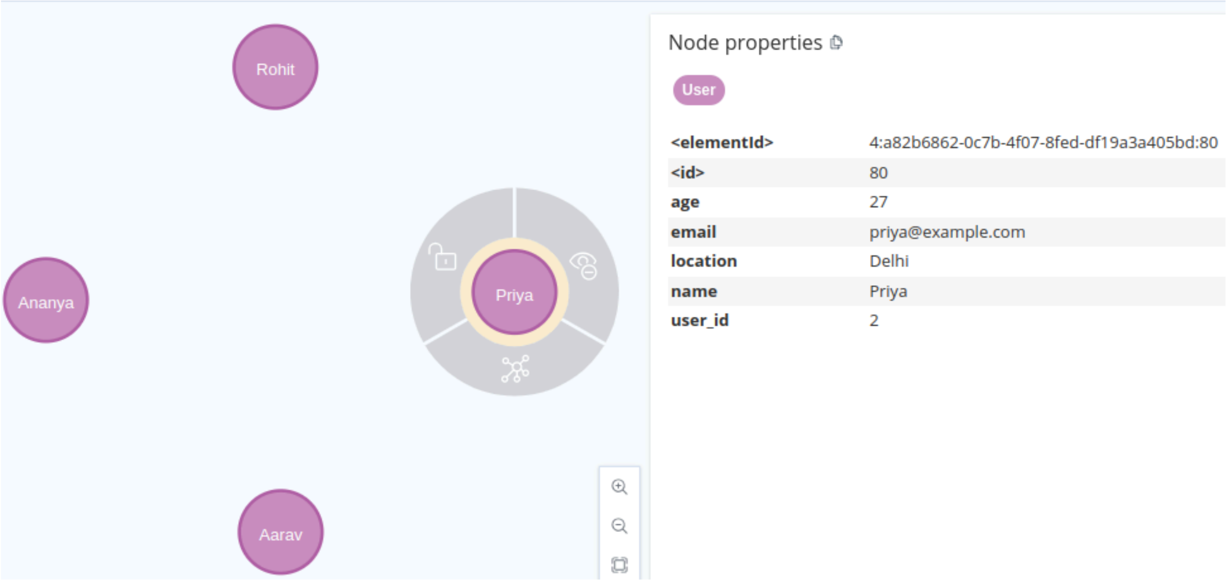
```
CREATE (u1:User {user_id: 1, name: "Aarav", age: 25, location: "Mumbai", email:
"aarav@example.com"})
CREATE (u2:User {user_id: 2, name: "Priya", age: 27, location: "Delhi", email:
"priya@example.com"})
CREATE (u3:User {user_id: 3, name: "Rohit", age: 30, location: "Bangalore", email:
"rohit@example.com"})
CREATE (u4:User {user_id: 4, name: "Ananya", age: 28, location: "Mumbai", email:
"ananya@example.com"})
RETURN u1, u2, u3, u4;
```



Node properties

User

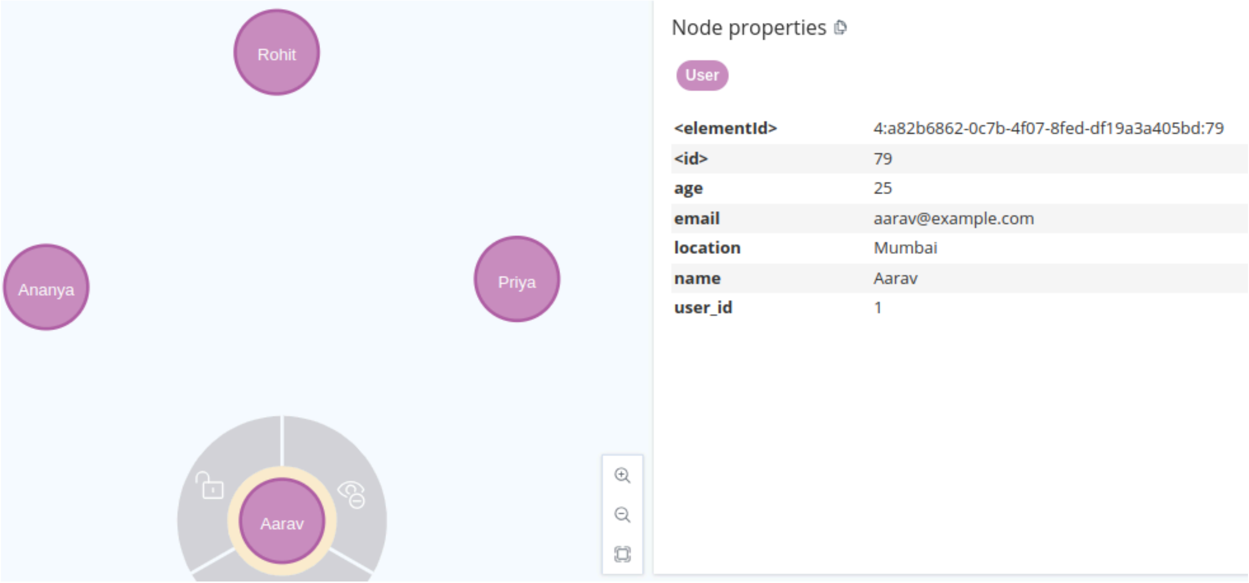
<elementId>	4:a82b6862-0c7b-4f07-8fed-df19a3a405bd:81
<id>	81
age	30
email	rohit@example.com
location	Bangalore
name	Rohit
user_id	3



Node properties

User

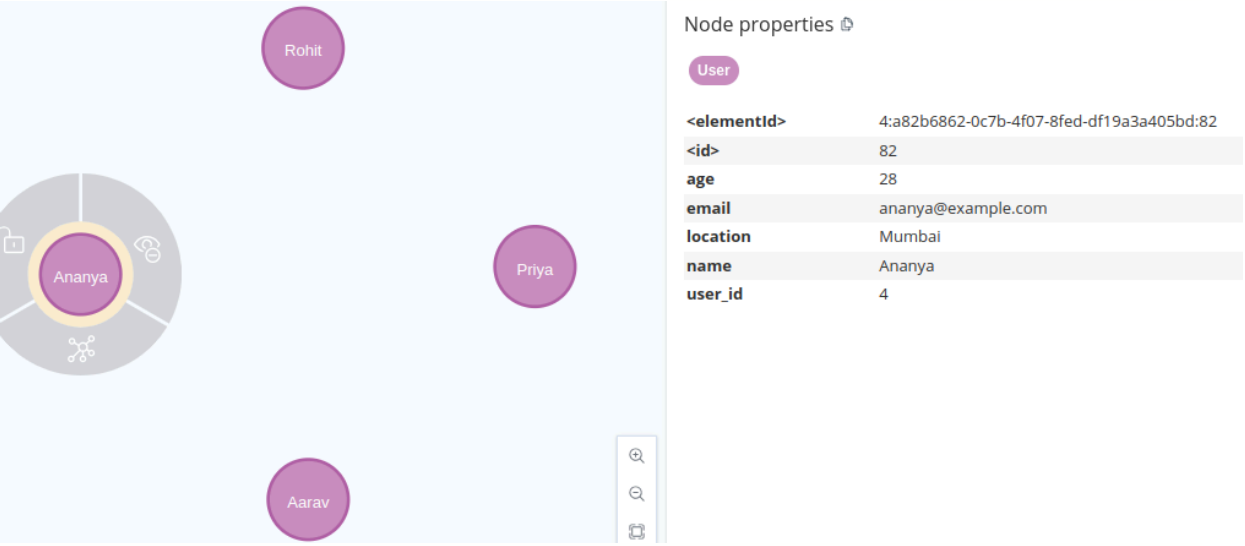
<elementId>	4:a82b6862-0c7b-4f07-8fed-df19a3a405bd:80
<id>	80
age	27
email	priya@example.com
location	Delhi
name	Priya
user_id	2



Node properties

User

<elementId>	4:a82b6862-0c7b-4f07-8fed-df19a3a405bd:79
<id>	79
age	25
email	aarav@example.com
location	Mumbai
name	Aarav
user_id	1



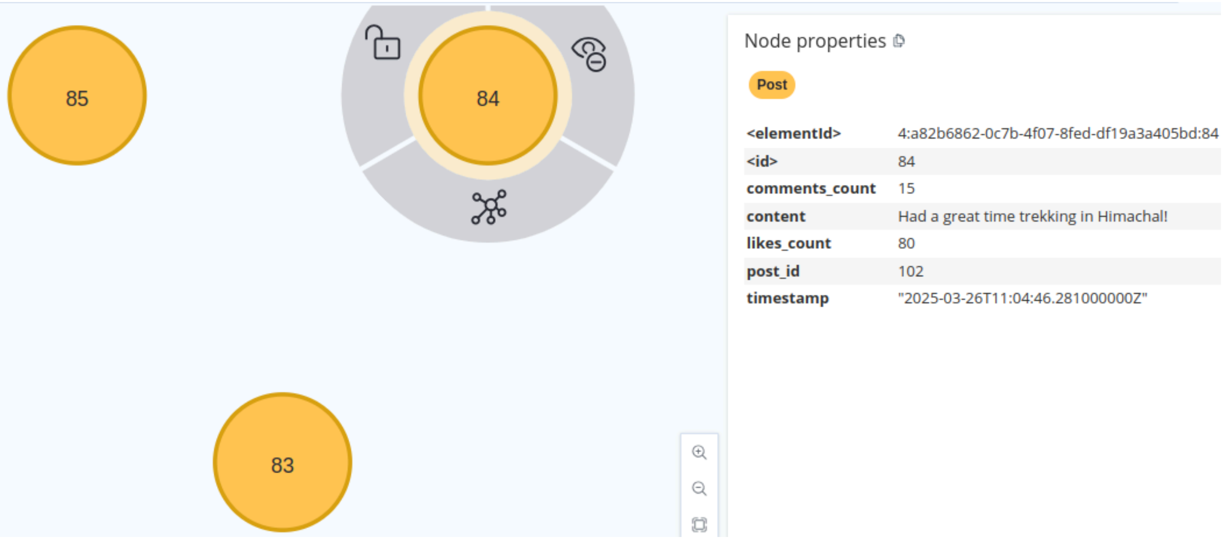
Node properties

User

<elementId>	4:a82b6862-0c7b-4f07-8fed-df19a3a405bd:82
<id>	82
age	28
email	ananya@example.com
location	Mumbai
name	Ananya
user_id	4

- **Post Node: Represents a post created by a user.**
  - **Properties: post_id, content, timestamp, likes_count, comments_count**

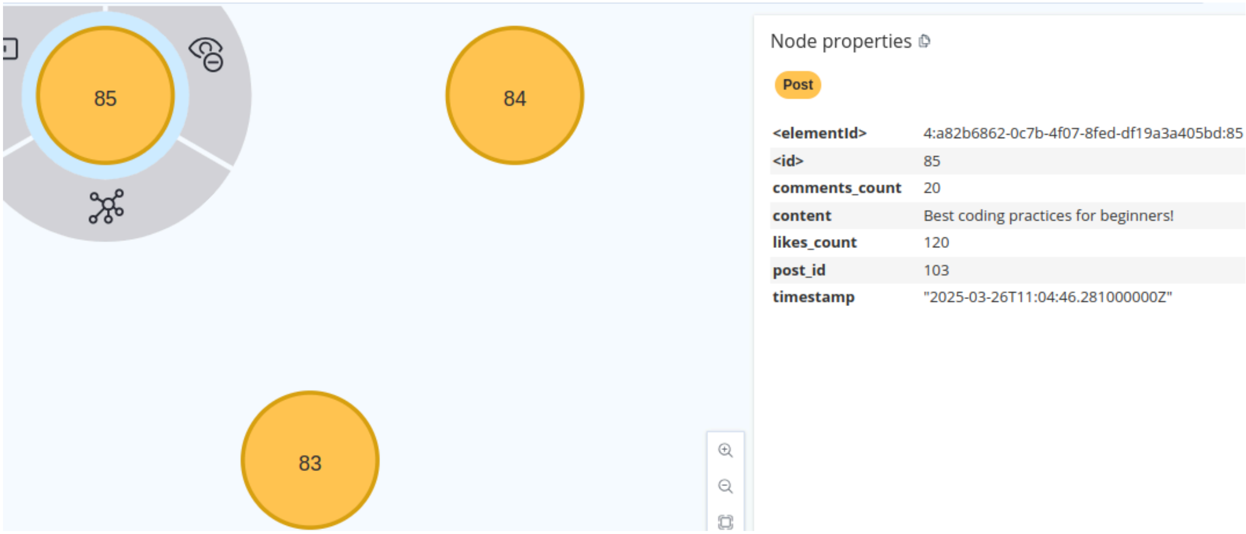
```
CREATE (p1:Post {post_id: 101, content: "Exploring Mumbai street food!", timestamp:
datetime(), likes_count: 50, comments_count: 8})
CREATE (p2:Post {post_id: 102, content: "Had a great time trekking in Himachal!",
timestamp: datetime(), likes_count: 80, comments_count: 15})
CREATE (p3:Post {post_id: 103, content: "Best coding practices for beginners!", timestamp:
datetime(), likes_count: 120, comments_count: 20})
RETURN p1, p2, p3;
```



Node properties

Post

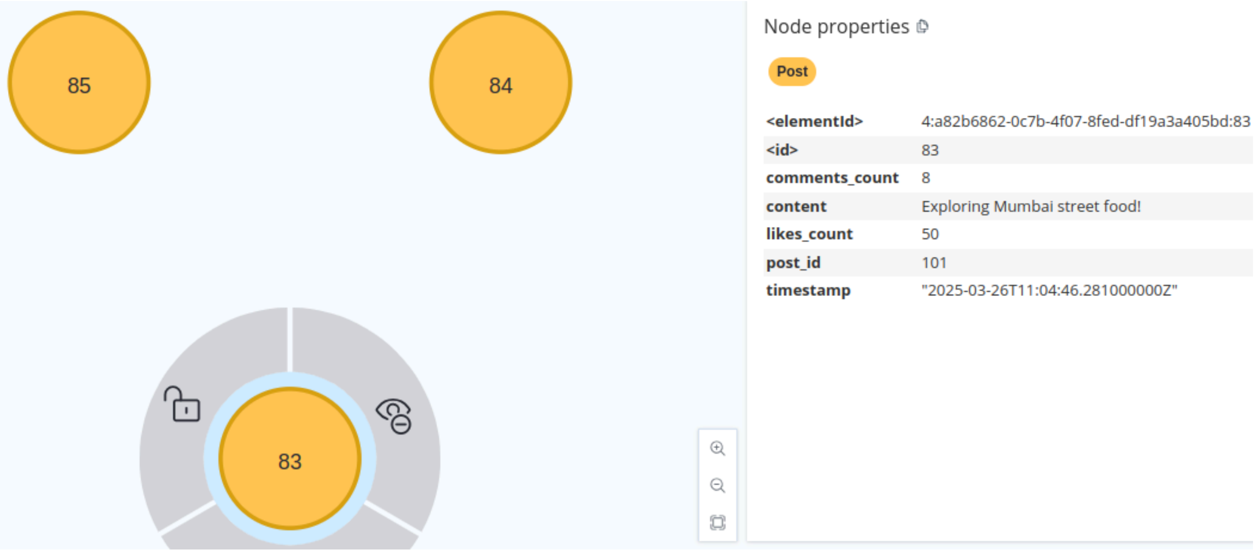
<elementId>	4:a82b6862-0c7b-4f07-8fed-df19a3a405bd:84
<id>	84
comments_count	15
content	Had a great time trekking in Himachal!
likes_count	80
post_id	102
timestamp	"2025-03-26T11:04:46.281000000Z"



Node properties

Post

<elementId>	4:a82b6862-0c7b-4f07-8fed-df19a3a405bd:85
<id>	85
comments_count	20
content	Best coding practices for beginners!
likes_count	120
post_id	103
timestamp	"2025-03-26T11:04:46.281000000Z"



Node properties

Post

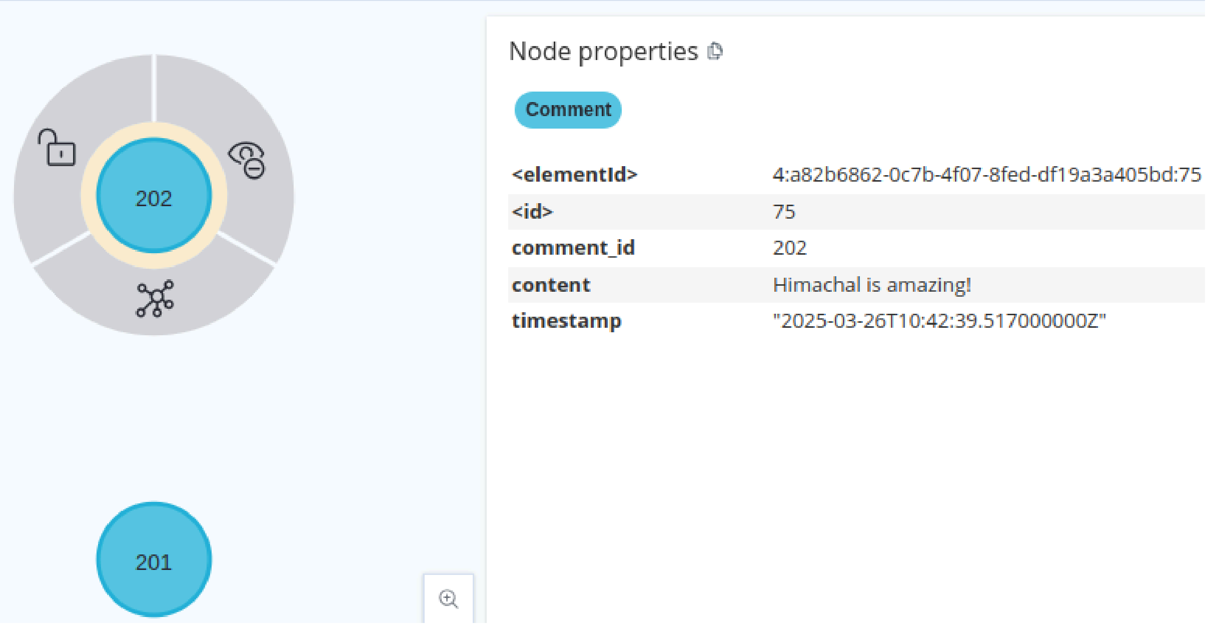
<elementId>	4:a82b6862-0c7b-4f07-8fed-df19a3a405bd:83
<id>	83
comments_count	8
content	Exploring Mumbai street food!
likes_count	50
post_id	101
timestamp	"2025-03-26T11:04:46.281000000Z"

- **Comment Node: Represents a comment on a post.**

○ **Properties: comment_id, content, timestamp**

```
CREATE (c1:Comment {comment_id: 201, content: "Looks delicious!", timestamp: datetime()})
CREATE (c2:Comment {comment_id: 202, content: "Himachal is amazing!", timestamp: datetime()})
RETURN c1, c2;
```

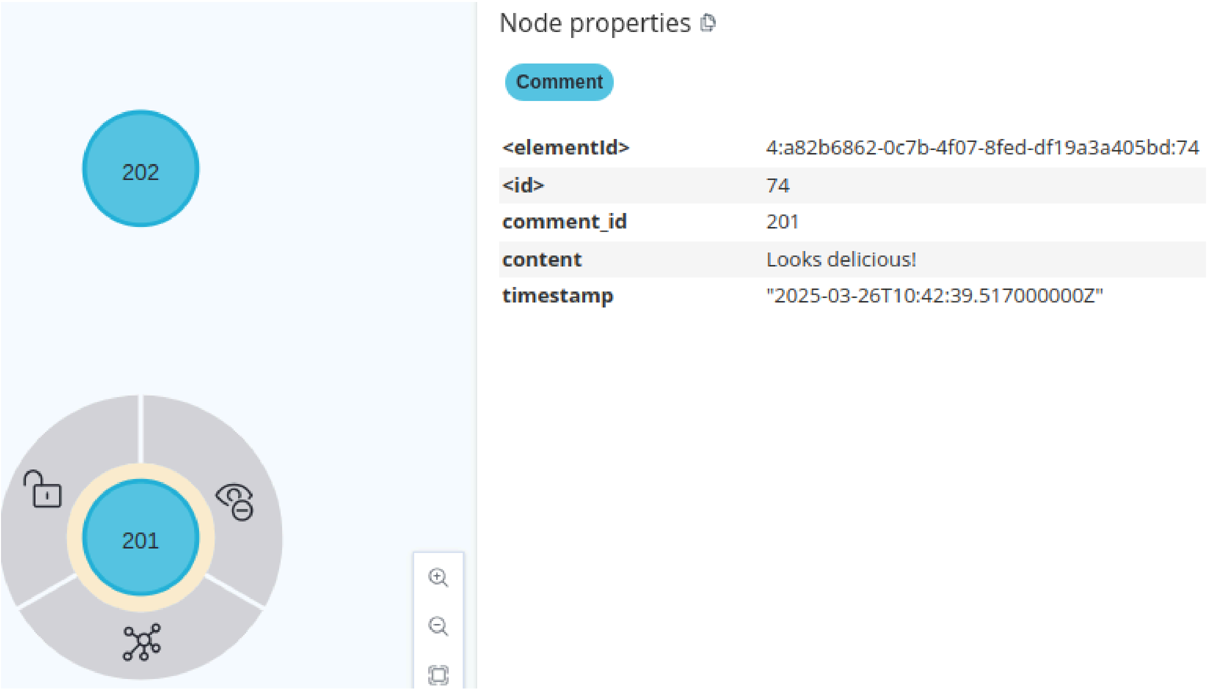
```
p: datetime()})
```



Node properties

Comment

<elementId>	4:a82b6862-0c7b-4f07-8fed-df19a3a405bd:75
<id>	75
comment_id	202
content	Himachal is amazing!
timestamp	"2025-03-26T10:42:39.517000000Z"



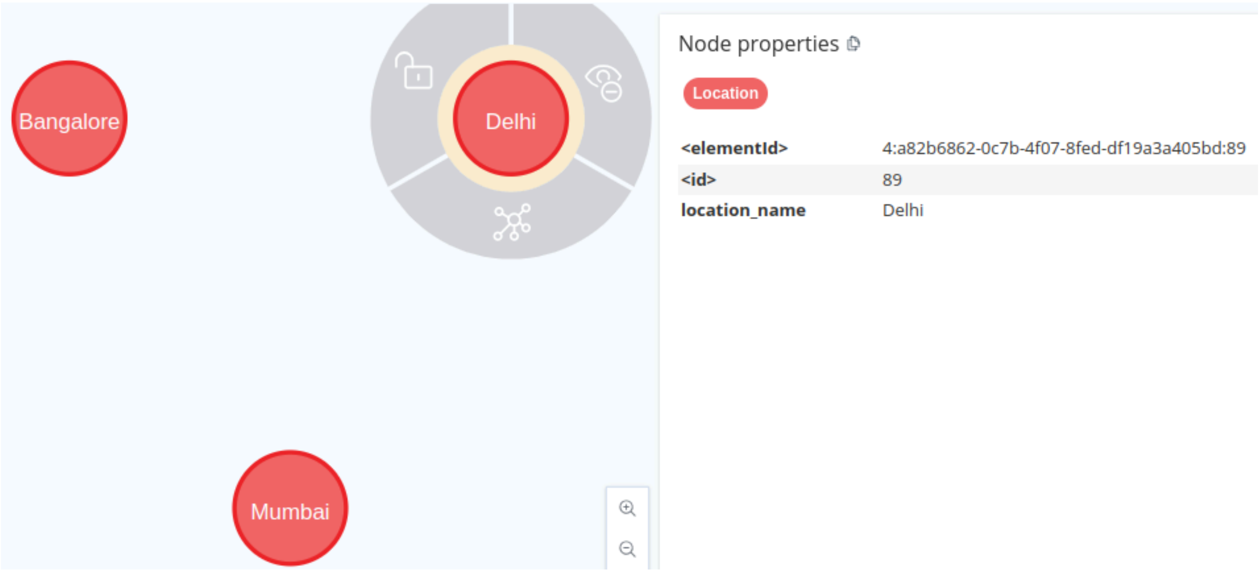
Node properties

Comment

<elementId>	4:a82b6862-0c7b-4f07-8fed-df19a3a405bd:74
<id>	74
comment_id	201
content	Looks delicious!
timestamp	"2025-03-26T10:42:39.517000000Z"

- **Location Node: (Optional) If you're going to query based on location.**
  - **Properties: location_name**

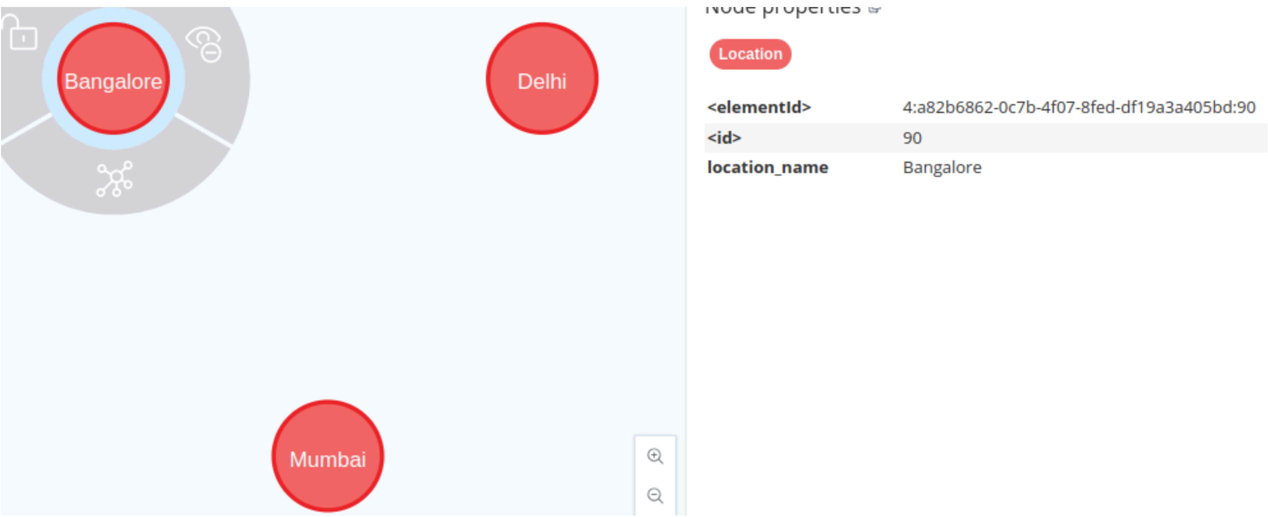
```
CREATE (l1:Location {location_name: "Mumbai"})
CREATE (l2:Location {location_name: "Delhi"})
CREATE (l3:Location {location_name: "Bangalore"})
RETURN l1, l2, l3;
```



Node properties

Location

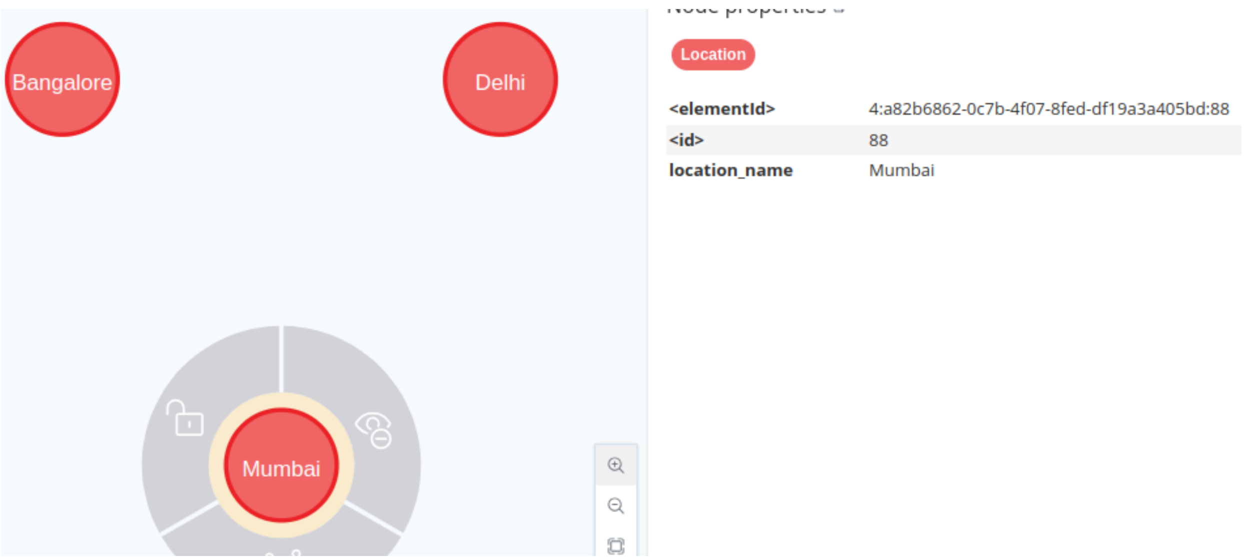
<elementId>	4:a82b6862-0c7b-4f07-8fed-df19a3a405bd:89
<id>	89
location_name	Delhi



Node properties

Location

<elementId>	4:a82b6862-0c7b-4f07-8fed-df19a3a405bd:90
<id>	90
location_name	Bangalore



Node properties

Location

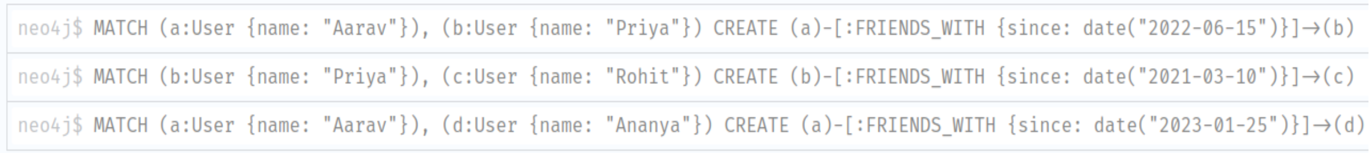
<elementId>	4:a82b6862-0c7b-4f07-8fed-df19a3a405bd:88
<id>	88
location_name	Mumbai

2. Relationships:

● FRIENDS_WITH: Represents the friendship relationship between two users.

○ Properties: since (the date when the friendship started)

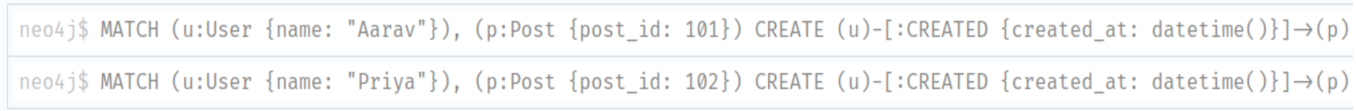
```
MATCH (a:User {name: "Aarav"}), (b:User {name: "Priya"})
CREATE (a)-[:FRIENDS_WITH {since: date("2022-06-15")}]->(b);

MATCH (b:User {name: "Priya"}), (c:User {name: "Rohit"})
CREATE (b)-[:FRIENDS_WITH {since: date("2021-03-10")}]->(c);

MATCH (a:User {name: "Aarav"}), (d:User {name: "Ananya"})
CREATE (a)-[:FRIENDS_WITH {since: date("2023-01-25")}]->(d);
```



```
neo4j$ MATCH (a:User {name: "Aarav"}), (b:User {name: "Priya"}) CREATE (a)-[:FRIENDS_WITH {since: date("2022-06-15")}]→(b)
neo4j$ MATCH (b:User {name: "Priya"}), (c:User {name: "Rohit"}) CREATE (b)-[:FRIENDS_WITH {since: date("2021-03-10")}]→(c)
neo4j$ MATCH (a:User {name: "Aarav"}), (d:User {name: "Ananya"}) CREATE (a)-[:FRIENDS_WITH {since: date("2023-01-25")}]→(d)
```

● CREATED: Represents a user creating a post.
○ Properties: created_at (timestamp when the post was created)

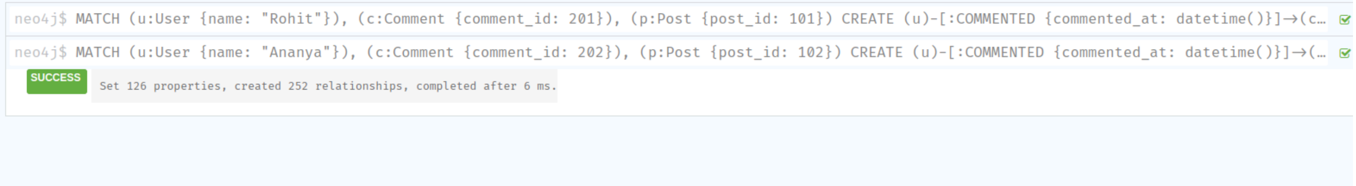
```
MATCH (u:User {name: "Aarav"}), (p:Post {post_id: 101})
CREATE (u)-[:CREATED {created_at: datetime()}]->(p);

MATCH (u:User {name: "Priya"}), (p:Post {post_id: 102})
CREATE (u)-[:CREATED {created_at: datetime()}]->(p);
```



```
neo4j$ MATCH (u:User {name: "Aarav"}), (p:Post {post_id: 101}) CREATE (u)-[:CREATED {created_at: datetime()}]→(p)
neo4j$ MATCH (u:User {name: "Priya"}), (p:Post {post_id: 102}) CREATE (u)-[:CREATED {created_at: datetime()}]→(p)
```

● COMMENTED: Represents a user commenting on a post.
○ Properties: commented_at (timestamp when the comment was made)

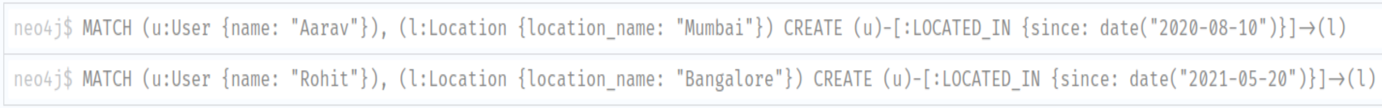
```
MATCH (u:User {name: "Rohit"}), (c:Comment {comment_id: 201}), (p:Post {post_id:
101})
CREATE (u)-[:COMMENTED {commented_at: datetime()}]->(c)-[:ON_POST]->(p);

MATCH (u:User {name: "Ananya"}), (c:Comment {comment_id: 202}), (p:Post {post_id:
102})
CREATE (u)-[:COMMENTED {commented_at: datetime()}]->(c)-[:ON_POST]->(p);
```



```
neo4j$ MATCH (u:User {name: "Rohit"}), (c:Comment {comment_id: 201}), (p:Post {post_id: 101}) CREATE (u)-[:COMMENTED {commented_at: datetime()}]→(c… ✓
neo4j$ MATCH (u:User {name: "Ananya"}), (c:Comment {comment_id: 202}), (p:Post {post_id: 102}) CREATE (u)-[:COMMENTED {commented_at: datetime()}]→(… ✓
SUCCESS  Set 126 properties, created 252 relationships, completed after 6 ms.
```

● LOCATED_IN: (Optional) Connects a user to a location.
○ Properties: since (timestamp when the user joined this location)

```
MATCH (u:User {name: "Aarav"}), (l:Location {location_name: "Mumbai"})
CREATE (u)-[:LOCATED_IN {since: date("2020-08-10")}]->(l);

MATCH (u:User {name: "Rohit"}), (l:Location {location_name: "Bangalore"})
CREATE (u)-[:LOCATED_IN {since: date("2021-05-20")}]->(l);
```

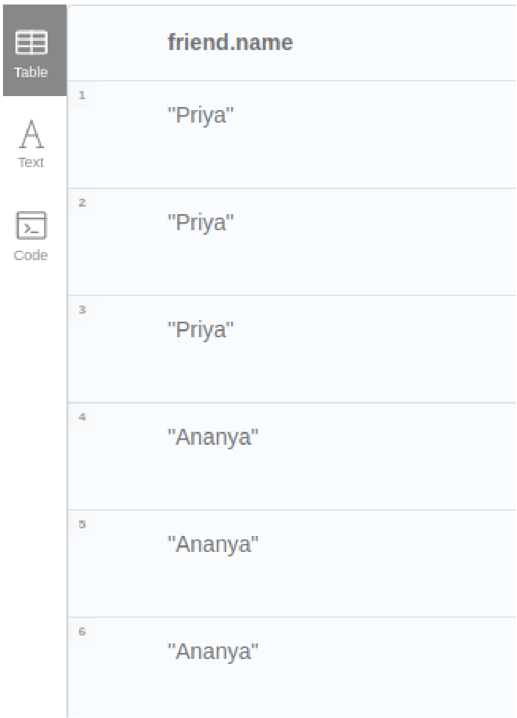


```
neo4j$ MATCH (u:User {name: "Aarav"}), (l:Location {location_name: "Mumbai"}) CREATE (u)-[:LOCATED_IN {since: date("2020-08-10")}]→(l)
neo4j$ MATCH (u:User {name: "Rohit"}), (l:Location {location_name: "Bangalore"}) CREATE (u)-[:LOCATED_IN {since: date("2021-05-20")}]→(l)
```

3. Queries:

a. Retrieve all friends of a specific user.

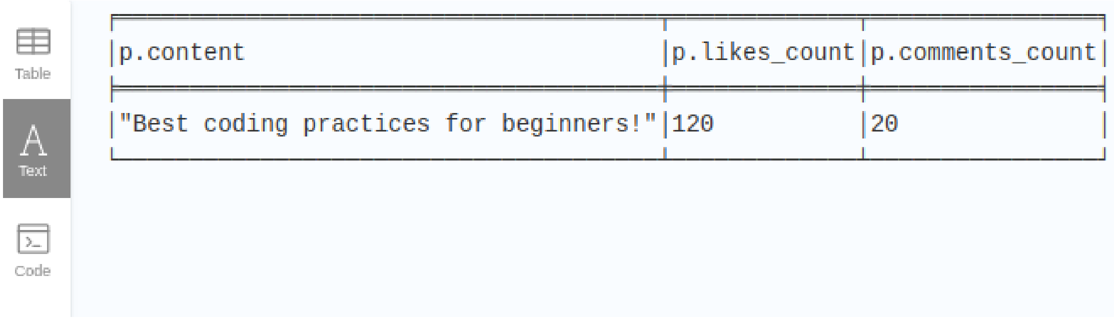
```
MATCH (:User {name: "Aarav"})-[:FRIENDS_WITH]-(friend)
RETURN friend.name;
```



	friend.name
1	"Priya"
2	"Priya"
3	"Priya"
4	"Ananya"
5	"Ananya"
6	"Ananya"

b. Find the most popular post (by the number of comments or likes).

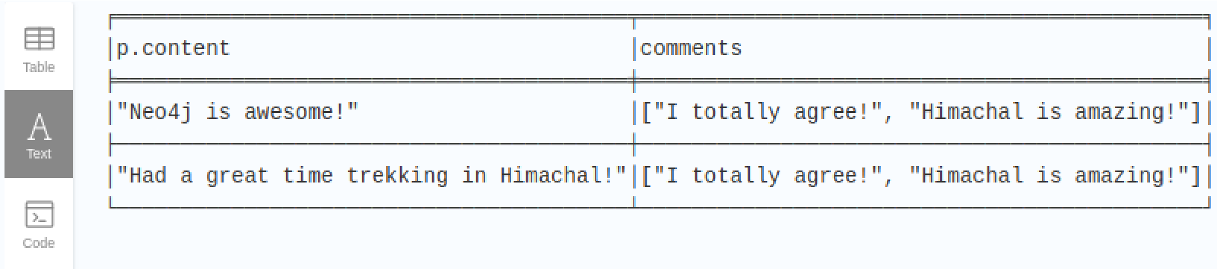
```
MATCH (p:Post)
RETURN p.content, p.likes_count, p.comments_count
ORDER BY p.likes_count DESC, p.comments_count DESC
LIMIT 1;
```



```
╒═══════════════════════════════════════╤═════════════╤════════════════╕
│p.content                              │p.likes_count│p.comments_count│
╞═══════════════════════════════════════╪═════════════╪════════════════╡
│"Best coding practices for beginners!" │120          │20              │
└───────────────────────────────────────┴─────────────┴────────────────┘
```

c. List all posts by a specific user and their associated comments.

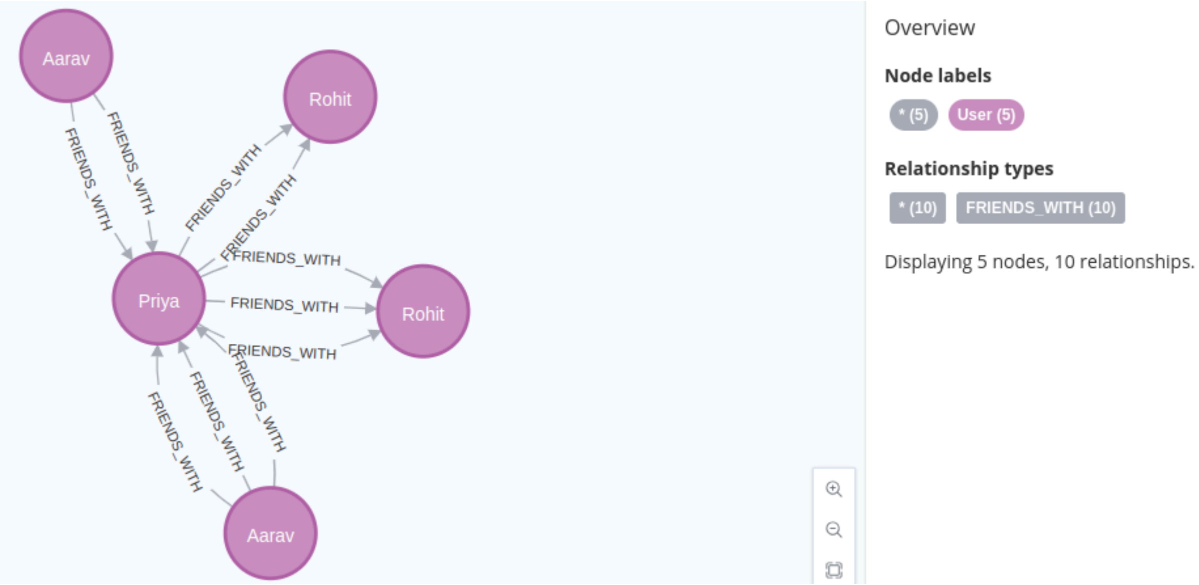
```
MATCH (u:User {name: "Priya"})-[:CREATED]->(p:Post)
OPTIONAL MATCH (p)<-[:ON_POST]-(c:Comment)
RETURN p.content, collect(DISTINCT c.content) AS comments;
```



```
╒═════════════════════════════════════════╤═══════════════════════════════════════════╕
│p.content                                │comments                                   │
╞═════════════════════════════════════════╪═══════════════════════════════════════════╡
│"Neo4j is awesome!"                      │["I totally agree!", "Himachal is amazing!"]│
├─────────────────────────────────────────┼───────────────────────────────────────────┤
│"Had a great time trekking in Himachal!"│["I totally agree!", "Himachal is amazing!"]│
└─────────────────────────────────────────┴───────────────────────────────────────────┘
```

d. Identify the shortest path between two users (mutual friends or common connections).

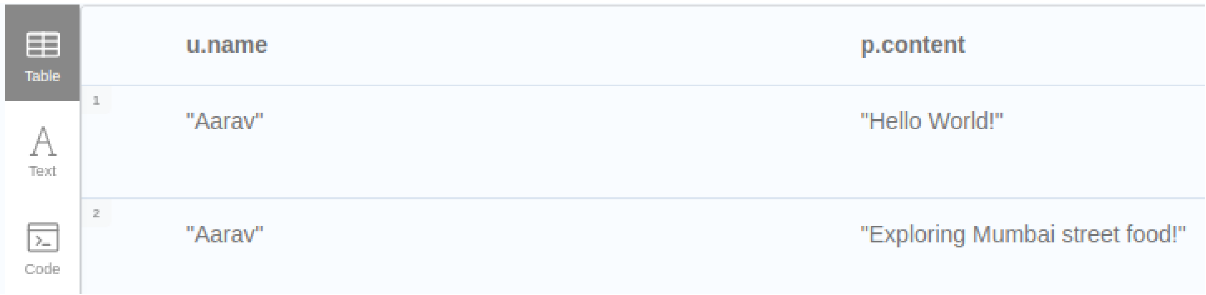
```
MATCH path = shortestPath((u1:User {name: "Aarav"})-[:FRIENDS_WITH*]-(u2:User
{name: "Rohit"}))
RETURN path;
```



Displaying 5 nodes, 10 relationships.

e. List all posts made by users who are located in a specific region.

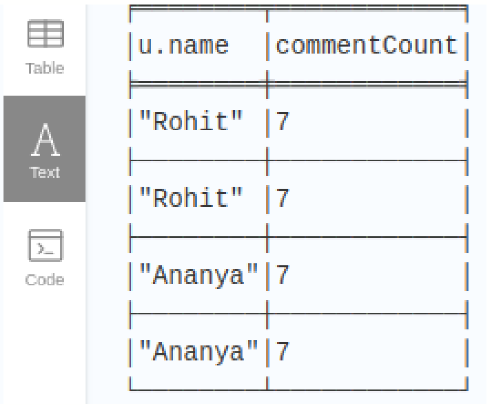
```
MATCH (u:User)-[:LOCATED_IN]->(l:Location {location_name: "Mumbai"}),
(u)-[:CREATED]->(p:Post)
RETURN DISTINCT u.name, p.content;
```



	u.name	p.content
1	"Aarav"	"Hello World!"
2	"Aarav"	"Exploring Mumbai street food!"

f. Find users who have commented on more than 5 posts in the last month.

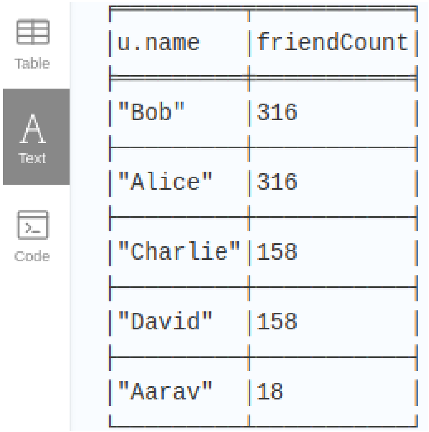
```
MATCH (u:User)-[:COMMENTED]->(c:Comment)
WHERE c.timestamp >= datetime() - duration({months: 1})
WITH u, count(DISTINCT c) AS commentCount
WHERE commentCount > 5
RETURN u.name, commentCount;
```



```
╒════════╤════════════╕
│u.name  │commentCount│
╞════════╪════════════╡
│"Rohit" │7           │
├────────┼────────────┤
│"Rohit" │7           │
├────────┼────────────┤
│"Ananya"│7           │
├────────┼────────────┤
│"Ananya"│7           │
└────────┴────────────┘
```

g. Find users who are the most connected (central users in the graph).

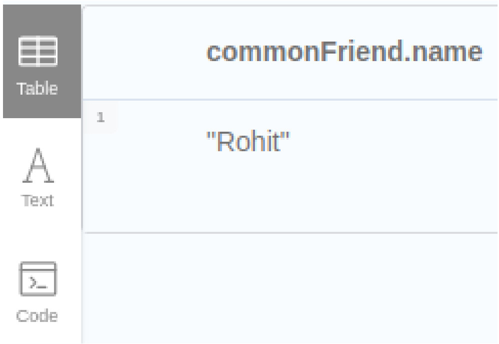
```
MATCH (u:User)-[:FRIENDS_WITH]-(friend)
RETURN u.name, count(friend) AS friendCount
ORDER BY friendCount DESC
LIMIT 5;
```



```
╒═════════╤═══════════╕
│u.name   │friendCount│
╞═════════╪═══════════╡
│"Bob"    │316        │
├─────────┼───────────┤
│"Alice"  │316        │
├─────────┼───────────┤
│"Charlie"│158        │
├─────────┼───────────┤
│"David"  │158        │
├─────────┼───────────┤
│"Aarav"  │18         │
└─────────┴───────────┘
```

h. Find mutual friends between two users.

```
MATCH (u1:User {name:
"Aarav"})-[:FRIENDS_WITH]-(commonFriend)-[:FRIENDS_WITH]-(u2:User {name:
"Priya"})
RETURN commonFriend.name;
```



	commonFriend.name
1	"Rohit"

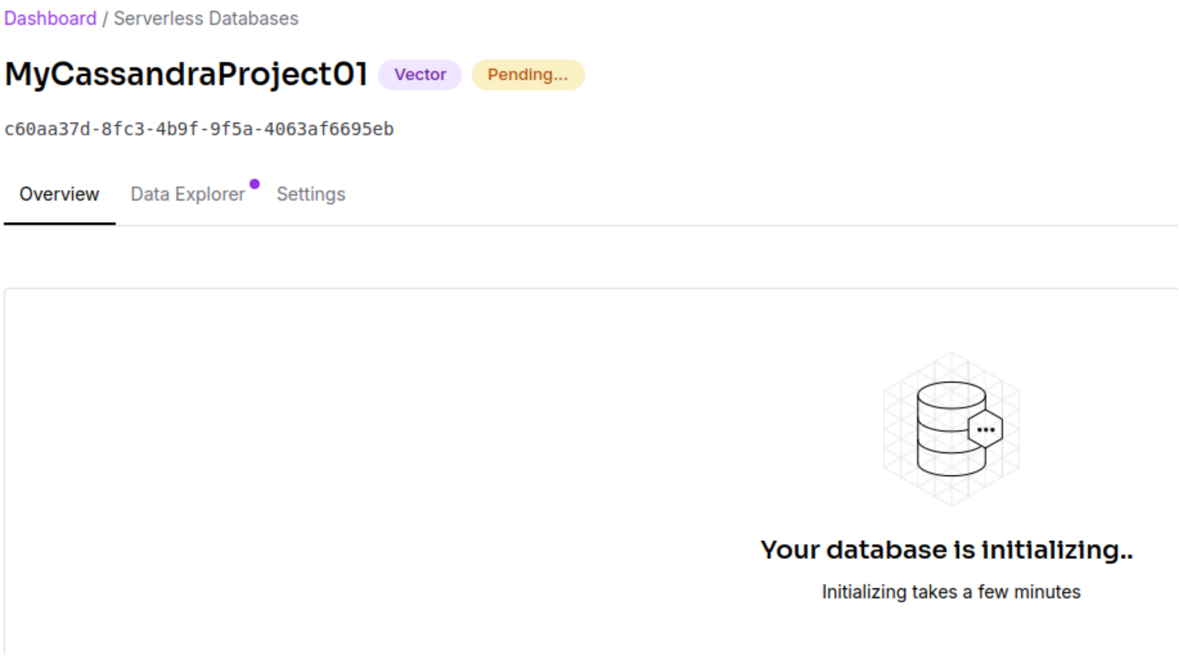
Summarised learning:

In this lab, I worked with Neo4j to build a social network model, incorporating nodes such as users, posts, comments, and locations, and relationships like friendship, post creation, and commenting. I wrote and executed various Cypher queries to explore the relationships and interactions within the network. The tasks included retrieving friends of a specific user, identifying the most popular posts, listing posts with comments, finding mutual friends, and analyzing user activity based on location and comment counts. This lab helped me gain hands-on experience in structuring data in a graph format and efficiently querying it to extract valuable insights.

Cassandra

Aim:- To understand and implement the use of distributed NoSQL database/s like Cassandra.

1. Objective : Students will get familiar with the distributed database.they are able to understand the architecture perspective of cassandra along with Cassandra Query Language(CQL).



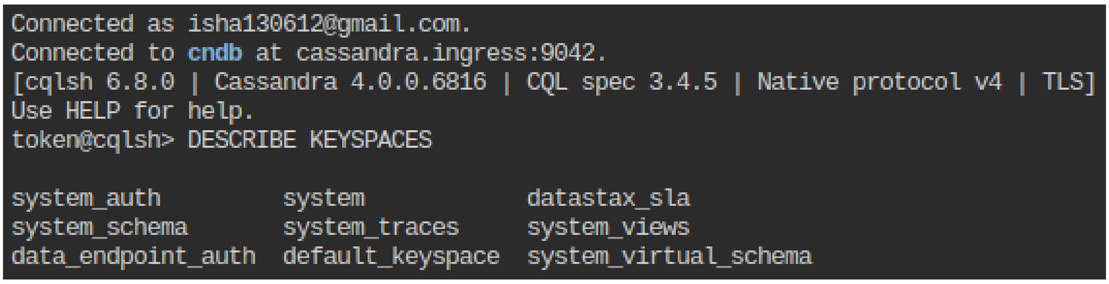
MyCassandraProject01

c60aa37d-8fc3-4b9f-9f5a-4063af6695eb

Your database is initializing..

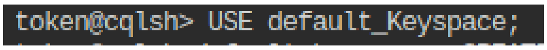
Initializing takes a few minutes

To find/describe the keyspaces



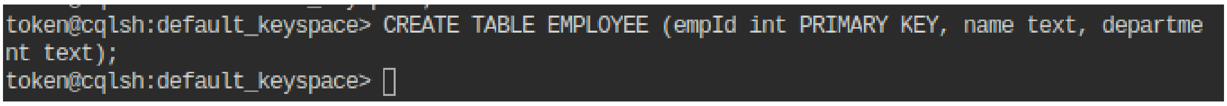
```
Connected as isha130612@gmail.com.
Connected to cndb at cassandra.ingress:9042.
[cqlsh 6.8.0 | Cassandra 4.0.0.6816 | CQL spec 3.4.5 | Native protocol v4 | TLS]
Use HELP for help.
token@cqlsh> DESCRIBE KEYSPACES

system_auth        system          datastax_sla
system_schema      system_traces   system_views
data_endpoint_auth default_keyspace system_virtual_schema
```

To use the keyspace



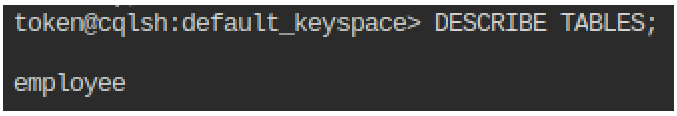
```
token@cqlsh> USE default_Keyspace;
```

To create a table in keyspace use CREATE TABLE COMMAND

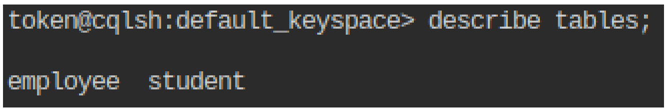


```
token@cqlsh:default_keyspace> CREATE TABLE EMPLOYEE (empId int PRIMARY KEY, name text, departme
nt text);
token@cqlsh:default_keyspace>
```

To display the list of tables present in keyspace

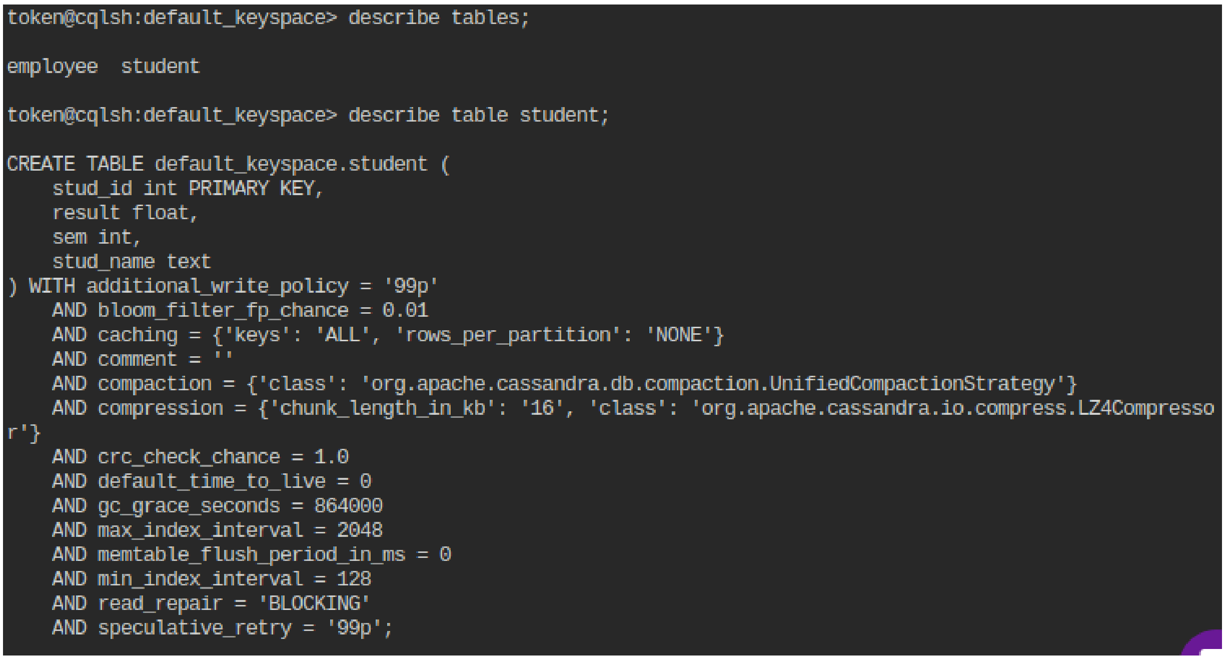


```
token@cqlsh:default_keyspace> DESCRIBE TABLES;

employee
```



```
token@cqlsh:default_keyspace> describe tables;

employee  student
```

To display the structure of a table use describe table tablename



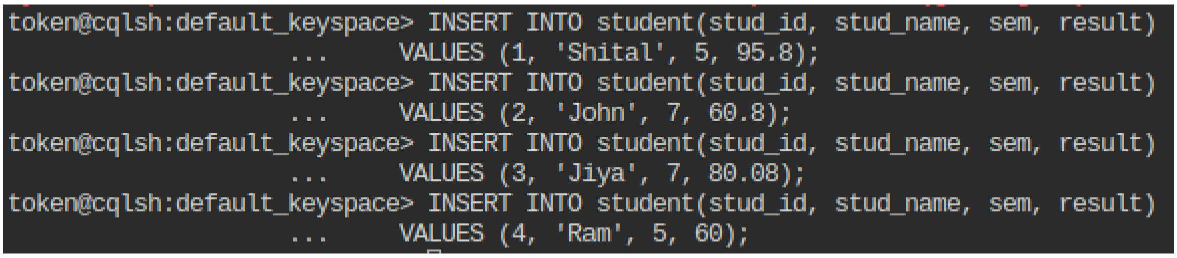
```
token@cqlsh:default_keyspace> describe tables;

employee  student

token@cqlsh:default_keyspace> describe table student;

CREATE TABLE default_keyspace.student (
    stud_id int PRIMARY KEY,
    result float,
    sem int,
    stud_name text
) WITH additional_write_policy = '99p'
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.UnifiedCompactionStrategy'}
    AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compresso
r'}
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    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';
```

Upsert Operation in Cassandra

Cassandra handles upserts with its INSERT and UPDATE statements, where:

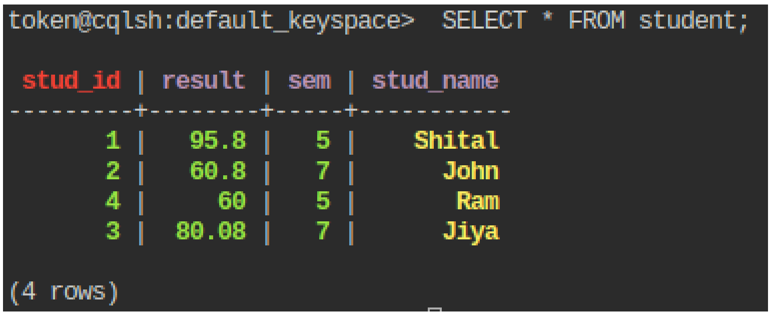
- INSERT Statement: If a record with the specified primary key does not exist, INSERT creates a new record. If it does exist, INSERT overwrites the existing record with the new values.
- UPDATE Statement: If the primary key exists, UPDATE modifies the existing record

with the new values. If it does not exist, UPDATE effectively performs an upsert by creating a new record with the specified values.



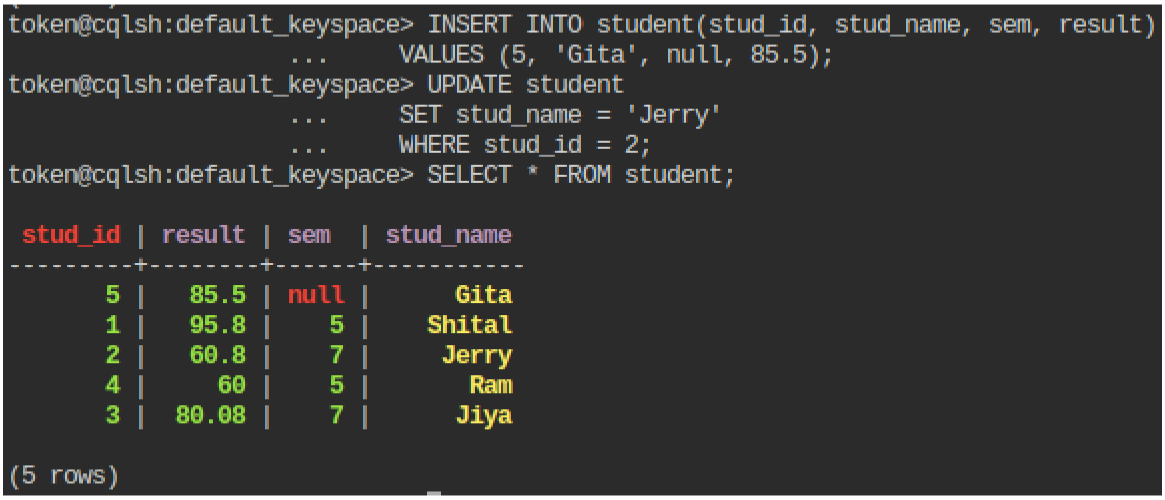
```
token@cqlsh:default_keyspace> INSERT INTO student(stud_id, stud_name, sem, result)
                          ...     VALUES (1, 'Shital', 5, 95.8);
token@cqlsh:default_keyspace> INSERT INTO student(stud_id, stud_name, sem, result)
                          ...     VALUES (2, 'John', 7, 60.8);
token@cqlsh:default_keyspace> INSERT INTO student(stud_id, stud_name, sem, result)
                          ...     VALUES (3, 'Jiya', 7, 80.08);
token@cqlsh:default_keyspace> INSERT INTO student(stud_id, stud_name, sem, result)
                          ...     VALUES (4, 'Ram', 5, 60);
```

To display the records use select



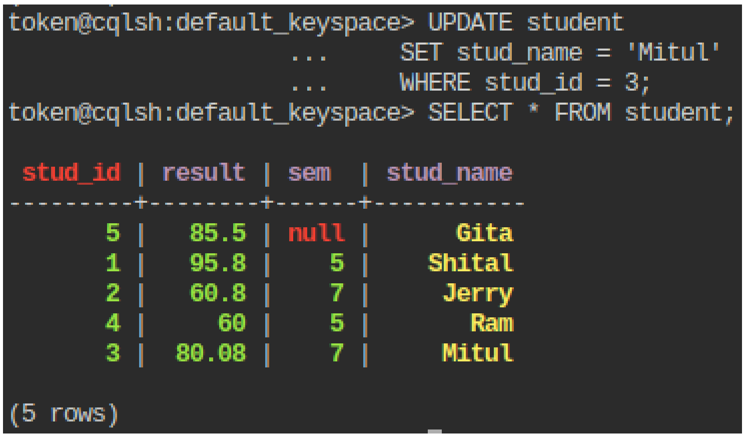
```
token@cqlsh:default_keyspace>  SELECT * FROM student;

 stud_id | result | sem | stud_name
---------+--------+-----+-----------
       1 |   95.8 |   5 |    Shital
       2 |   60.8 |   7 |      John
       4 |     60 |   5 |       Ram
       3 |  80.08 |   7 |      Jiya

(4 rows)
```

Query Example: (Student Result)



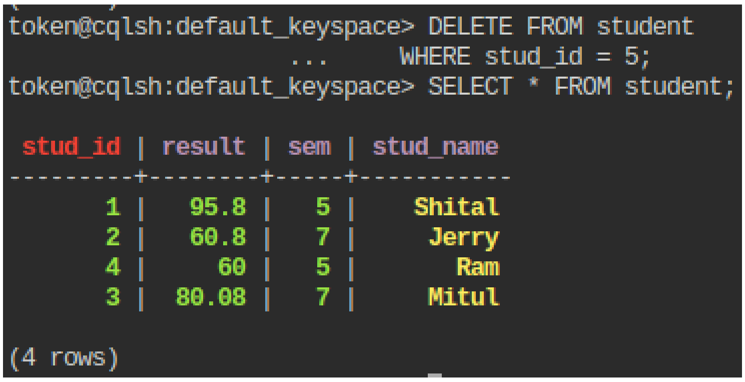
```
token@cqlsh:default_keyspace> INSERT INTO student(stud_id, stud_name, sem, result)
                          ...     VALUES (5, 'Gita', null, 85.5);
token@cqlsh:default_keyspace> UPDATE student
                          ...     SET stud_name = 'Jerry'
                          ...     WHERE stud_id = 2;
token@cqlsh:default_keyspace> SELECT * FROM student;

 stud_id | result | sem  | stud_name
---------+--------+------+-----------
       5 |   85.5 | null |      Gita
       1 |   95.8 |    5 |    Shital
       2 |   60.8 |    7 |     Jerry
       4 |     60 |    5 |       Ram
       3 |  80.08 |    7 |      Jiya

(5 rows)
```

Update will work as an insert if data is not present



```
token@cqlsh:default_keyspace> UPDATE student
                          ...     SET stud_name = 'Mitul'
                          ...     WHERE stud_id = 3;
token@cqlsh:default_keyspace> SELECT * FROM student;

 stud_id | result | sem  | stud_name
---------+--------+------+-----------
       5 |   85.5 | null |      Gita
       1 |   95.8 |    5 |    Shital
       2 |   60.8 |    7 |     Jerry
       4 |     60 |    5 |       Ram
       3 |  80.08 |    7 |     Mitul

(5 rows)
```

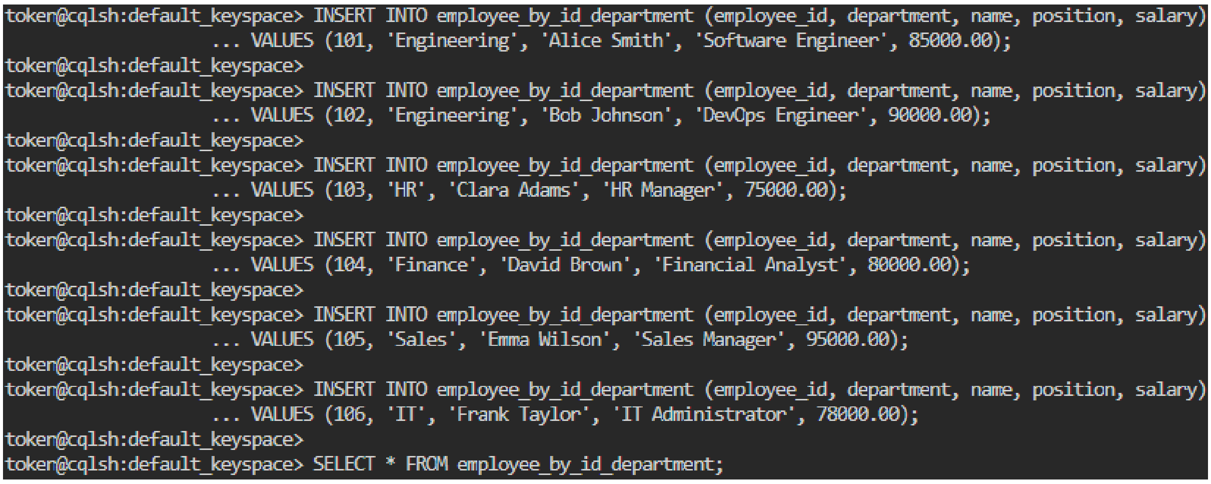
Delete operation



```
token@cqlsh:default_keyspace> DELETE FROM student
                    ...     WHERE stud_id = 5;
token@cqlsh:default_keyspace> SELECT * FROM student;

 stud_id | result | sem | stud_name
---------+--------+-----+-----------
       1 |   95.8 |   5 |    Shital
       2 |   60.8 |   7 |     Jerry
       4 |     60 |   5 |       Ram
       3 |  80.08 |   7 |     Mitul

(4 rows)
```

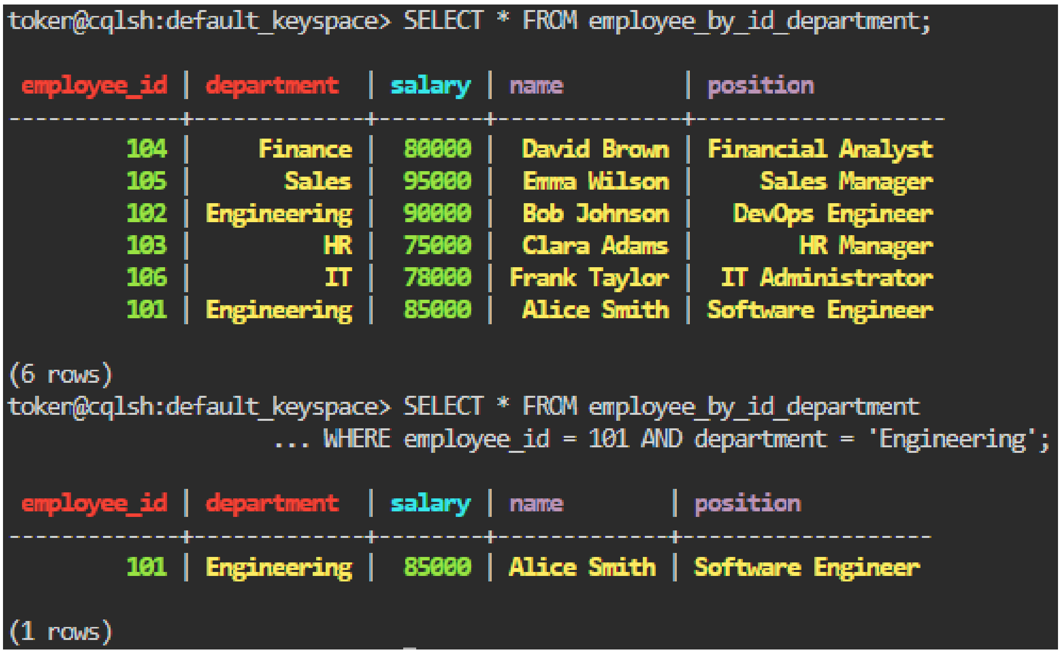
Design 1: By Employee ID and Department

After creating table, we insert these data values:



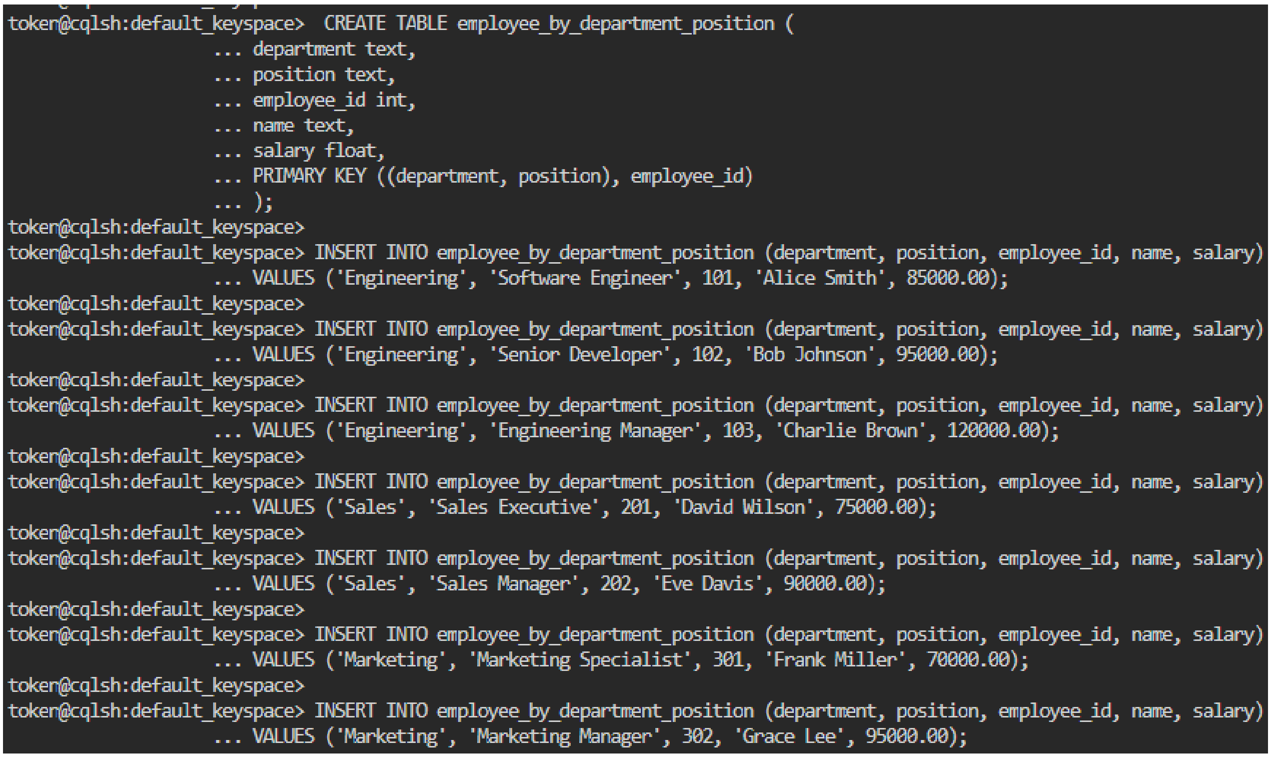
```
token@cqlsh:default_keyspace> INSERT INTO employee_by_id_department (employee_id, department, name, position, salary)
                          ... VALUES (101, 'Engineering', 'Alice Smith', 'Software Engineer', 85000.00);
token@cqlsh:default_keyspace>
token@cqlsh:default_keyspace> INSERT INTO employee_by_id_department (employee_id, department, name, position, salary)
                          ... VALUES (102, 'Engineering', 'Bob Johnson', 'DevOps Engineer', 90000.00);
token@cqlsh:default_keyspace>
token@cqlsh:default_keyspace> INSERT INTO employee_by_id_department (employee_id, department, name, position, salary)
                          ... VALUES (103, 'HR', 'Clara Adams', 'HR Manager', 75000.00);
token@cqlsh:default_keyspace>
token@cqlsh:default_keyspace> INSERT INTO employee_by_id_department (employee_id, department, name, position, salary)
                          ... VALUES (104, 'Finance', 'David Brown', 'Financial Analyst', 80000.00);
token@cqlsh:default_keyspace>
token@cqlsh:default_keyspace> INSERT INTO employee_by_id_department (employee_id, department, name, position, salary)
                          ... VALUES (105, 'Sales', 'Emma Wilson', 'Sales Manager', 95000.00);
token@cqlsh:default_keyspace>
token@cqlsh:default_keyspace> INSERT INTO employee_by_id_department (employee_id, department, name, position, salary)
                          ... VALUES (106, 'IT', 'Frank Taylor', 'IT Administrator', 78000.00);
token@cqlsh:default_keyspace>
token@cqlsh:default_keyspace> SELECT * FROM employee_by_id_department;
```

Output:



```
token@cqlsh:default_keyspace> SELECT * FROM employee_by_id_department;

 employee_id | department  | salary | name        | position
-------------+-------------+--------+-------------+-------------------
         104 |     Finance |  80000 | David Brown | Financial Analyst
         105 |       Sales |  95000 | Emma Wilson |     Sales Manager
         102 | Engineering |  90000 | Bob Johnson |   DevOps Engineer
         103 |          HR |  75000 | Clara Adams |        HR Manager
         106 |          IT |  78000 | Frank Taylor |  IT Administrator
         101 | Engineering |  85000 | Alice Smith | Software Engineer

(6 rows)
token@cqlsh:default_keyspace> SELECT * FROM employee_by_id_department
                          ... WHERE employee_id = 101 AND department = 'Engineering';

 employee_id | department  | salary | name        | position
-------------+-------------+--------+-------------+-------------------
         101 | Engineering |  85000 | Alice Smith | Software Engineer

(1 rows)
```

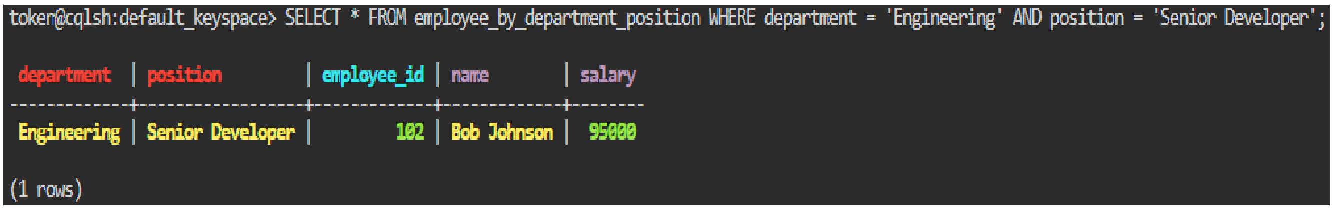
Design 2: By Department and Position



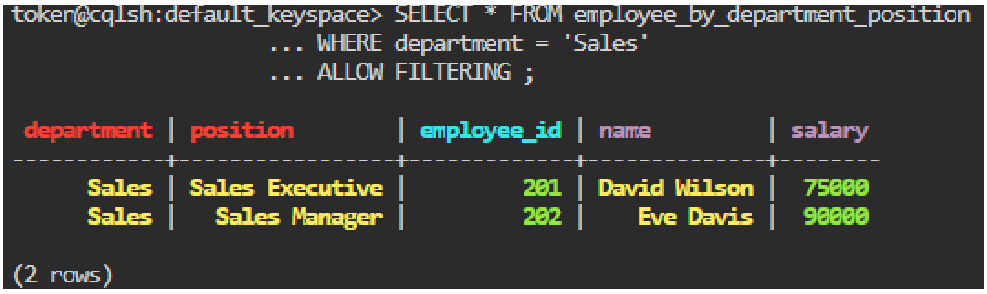
```
token@cqlsh:default_keyspace>  CREATE TABLE employee_by_department_position (
                           ... department text,
                           ... position text,
                           ... employee_id int,
                           ... name text,
                           ... salary float,
                           ... PRIMARY KEY ((department, position), employee_id)
                           ... );
token@cqlsh:default_keyspace>
token@cqlsh:default_keyspace> INSERT INTO employee_by_department_position (department, position, employee_id, name, salary)
                           ... VALUES ('Engineering', 'Software Engineer', 101, 'Alice Smith', 85000.00);
token@cqlsh:default_keyspace>
token@cqlsh:default_keyspace> INSERT INTO employee_by_department_position (department, position, employee_id, name, salary)
                           ... VALUES ('Engineering', 'Senior Developer', 102, 'Bob Johnson', 95000.00);
token@cqlsh:default_keyspace>
token@cqlsh:default_keyspace> INSERT INTO employee_by_department_position (department, position, employee_id, name, salary)
                           ... VALUES ('Engineering', 'Engineering Manager', 103, 'Charlie Brown', 120000.00);
token@cqlsh:default_keyspace>
token@cqlsh:default_keyspace> INSERT INTO employee_by_department_position (department, position, employee_id, name, salary)
                           ... VALUES ('Sales', 'Sales Executive', 201, 'David Wilson', 75000.00);
token@cqlsh:default_keyspace>
token@cqlsh:default_keyspace> INSERT INTO employee_by_department_position (department, position, employee_id, name, salary)
                           ... VALUES ('Sales', 'Sales Manager', 202, 'Eve Davis', 90000.00);
token@cqlsh:default_keyspace>
token@cqlsh:default_keyspace> INSERT INTO employee_by_department_position (department, position, employee_id, name, salary)
                           ... VALUES ('Marketing', 'Marketing Specialist', 301, 'Frank Miller', 70000.00);
token@cqlsh:default_keyspace>
token@cqlsh:default_keyspace> INSERT INTO employee_by_department_position (department, position, employee_id, name, salary)
                           ... VALUES ('Marketing', 'Marketing Manager', 302, 'Grace Lee', 95000.00);
```

Query Example:

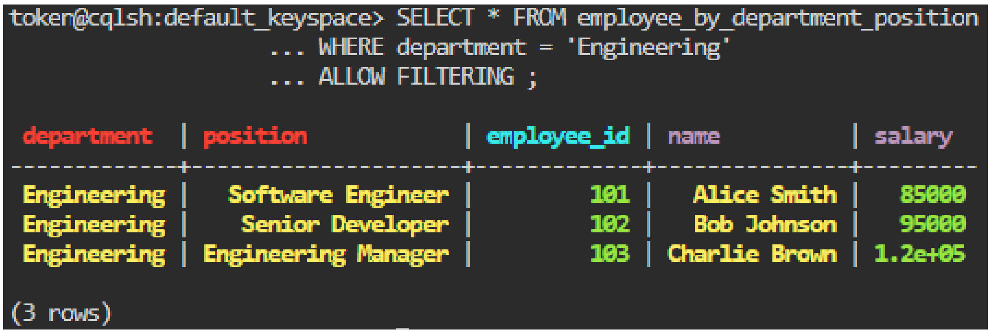
Query by Department and Position



```
token@cqlsh:default_keyspace> SELECT * FROM employee_by_department_position WHERE department = 'Engineering' AND position = 'Senior Developer';

 department  | position         | employee_id | name        | salary
-------------+------------------+-------------+-------------+--------
 Engineering | Senior Developer |         102 | Bob Johnson |  95000

(1 rows)
```

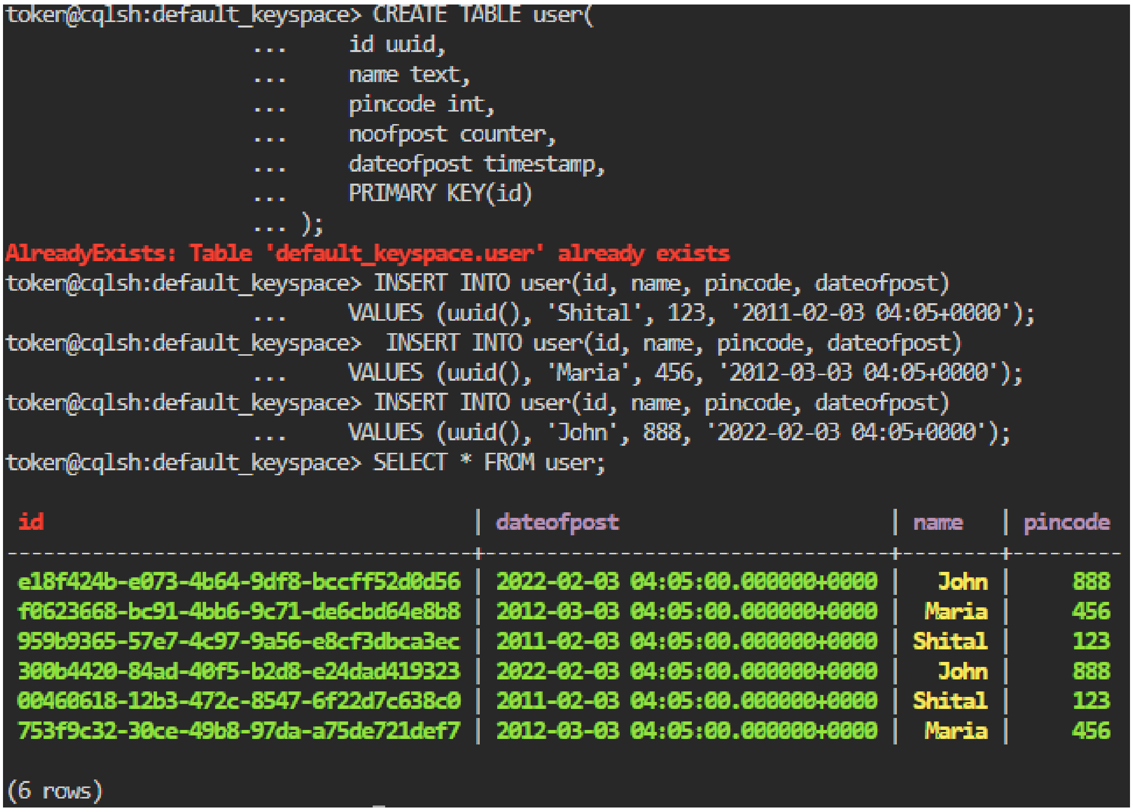


```
token@cqlsh:default_keyspace> SELECT * FROM employee_by_department_position
                           ... WHERE department = 'Sales'
                           ... ALLOW FILTERING ;

 department | position        | employee_id | name         | salary
------------+-----------------+-------------+--------------+--------
      Sales | Sales Executive |         201 | David Wilson |  75000
      Sales |   Sales Manager |         202 |    Eve Davis |  90000

(2 rows)
```



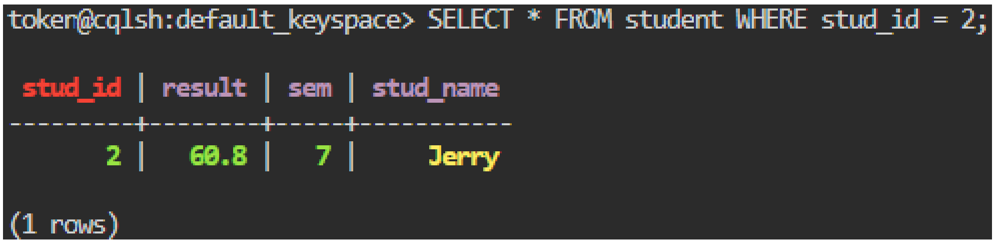
```
token@cqlsh:default_keyspace> SELECT * FROM employee_by_department_position
                           ... WHERE department = 'Engineering'
                           ... ALLOW FILTERING ;

 department  | position            | employee_id | name          | salary
-------------+---------------------+-------------+---------------+---------
 Engineering |   Software Engineer |         101 |   Alice Smith |   85000
 Engineering |    Senior Developer |         102 |   Bob Johnson |   95000
 Engineering | Engineering Manager |         103 | Charlie Brown | 1.2e+05

(3 rows)
```

Query example: datatypes uuid,timestamp (Table user)



```
token@cqlsh:default_keyspace> CREATE TABLE user(
                          ...     id uuid,
                          ...     name text,
                          ...     pincode int,
                          ...     noofpost counter,
                          ...     dateofpost timestamp,
                          ...     PRIMARY KEY(id)
                          ... );
AlreadyExists: Table 'default_keyspace.user' already exists
token@cqlsh:default_keyspace> INSERT INTO user(id, name, pincode, dateofpost)
                          ...     VALUES (uuid(), 'Shital', 123, '2011-02-03 04:05+0000');
token@cqlsh:default_keyspace>  INSERT INTO user(id, name, pincode, dateofpost)
                          ...     VALUES (uuid(), 'Maria', 456, '2012-03-03 04:05+0000');
token@cqlsh:default_keyspace> INSERT INTO user(id, name, pincode, dateofpost)
                          ...     VALUES (uuid(), 'John', 888, '2022-02-03 04:05+0000');
token@cqlsh:default_keyspace> SELECT * FROM user;

 id                                   | dateofpost                      | name   | pincode
--------------------------------------+---------------------------------+--------+---------
 e18f424b-e073-4b64-9df8-bccff52d0d56 | 2022-02-03 04:05:00.000000+0000 |   John |     888
 f0623668-bc91-4bb6-9c71-de6cbd64e8b8 | 2012-03-03 04:05:00.000000+0000 |  Maria |     456
 959b9365-57e7-4c97-9a56-e8cf3dbca3ec | 2011-02-03 04:05:00.000000+0000 | Shital |     123
 300b4420-84ad-40f5-b2d8-e24dad419323 | 2022-02-03 04:05:00.000000+0000 |   John |     888
 00460618-12b3-472c-8547-6f22d7c638c0 | 2011-02-03 04:05:00.000000+0000 | Shital |     123
 753f9c32-30ce-49b8-97da-a75de721def7 | 2012-03-03 04:05:00.000000+0000 |  Maria |     456

(6 rows)
```

To display output

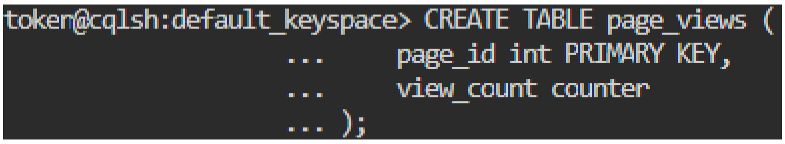


```
token@cqlsh:default_keyspace> SELECT * FROM student WHERE stud_id = 2;

 stud_id | result | sem | stud_name
---------+--------+-----+-----------
       2 |   60.8 |   7 |     Jerry

(1 rows)
```

COUNTER IN CASSANDRA:

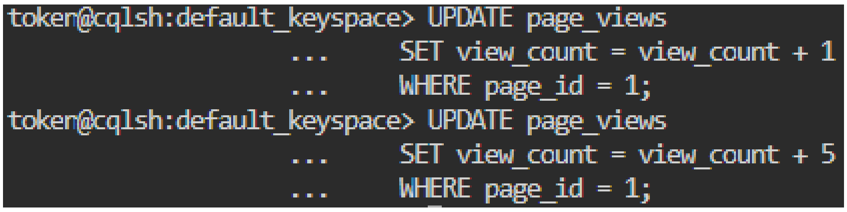
1. Create a Counter Table

In Cassandra, counter columns require a PRIMARY KEY, and counters cannot exist with non-counter columns except the primary key.



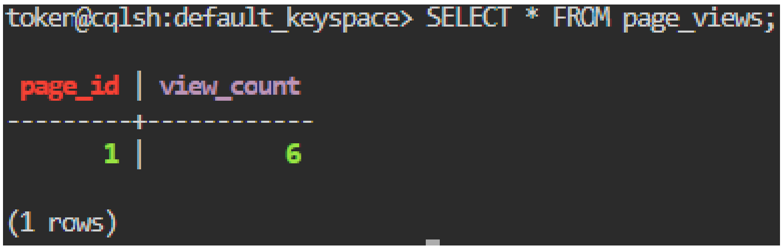
```
token@cqlsh:default_keyspace> CREATE TABLE page_views (
                          ...     page_id int PRIMARY KEY,
                          ...     view_count counter
                          ... );
```

2. Increment the Counter



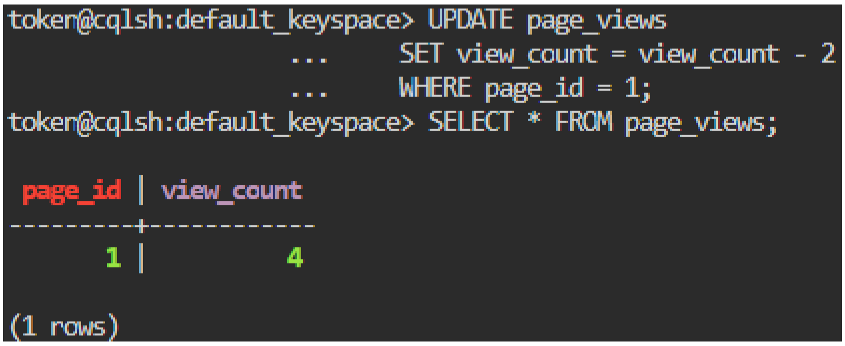
```
token@cqlsh:default_keyspace> UPDATE page_views
                          ...     SET view_count = view_count + 1
                          ...     WHERE page_id = 1;
token@cqlsh:default_keyspace> UPDATE page_views
                          ...     SET view_count = view_count + 5
                          ...     WHERE page_id = 1;
```

3. Select Data from the Counter Table



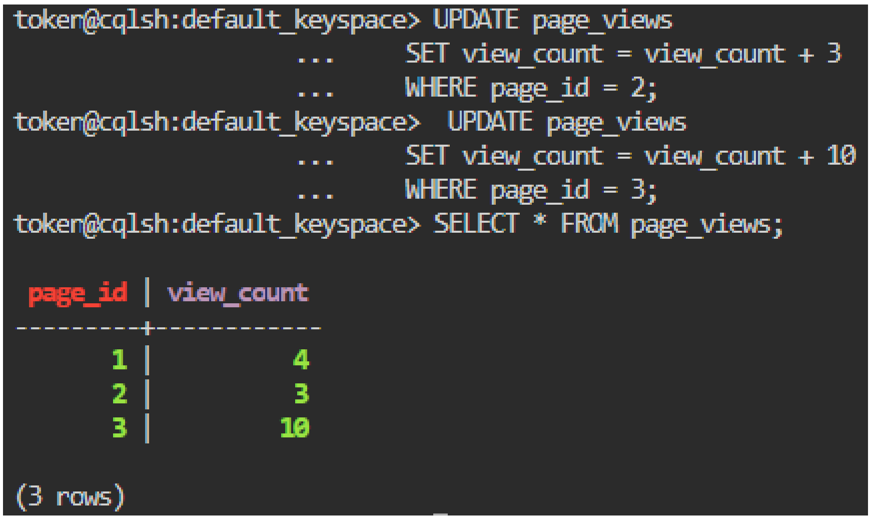
```
token@cqlsh:default_keyspace> SELECT * FROM page_views;

 page_id | view_count
---------+------------
       1 |          6

(1 rows)
```

4. Decrement the Counter



```
token@cqlsh:default_keyspace> UPDATE page_views
                          ...     SET view_count = view_count - 2
                          ...     WHERE page_id = 1;
token@cqlsh:default_keyspace> SELECT * FROM page_views;

 page_id | view_count
---------+------------
       1 |          4

(1 rows)
```

5. Insert More Pages and Track Views

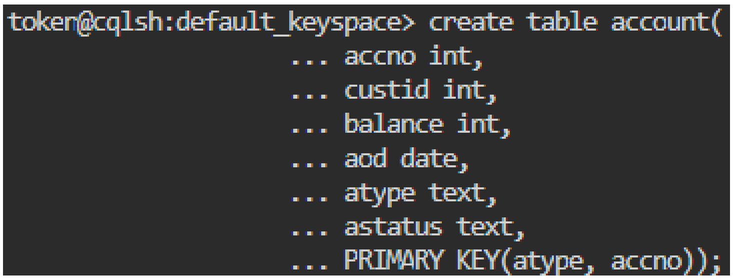


```
token@cqlsh:default_keyspace> UPDATE page_views
                          ...     SET view_count = view_count + 3
                          ...     WHERE page_id = 2;
token@cqlsh:default_keyspace>  UPDATE page_views
                          ...     SET view_count = view_count + 10
                          ...     WHERE page_id = 3;
token@cqlsh:default_keyspace> SELECT * FROM page_views;

 page_id | view_count
---------+------------
       1 |          4
       2 |          3
       3 |         10

(3 rows)
```

This completes all the steps for **creating, incrementing, decrementing, and deleting** counter values in Cassandra

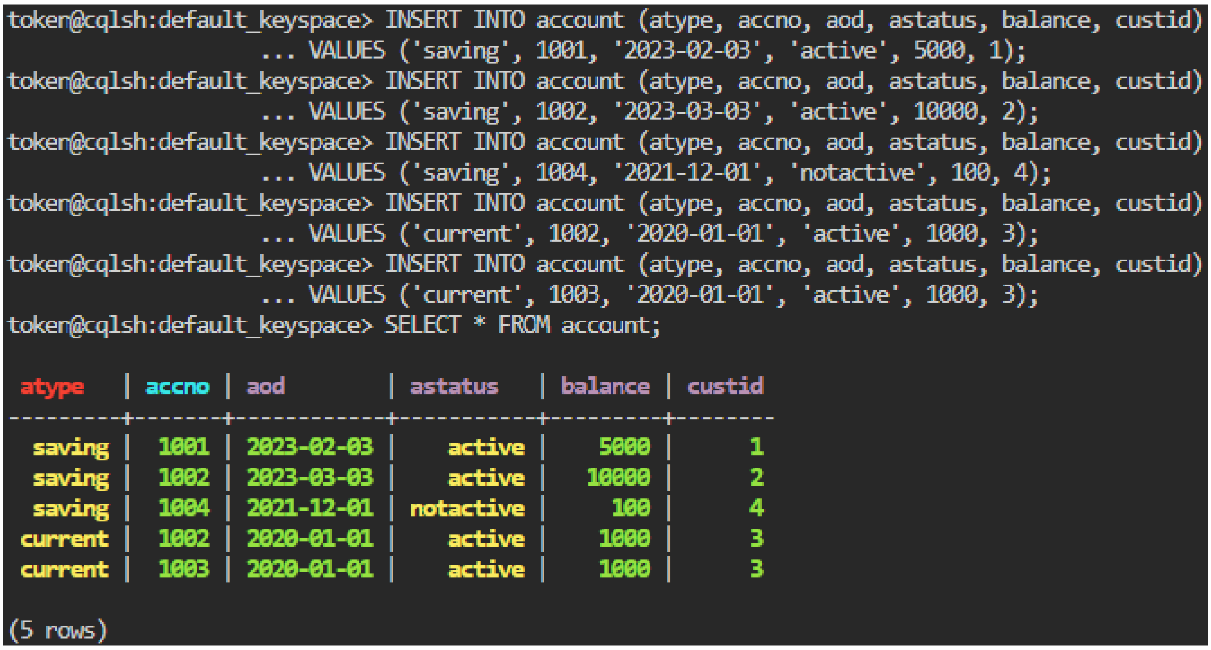
EXERCISE:

1. Create table account as follows



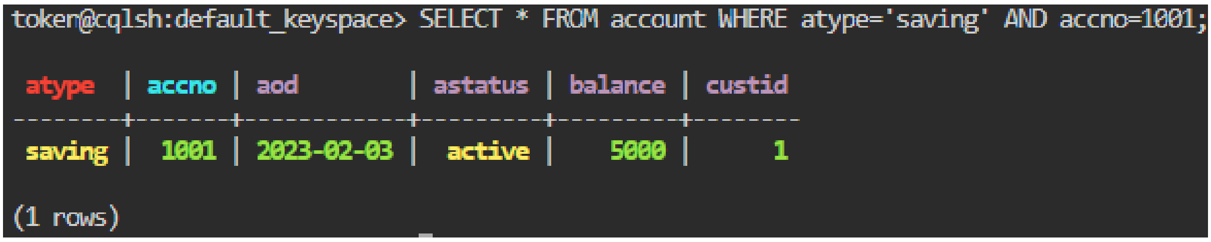
```
token@cqlsh:default_keyspace> create table account(
                          ... accno int,
                          ... custid int,
                          ... balance int,
                          ... aod date,
                          ... atype text,
                          ... astatus text,
                          ... PRIMARY KEY(atype, accno));
```

2. Insert records into the table account as follows.(using upsert operations in cassandra)



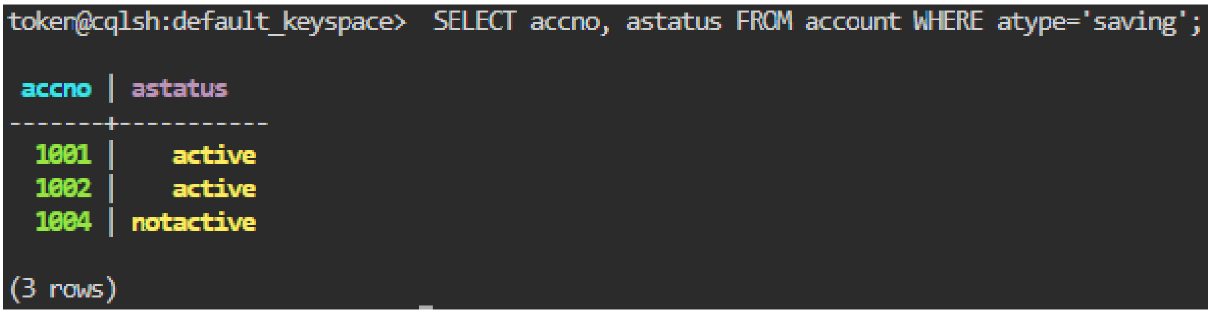
```
token@cqlsh:default_keyspace> INSERT INTO account (atype, accno, aod, astatus, balance, custid)
                          ... VALUES ('saving', 1001, '2023-02-03', 'active', 5000, 1);
token@cqlsh:default_keyspace> INSERT INTO account (atype, accno, aod, astatus, balance, custid)
                          ... VALUES ('saving', 1002, '2023-03-03', 'active', 10000, 2);
token@cqlsh:default_keyspace> INSERT INTO account (atype, accno, aod, astatus, balance, custid)
                          ... VALUES ('saving', 1004, '2021-12-01', 'notactive', 100, 4);
token@cqlsh:default_keyspace> INSERT INTO account (atype, accno, aod, astatus, balance, custid)
                          ... VALUES ('current', 1002, '2020-01-01', 'active', 1000, 3);
token@cqlsh:default_keyspace> INSERT INTO account (atype, accno, aod, astatus, balance, custid)
                          ... VALUES ('current', 1003, '2020-01-01', 'active', 1000, 3);
token@cqlsh:default_keyspace> SELECT * FROM account;

 atype   | accno | aod        | astatus   | balance | custid
---------+-------+------------+-----------+---------+--------
  saving |  1001 | 2023-02-03 |    active |    5000 |      1
  saving |  1002 | 2023-03-03 |    active |   10000 |      2
  saving |  1004 | 2021-12-01 | notactive |     100 |      4
 current |  1002 | 2020-01-01 |    active |    1000 |      3
 current |  1003 | 2020-01-01 |    active |    1000 |      3

(5 rows)
```

3. Display the details of account 1001.



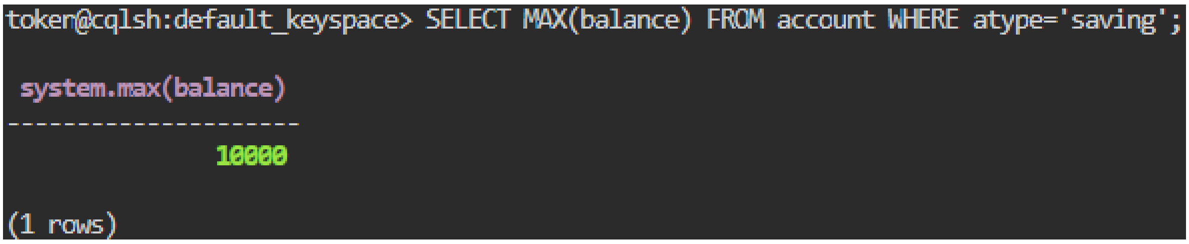
```
token@cqlsh:default_keyspace> SELECT * FROM account WHERE atype='saving' AND accno=1001;

 atype  | accno | aod        | astatus | balance | custid
--------+-------+------------+---------+---------+--------
 saving |  1001 | 2023-02-03 |  active |    5000 |      1

(1 rows)
```

4. Display account status along with account number of all the saving accounts.



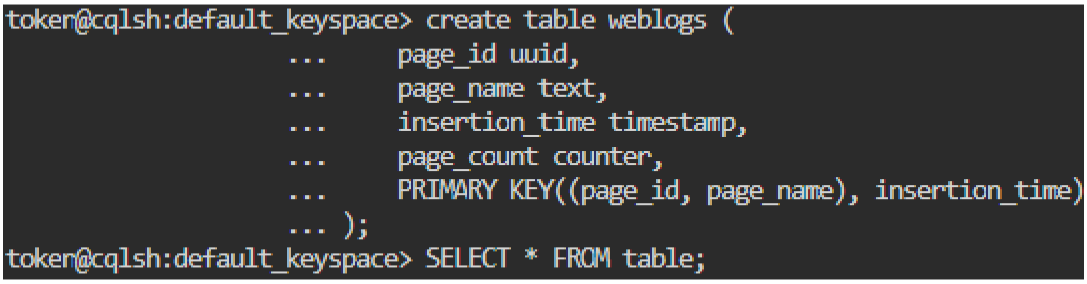
```
token@cqlsh:default_keyspace>  SELECT accno, astatus FROM account WHERE atype='saving';

 accno | astatus
-------+-----------
  1001 |    active
  1002 |    active
  1004 | notactive

(3 rows)
```

5. Display the highest balance of the savings account.



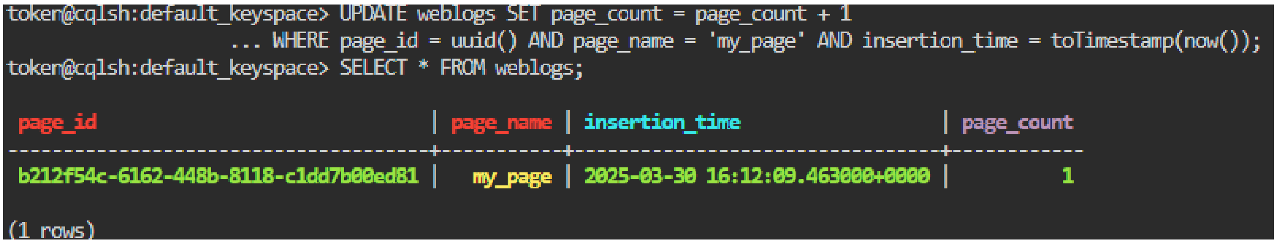
```
token@cqlsh:default_keyspace> SELECT MAX(balance) FROM account WHERE atype='saving';

 system.max(balance)
---------------------
               10000

(1 rows)
```

6. Create table weblogs as per the following given data types.
(Check the function now(),dateof() for current time and date of the system)



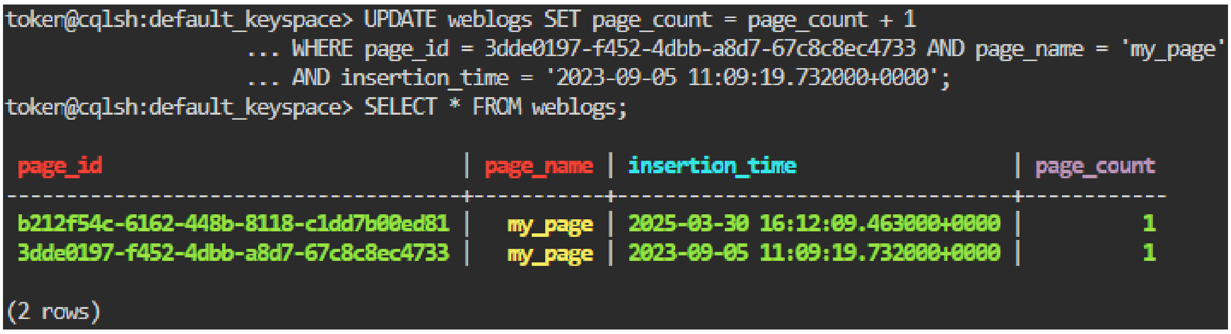
```
token@cqlsh:default_keyspace> create table weblogs (
                          ...     page_id uuid,
                          ...     page_name text,
                          ...     insertion_time timestamp,
                          ...     page_count counter,
                          ...     PRIMARY KEY((page_id, page_name), insertion_time)
                          ... );
token@cqlsh:default_keyspace> SELECT * FROM table;
```

7. Insert records in order to understand the use of all different types.



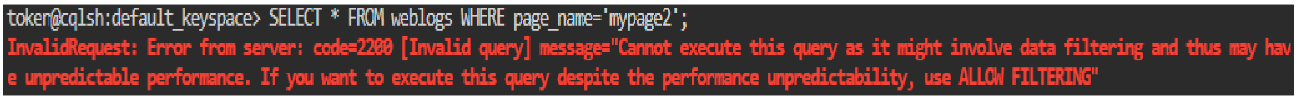
```
token@cqlsh:default_keyspace> UPDATE weblogs SET page_count = page_count + 1
                          ... WHERE page_id = uuid() AND page_name = 'my_page' AND insertion_time = toTimestamp(now());
token@cqlsh:default_keyspace> SELECT * FROM weblogs;

 page_id                              | page_name | insertion_time                  | page_count
--------------------------------------+-----------+---------------------------------+------------
 b212f54c-6162-448b-8118-c1dd7b00ed81 |   my_page | 2025-03-30 16:12:09.463000+0000 |          1

(1 rows)
```

8. Let suppose that a user visited the same page for the second time then the page_count should be incremented to one.write a cassandra query to insert the value in the given table.your query should generate the output as shown in below Figure.



```
token@cqlsh:default_keyspace> UPDATE weblogs SET page_count = page_count + 1
                          ... WHERE page_id = 3dde0197-f452-4dbb-a8d7-67c8c8ec4733 AND page_name = 'my_page'
                          ... AND insertion_time = '2023-09-05 11:09:19.732000+0000';
token@cqlsh:default_keyspace> SELECT * FROM weblogs;

 page_id                              | page_name | insertion_time                  | page_count
--------------------------------------+-----------+---------------------------------+------------
 b212f54c-6162-448b-8118-c1dd7b00ed81 |   my_page | 2025-03-30 16:12:09.463000+0000 |          1
 3dde0197-f452-4dbb-a8d7-67c8c8ec4733 |   my_page | 2023-09-05 11:09:19.732000+0000 |          1

(2 rows)
```

9. Check what if we execute the following query. Justify your answer.
select * from weblogs where page_name='mypage2'

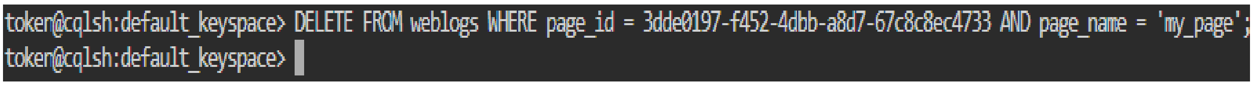


```
token@cqlsh:default_keyspace> SELECT * FROM weblogs WHERE page_name='mypage2';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involve data filtering and thus may hav
e unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING"
```

Justification:

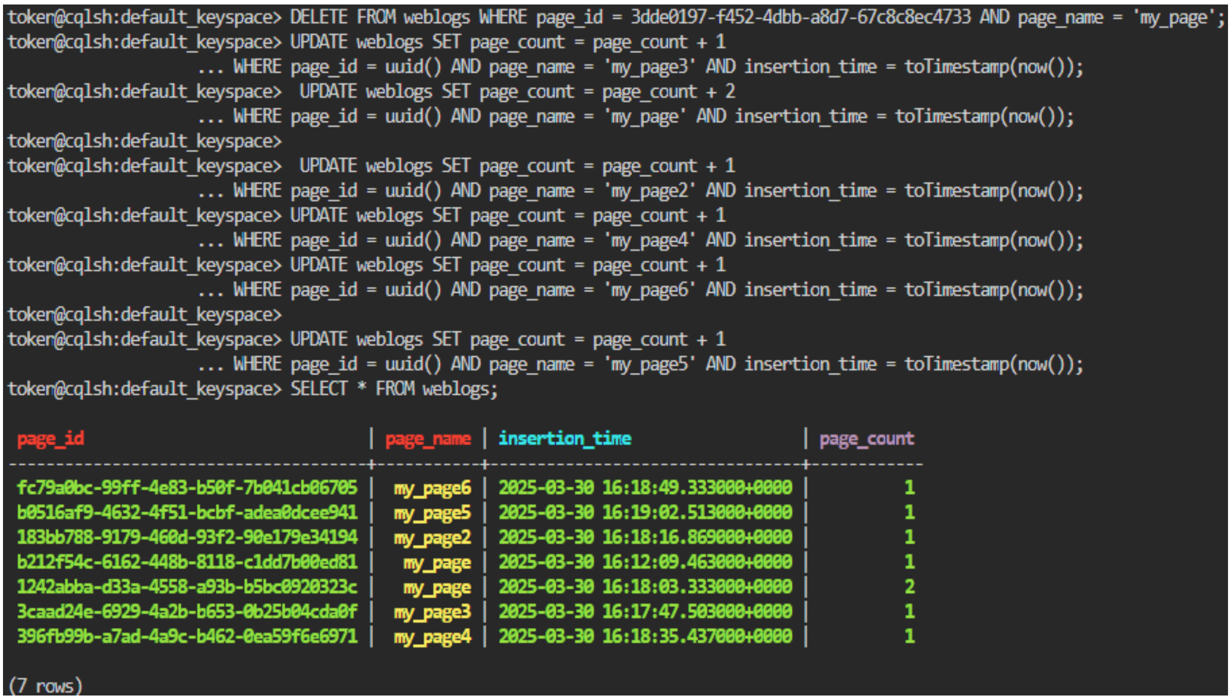
This query will fail because page_name is part of the composite primary key but is not the partition key. Cassandra requires at least the partition key (page_id) in the WHERE clause.

10.Delete one of the Pages from the table.



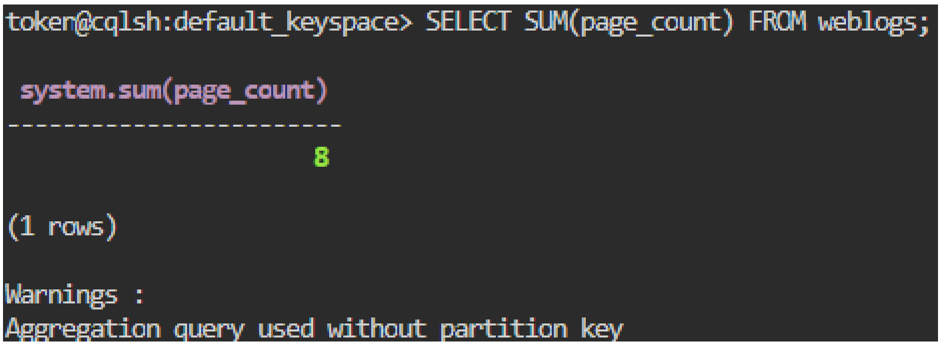
```
token@cqlsh:default_keyspace> DELETE FROM weblogs WHERE page_id = 3dde0197-f452-4dbb-a8d7-67c8c8ec4733 AND page_name = 'my_page';
token@cqlsh:default_keyspace>
```

11. Insert a few page records and display them like the following figure. understand the purpose of partition key column, clustering key column.



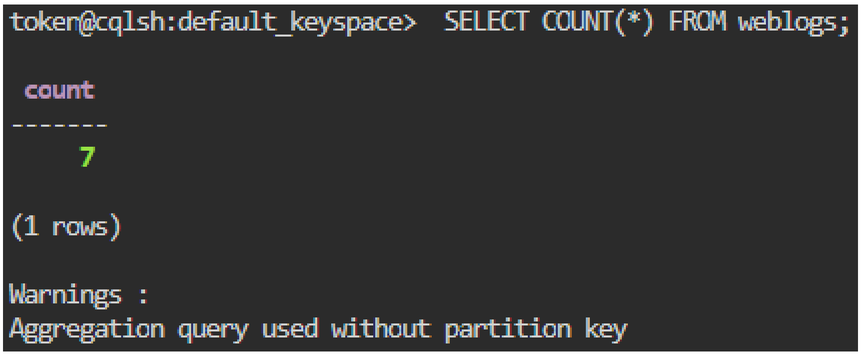
```
token@cqlsh:default_keyspace> DELETE FROM weblogs WHERE page_id = 3dde0197-f452-4dbb-a8d7-67c8c8ec4733 AND page_name = 'my_page';
token@cqlsh:default_keyspace> UPDATE weblogs SET page_count = page_count + 1
                          ... WHERE page_id = uuid() AND page_name = 'my_page3' AND insertion_time = toTimestamp(now());
token@cqlsh:default_keyspace>  UPDATE weblogs SET page_count = page_count + 2
                          ... WHERE page_id = uuid() AND page_name = 'my_page' AND insertion_time = toTimestamp(now());
token@cqlsh:default_keyspace>
token@cqlsh:default_keyspace>  UPDATE weblogs SET page_count = page_count + 1
                          ... WHERE page_id = uuid() AND page_name = 'my_page2' AND insertion_time = toTimestamp(now());
token@cqlsh:default_keyspace> UPDATE weblogs SET page_count = page_count + 1
                          ... WHERE page_id = uuid() AND page_name = 'my_page4' AND insertion_time = toTimestamp(now());
token@cqlsh:default_keyspace> UPDATE weblogs SET page_count = page_count + 1
                          ... WHERE page_id = uuid() AND page_name = 'my_page6' AND insertion_time = toTimestamp(now());
token@cqlsh:default_keyspace>
token@cqlsh:default_keyspace> UPDATE weblogs SET page_count = page_count + 1
                          ... WHERE page_id = uuid() AND page_name = 'my_page5' AND insertion_time = toTimestamp(now());
token@cqlsh:default_keyspace> SELECT * FROM weblogs;

 page_id                              | page_name | insertion_time                  | page_count
--------------------------------------+-----------+---------------------------------+------------
 fc79a0bc-99ff-4e83-b50f-7b041cb06705 |  my_page6 | 2025-03-30 16:18:49.333000+0000 |          1
 b0516af9-4632-4f51-bcbf-adea0dcee941 |  my_page5 | 2025-03-30 16:19:02.513000+0000 |          1
 183bb788-9179-460d-93f2-90e179e34194 |  my_page2 | 2025-03-30 16:18:16.869000+0000 |          1
 b212f54c-6162-448b-8118-c1dd7b00ed81 |   my_page | 2025-03-30 16:12:09.463000+0000 |          1
 1242abba-d33a-4558-a93b-b5bc0920323c |   my_page | 2025-03-30 16:18:03.333000+0000 |          2
 3caad24e-6929-4a2b-b653-0b25b04cda0f |  my_page3 | 2025-03-30 16:17:47.503000+0000 |          1
 396fb99b-a7ad-4a9c-b462-0ea59f6e6971 |  my_page4 | 2025-03-30 16:18:35.437000+0000 |          1

(7 rows)
```

12.Display the total of all page record count.



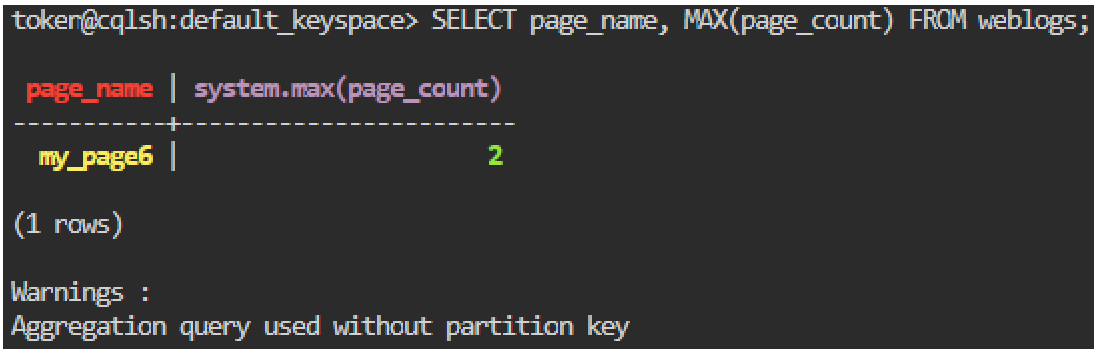
```
token@cqlsh:default_keyspace> SELECT SUM(page_count) FROM weblogs;

 system.sum(page_count)
------------------------
                      8

(1 rows)

Warnings :
Aggregation query used without partition key
```

13.Display how many pages are stored in the table.



```
token@cqlsh:default_keyspace>  SELECT COUNT(*) FROM weblogs;

 count
-------
     7

(1 rows)

Warnings :
Aggregation query used without partition key
```

14.Display the page which has the highest number of visitors(page_count).



```
token@cqlsh:default_keyspace> SELECT page_name, MAX(page_count) FROM weblogs;

 page_name | system.max(page_count)
-----------+------------------------
  my_page6 |                      2

(1 rows)

Warnings :
Aggregation query used without partition key
```

15.Explore, Are Joins and Subqueries supported in Cassandra? If Yes, then How? If No, then Why?

No, Joins and Subqueries are not supported in Cassandra.

Why?

- ***Denormalization is encouraged****:* Cassandra is a NoSQL database optimized for speed and scalability, so it avoids joins to improve performance.

- ***No Complex Queries****:* Cassandra does not have relational integrity constraints like foreign keys.

- ***Designed for Distributed Systems****:* Since data is stored across multiple nodes, performing joins would be inefficient and slow.

- ***Alternative Approach***: Data should be pre-joined and stored in separate tables based on access patterns.

Summarised learning:

In this lab, I explored key concepts and operations in Cassandra, including table creation, data insertion using upsert operations, and querying. I worked with partition and clustering keys to organize data, and explored counters for managing increment/decrement operations on values. Additionally, I gained hands-on experience with various query operations such as retrieving specific data, calculating aggregations, and managing counter columns. I also worked with more advanced topics like timestamp and counter-based tables, and understood the importance of partition and clustering keys in efficient data management.