Day – 14

1.Write cql query to display all the employees whose name   ends with ‘a’?

```
cqlsh > Select * from Employee where emp_name LIKE ‘%a’;
```

2. Write cql query to display the total salary of all the

employees whose designation is programmer?

```
cqlsh > Select   sum( salary) from Employee where emp_dsgn =’Programmer’;
```

3. Write sql query to display all the employees whose salary is between 30000 and 45000?

```
cqlsh > Select * from Employee where emp_salary >= 30000 AND emp_salary <=
45000;
```

4. Write sql query to display all the employees whose are coming from medchal.?

```
cqlsh > Select * from Employee where emp_city< ‘Medchal’;
```

5 Write cql query to display the details of the employee with highest experience?

```
cqlsh > Select * from Employee where emp_sal = MAX(emp_sal);
```

6. Write cql query to display the details of the employees whose name contains‘ee’

```
cqlsh > Select * from Employee where emp_name LIKE ‘%ee%’;
```

7 Write cql query to increase the salaries of employees by 5000 whose designation is DBA?

```
cqlsh > Update Employee SET emp_sal=emp_sal+5000

where emp_dsgn = “DBA’;
```

8. Write cql query to display the employees whose salary is more than the average salary of all the employees.

```
cqlsh > Select * from Employee where emp_sal > AVG(emp_sal);
```

Day -15:

1.Create the table called Student with the below mentioned details.

Student (Sid (Primary Key) , Sname ,DOB, Gender, Course, Address,percentage)

cqlsh > CREATE TABLE Student (Sid int Primary Key , Sname text, DOB text, Gender text, Course text, Address text ,percentage float)

I. Insert 10 rows in to the Student Table.

cqlsh > INSERT INTO Student (Sid, Sname , DOB, Gender, Course, Address, percentage) VALUES (51, 'Kashyap', '2001-09-23', Male', 'Data Science', 'Medchal', 79.7)

Same way Write query to Insert 10 rows with different values.

2. Insert 10 rows in to the Student Table.

Display all the student records

cqlsh > Select * from Student;

3. I Display the records Course is "Data Science"

II Write cql query to display all the female students enrolled under BCOM course.

I cqlsh > Select * from Student where Course= 'Data Science';

II. cqlsh > Select * from Student where Gender= 'Female' AND Course= 'BCOM';

. 4 Write a cql query to add a new columns Contactno to the existing fields.

cqlsh > ALTER TABLE Student ADD Contact_no text;

5. Write a cql query to display all the Student names where the length of the name is 5 characters.

cqlsh > Select * from Student where Sname LIKE '_ _ _ _ _';

6.I Update the student contact no of the student with sid 105

II Display the student records whose percentage is less than 50

cqlsh > Update Student SET Contact_no = '9000000000' where Sid = 105;

cqlsh > Select * from Student where Percentage < 50;

7. Display the student records whose percentage is greater than 60 in descending order

cqlsh > Select * from Student where Percentage > 50 DESC;

8. Write cql query to delete all the students records who have enrolled for BA course.

cqlsh > DELETE FROM Student WHERE Course= 'BA';

Day-16:

Create the table called BOOK with the below mentioned details.

1.BOOK (BookId (Primary Key), BookName, Author,DatePurchased ,Publisher, Price)

cqlsh > Create table BOOK (BookId int Primary Key, BookName text, Author text,DatePurchased text, Publisher text, Price float);

2.

Insert 10 rows in to the Book Table.

cqlsh > INSERT INTO Book (Bookid, Bookname , DatePurchased, Publisher, Price) VALUES (51, 'Python', '2024-09-23', Male', 'Himalaya', 790);

…………………………………………………….

3. Write cql query to display the list of authors from Himalaya publications.

Write cql query to display the total cost of books purchased Publisher wise.

```
Cqlsh>  Select * from Book where publication='Himalaya';
```

```
cqlsh > Select Publisher, Sum(Price) From Book GROUP BY Publisher ;
```

4. Write cql query to count the total number of books under Kalyani publications.

Write cql query to rename the column Publisher as Publications

```
cqlsh > Select Publisher, Count(*) From Book where Publisher = 'Kalyani' ;
cqlsh > ALTER TABLE Book RENAME Publisher TO  'Publications';
```

5. Write a cql query to display the books in the ascending order of DatePurchased

```
cqlsh > Select * from Book ORDER BY DatePurchased ASC;
```

6. Write cql query to display the books whose price is between 500 and 700

```
cqlsh > Select * From Book where Price BETWEEN 500 AND 700;
```

7. Write cql query to increase the price of all the books by 200 for publishers other than Himalaya or Kalyani

```
cqlsh > UPDATE Book SET Price= Price +200
where Publishers NOT IN ('Himalaya','Kalyani' ;
```

8. Write cql query to display the book details where author name contains the name Sharma

```
cqlsh > Select * From Book where Author LIKE '%Sharma'
```

Day-17:

Installation of NoSQL Database - Neo4j on Windows

2. Installation of NoSQL Database - Neo4j on Linux

1. Installation On Ubuntu 20.04

2.Open to terminal (Ctrl+T)

3 Update your system: Update your system's app repositories

4 Import the GPG key: Import the GPG key used to sign Neo4j packages

5 Add the Neo4j repository: Add the Neo4j repository to your system

6 Install Neo4j: Install Neo4j using the command sudo apt-get install neo4j

7 Start the Neo4j service: Start the Neo4j service

8 Check the status: Check the status of the Neo4j service

9 Access Neo4j: Access Neo4j by opening your web browser and visiting http://localhost:7474

3. NoSQL Database - Neo4j Environment Setup
Setting up a Neo4j environment involves a few steps, depending on your operating system1
. Here's a general guide to get you started:
System Requirements
Ensure your system meets the requirements for Neo4j1
. You'll need:
Java Runtime Environment (JRE): Neo4j requires Java to run2
Disk Space: At least 1GB of free disk space
Memory: Minimum 2GB of RAM, but more is recommended for better performance
Installation Steps
Download Neo4j: Go to the Neo4j download page and download the appropriate installer for your OS (Windows, macOS, Linux)
Install Neo4j: Run the installer and follow the prompts2
. Make sure to give the installer permission to make changes to your computer2
Run Neo4j: After installation, you can start the Neo4j server2
. You can do this from the start menu or by running the neo4j command in your terminal.
Using Neo4j Desktop

If you prefer a more integrated environment, you can use Neo4j Desktop
. It provides a graphical interface to manage your databases and run queries

4. Neo4j CQL commands.

Creating Nodes:

```
CREATE (n:Person {name: "John", age: 30})
```

Creating Relationships:

```
MATCH (a:Person {name: "John"}), (b:Person {name: "Jane"})

CREATE (a)-[:FRIENDS_WITH]->(b)
```

Querying Nodes:

```
MATCH (n:Person)

RETURN n
```

Updating Nodes:

```
MATCH (n:Person {name: "John"})

SET n.age = 31
```

Deleting Nodes:

```
MATCH (n:Person {name: "John"})

DELETE n
```

Using WHERE Clauses:

```
MATCH (n:Person)

WHERE n.age > 25

RETURN n
```

5. Neo4j CQL operators

Comparison Operators:

```
MATCH (n:Person {name: "John"})

RETURN n
```

<>: Not Equals

```
MATCH (n:Person)
```

```
WHERE n.name <> "John"
RETURN n
```

<: Less Than

```
MATCH (n:Person)
WHERE n.age < 30
RETURN n
```

<=: Less Than or Equal To

```
MATCH (n:Person)
WHERE n.age <= 30
RETURN n
```

AND: Logical AND

```
MATCH (n:Person)
WHERE n.age > 30 AND n.name = "John"
RETURN n
```

OR: Logical OR:

```
MATCH (n:Person)
WHERE n.age > 30 OR n.name = "John"
RETURN n
```

String Operators:

CONTAINS: String contains

```
MATCH (n:Person)
WHERE n.name CONTAINS "hn"
RETURN n
```

ENDS WITH: String ends with: MATCH (n:Person)

WHERE n.name ENDS WITH "hn"

RETURN n

String starts with

MATCH (n:Person)

WHERE n.name STARTS WITH "Jo"

RETURN n

Pattern Matching Operators:

=: Matches pattern

MATCH (n:Person)-[:FRIENDS_WITH]->(m)

RETURN n, m

IN: Checks if element is in a collection:

MATCH (n:Person)

WHERE n.name IN ["John", "Jane", "Jim"]

RETURN n

6. Neo4j CQL Write Class

Writing to the database in Neo4j involves using Cypher commands to create, update, or delete nodes and relationships. Here's a quick look at some essential write operations:

Creating Nodes

CREATE (n:Person {name: "Alice", age: 35})

Creating Relationships

MATCH (a:Person {name: "Alice"}), (b:Person {name: "Bob"})

```
CREATE (a)-[:KNOWS]->(b)
```

Updating Nodes

```
MATCH (n:Person {name: "Alice"})
SET n.age = 36
```

<u>Deleting Nodes and Relationships</u>

```
MATCH (n:Person {name: "Alice"})
DETACH DELETE n
```

Merging Nodes:

```
MERGE (n:Person {name: "Alice"})
ON CREATE SET n.created = timestamp()
ON MATCH SET n.lastSeen = timestamp()
```

7. Neo4j CQL Read Class

Basic Match and Return

```
MATCH (n:Person)
RETURN n
```

Specifying Properties:

```
MATCH (n:Person {name: "Alice"})
RETURN n
```

Using WHERE Clauses:

```
MATCH (n:Person)
WHERE n.age > 30
RETURN n
```

Return Specific Properties

```
MATCH (n:Person)

RETURN n.name, n.age
```

Pattern Matching and Relationships:

```
MATCH (a:Person)-[r:KNOWS]->(b:Person)

RETURN a, b, r
```

Using Aggregations

```
MATCH (n:Person)

RETURN n.age, count(n) AS count
```

Limiting Results

```
MATCH (n:Person)

RETURN n

LIMIT 5
```

Ordering Results:

```
MATCH (n:Person)

RETURN n.name, n.age

ORDER BY n.age DESC
```

Day-18:

1. **Creating Nodes using Neo4j CQL**
   ```
   CREATE (n:Person {name: "Alice", age: 35})
   ```

2. **Creating Relationships using Neo4j CQL**

```
MATCH (a:Person {name: "Alice"}), (b:Person {name: "Bob"})
CREATE (a)-[:KNOWS]->(b)
```

3. **Create a node Student (std,sname,cid) and display the Node**

```
CREATE (s:Student {std: "1", sname: "John Doe", cid: "101"})
RETURN s
```

4. **Create a node Course (cid,cname) and display the Node**

```
CREATE (c:Course {cid: "CS101", cname: "Introduction to Computer Science"})

RETURN c
```

5. **Display all the Nodes in the database**

```
MATCH (n)
RETURN n
```

6. **Create a relationship between Student and Course based on cid and display the relationship**

```
MATCH (s:Student), (c:Course)

WHERE s.cid = c.cid

CREATE (s)-[r:ENROLLED_IN]->(c)

RETURN s, r, c
```

7. **Display the Student whose name is “Sai” and cid is 539**

```
MATCH (s:Student {sname: "Sai", cid: "539"})

RETURN s
```

8. **Display the Student whose cid is exist in Course**

```
MATCH (s:Student)

WHERE EXISTS {

   MATCH (c:Course)

   WHERE s.cid = c.cid
}
RETURN s
```

Day-19:

1. Create a node Employee (eno,ename,dno) and display it

```
CREATE (e:Employee {eno: "E123", ename: "Alex Martin", dno: "D456"})
RETURN e
```

2. Create a node Dept (dno,dname,add) and display it

```
CREATE (d:Dept {dno: "D456", dname: "Human Resources", add: "123 Main
Street"})
RETURN d
```

3. Create a relationship between Employee and Dept based on dno and display the Node

```
MATCH (e:Employee), (d:Dept)
WHERE e.dno = d.dno
CREATE (e)-[r:WORKS_IN]->(d)
RETURN e, r, d
```

4. **Display the Student whose name is “Sai” and cid is 539**

```
MATCH (s:Student {sname: "Sai", cid: "539"})
RETURN s
```

5. Create a node Artist (Name), Album (Name,Released). Then create a relationship Artist   Released Album

```
CREATE (a:Artist {Name: "John Doe"})

RETURN a


CREATE (al:Album {Name: "Greatest Hits", Released: 2024})

RETURN al


MATCH (a:Artist {Name: "John Doe"}), (al:Album {Name: "Greatest Hits"})

CREATE (a)-[r:RELEASED]->(al)

RETURN a, r, al
```

6. Create a node Person (Name), and create a relationship for a Person is produced and Performed in album

```
CREATE (p:Person {Name: "Jane Smith"})
RETURN p

MATCH (p:Person {Name: "Jane Smith"}), (al:Album {Name: "Greatest
Hits"})
CREATE (p)-[:PRODUCED]->(al)
CREATE (p)-[:PERFORMED_IN]->(al)
RETURN p, al
```

7 Create a relationship for a Person is produced ,Performed in album and plays in Artist

```
MATCH (p:Person {Name: "Jane Smith"}), (al:Album {Name: "Greatest
Hits"}), (ar:Artist {Name: "John Doe"})
CREATE (p)-[:PRODUCED]->(al)
CREATE (p)-[:PERFORMED_IN]->(al)
CREATE (p)-[:PLAYS_IN]->(ar)
RETURN p, al, ar
```

Day-20:

1. Name,Age,City
John,30,New York
Jane,25,Los Angeles (CSV FILE)

Load the Data:

```
LOAD CSV WITH HEADERS FROM 'file:///yourfile.csv' AS row
CREATE (p:Person {name: row.Name, age: toInteger(row.Age), city:
row.City})
RETURN p
```

2.

```
std,sname,cid
1,John Doe,101
2,Jane Smith,102
3,Sai Kumar,103   (student.csv)
```

Load the Data:

```
LOAD CSV WITH HEADERS FROM 'file:///students.csv' AS row
CREATE (s:Student {std: row.std, sname: row.sname, cid: row.cid})
RETURN s
```

3.

```
cid,cname
CS101,Introduction to Computer Science
CS102,Data Structures and Algorithms
CS103,Database Systems     (Course.csv)
```

```
LOAD CSV WITH HEADERS FROM 'file:///courses.csv' AS row
CREATE (c:Course {cid: row.cid, cname: row.cname})
RETURN c
```

5. Create a node by importing Employee (eno,ename,dno) and display it
   eno,ename,dno
   E123,Alex Martin,D456
   E124,Maria Garcia,D457
   E125,David Kim,D458 (employees.csv)

```
LOAD CSV WITH HEADERS FROM 'file:///employees.csv' AS row
CREATE (e:Employee {eno: row.eno, ename: row.ename, dno: row.dno})
RETURN e
```

6.
departments.csv':
dno,dname,add
D456,Human Resources,123 Main Street
D457,Finance,456 Elm Street
D458,Engineering,789 Oak Street

```
LOAD CSV WITH HEADERS FROM 'file:///departments.csv' AS row
CREATE (d:Dept {dno: row.dno, dname: row.dname, add: row.add})
RETURN d
```

6. Create a relationship between Employee and Dept based on dno and display the Node

```
MATCH (e:Employee), (d:Dept)

WHERE e.dno = d.dno

CREATE (e)-[r:WORKS_IN]->(d)

RETURN e, r, d
```

7. Write a program to count the number of occurrences of a word using MapReduce

```
pip install mrjob


from mrjob.job import MRJob
import re

WORD_RE = re.compile(r"\w+")

class MRWordCount(MRJob):

    def mapper(self, _, line):
        words = WORD_RE.findall(line)
        for word in words:
            yield (word.lower(), 1)

    def reducer(self, word, counts):
        yield (word, sum(counts))

if __name__ == "__main__":
    MRWordCount.run()
```