ADVANCED DATABASE MANAGEMENT SYSTEMS

LAB ASSIGNMENT-1

SUBMITTED BY:

GROUP-VI

NIHAL K (MCA226)

NITHIN RAJ (MCA227)

PRAJITHA P (MCA228)

RASIKA V V (MCA229)

RESHMA KRISHNAN (MCA230)

TOPIC: Design database schemas and implement min 10 queries using Cassandra column based databases

Aim
To Design database schemas and implement queries using Cassandra databases

Objective(s)	
1	Study of NOSQL Cassandra.
2	Study the procedure to execute a query using Apache Cassandra.
3	Execute min 10 queries using Cassandra column based database.

1. STUDY OF NOSQL CASSANDRA

➢ CASSANDRA

Apache Cassandra is an open source distributed database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure.

FEATURES

i. **Scalability:**

Read and write throughput both increase linearly as new machines are added, with no downtime or interruption to applications.

ii. **Fault-tolerant:**

- Data is automatically replicated to multiple nodes for fault-tolerance.
- Replication across multiple data centers is supported.
- Failed nodes can be replaced with no downtime.

iii. **MapReduce support:**

Cassandra has Hadoop integration, with MapReduce support.

iv. **Query language:**

Cassandra introduces CQL (Cassandra Query Language), a SQL-like alternative to the traditional RPC interface.

v. **Keyspace:**

Keyspace is the outermost container for data. It is similar to the schema in a relational database.

Basic attributes of Keyspace are:

- **Replication Factor:** It is the number of machines in the cluster that will receive copies of the same data
- **Replica Placement Strategy:** It is a strategy to place replicas in the ring
- Simple Strategy,
- Old Network Topology Strategy
- Network Topology Strategy

➢ COLUMN FAMILIES:

Column family is a NoSQL object that contains columns of related data. It is a tuple (pair) that consists of a key-value pair, where the key is mapped to a value that is a set of columns. It is similar to a table in a relational database and each key-value pair being a row.

Each column is a tuple (triplet) consisting of

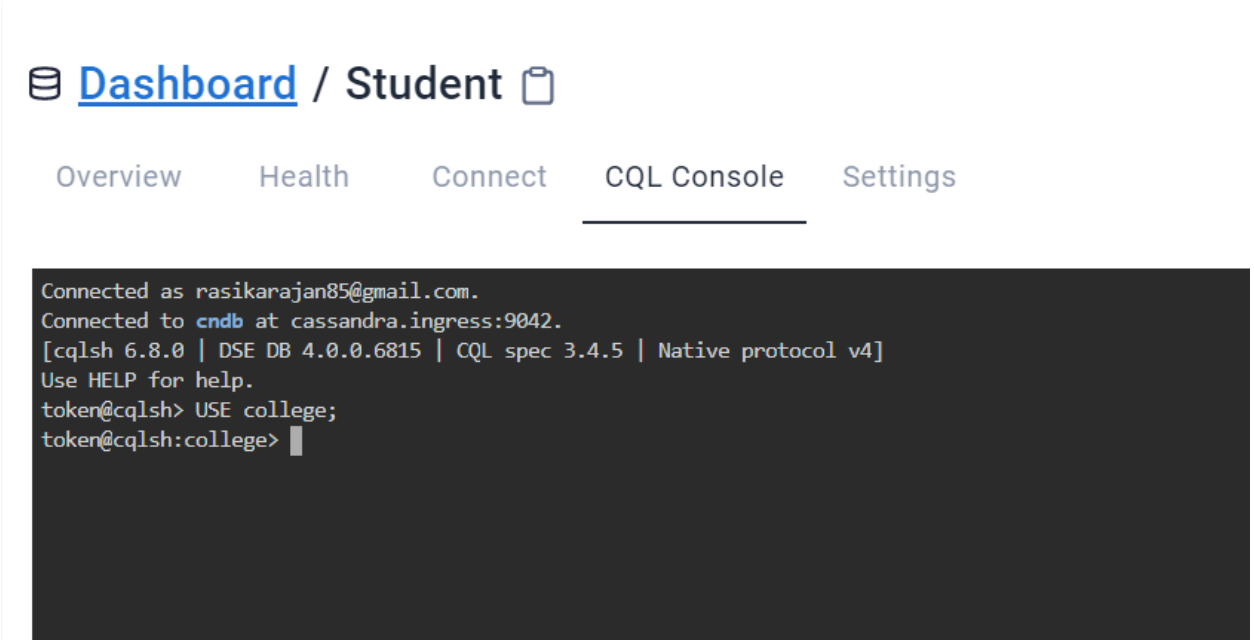
- Column name
- Value
- Timestamp

2. STUDY THE PROCEDURE TO EXECUTE A QUERY USING APACHE CASSANDRA

FUNCTION	SYNTAX
CREATING KEYSPACE:	❖ Create KEYSPACE is a statement used to create a KEYSPACE in Cassandra. A KEYSPACE in Cassandra is a KEYSPACE or a collection of tables. The syntax for this statement is as follows: **cqlsh>CREATE KEYSPACE ABC userdb replication={ 'class':'SimpleStrategy','replication_factor':'1'};** ❖ Here, IF NOT EXISTS is an optional clause, which notifies the user that a database with the same name already exists. We can use SCHEMA in place of DATABASE in this command.
CREATING A DATABASE:	❖ The following query is executed to create a database named userdb: **cqlsh> userdb;** **(OR)** **cqlsh> CREATE SCHEMA userdb;**
ALTERING A TABLE:	❖ Given below is the syntax for creating a table. **ALTER (TABLE | COLUMNFAMILY) <tablename> <instruction>**
CREATING DATA IN A TABLE:	❖ Given below is the syntax for creating data in a table. **INSERT INTO <tablename>** **(<column1 name>, <column2 name>....)** **VALUES (<value1>, <value2>....)** **USING <option>**
UPDATING DATA IN A TABLE:	❖ Given below is the syntax of UPDATE command − **UPDATE <tablename>** **SET <column name> = <new value>** **<column name> = <value>....** **WHERE <condition>**

READING DATA USING SELECT CLAUSE:	❖ Given below is the syntax of SELECT clause. **SELECT FROM <tablename>**
DELETING DATA FROM A TABLE:	❖ Given below is the syntax: **DELETE FROM <identifier> WHERE <condition>;**
DROPPING A TABLE:	❖ Given below is the syntax: **DROP TABLE <tablename>**

3. EXECUTE MINIMUM 10 QUERIES USING CASSANDRA COLUMN BASED DATABASE

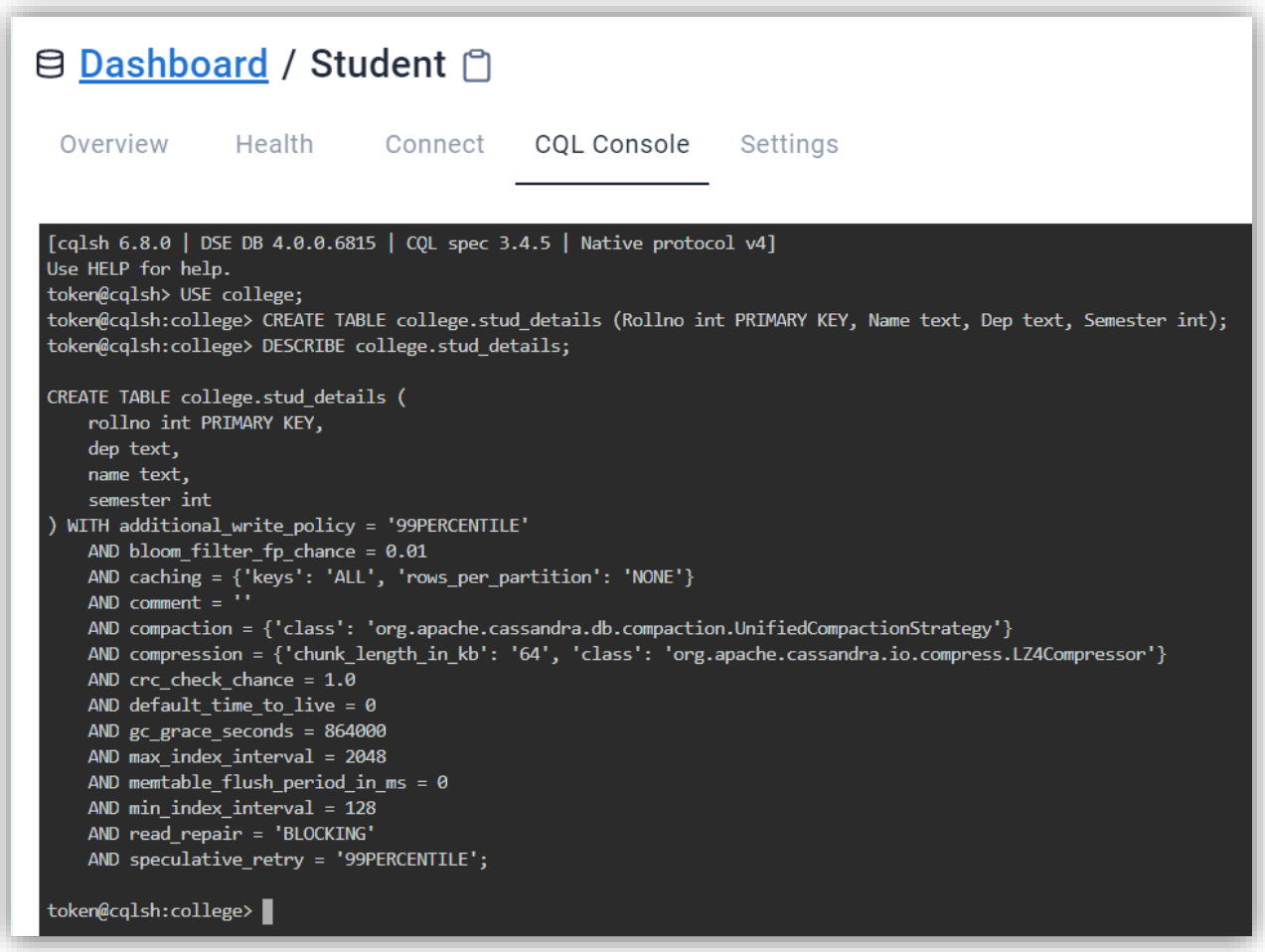


1. **CREATING A TABLE:**

Procedure:

```
CREATE TABLE college.stud_details (
rollno int PRIMARY KEY,
dep text,
name text,
semester int
)
```

Output:



2. **DATA INSERTION:**

Procedure:

INSERT INTO stud_details (Rollno,Name,Dep,Semester) values(101,'Michael','CS',2)
INSERT INTO stud_details (Rollno,Name,Dep,Semester) values(102,'Joy','BCOM',2)
INSERT INTO stud_details (Rollno,Name,Dep,Semester) values(103,'Avin','MECH',4)

Output:

```
token@cqlsh:college> INSERT INTO stud_details (Rollno,Name,Dep,Semester) values(101,'Michael','CS',2);
token@cqlsh:college> INSERT INTO stud_details (Rollno,Name,Dep,Semester) values(102,'Joy','BCOM',2);
token@cqlsh:college> INSERT INTO stud_details (Rollno,Name,Dep,Semester) values(103,'Avin','MECH',4);
token@cqlsh:college> INSERT INTO stud_details (Rollno,Name,Dep,Semester) values(104,'Jack','CS',2);
token@cqlsh:college> INSERT INTO stud_details (Rollno,Name,Dep,Semester) values(105,'Sam','MECH',6);
token@cqlsh:college>
```

3. DISPLAYING THE ENTIRE TABLE CONTENTS:

Procedure:

SELECT * FROM college.stud_details;

Output:

```
token@cqlsh:college> INSERT INTO stud_details (Rollno,Name,Dep,Semester) values(101,'Michael','CS',2);
token@cqlsh:college> INSERT INTO stud_details (Rollno,Name,Dep,Semester) values(102,'Joy','BCOM',2);
token@cqlsh:college> INSERT INTO stud_details (Rollno,Name,Dep,Semester) values(103,'Avin','MECH',4);
token@cqlsh:college> INSERT INTO stud_details (Rollno,Name,Dep,Semester) values(104,'Jack','CS',2);
token@cqlsh:college> INSERT INTO stud_details (Rollno,Name,Dep,Semester) values(105,'Sam','MECH',6);
token@cqlsh:college> SELECT * FROM college.stud_details;

 rollno | dep  | name    | semester
--------+------+---------+----------
    105 | MECH |     Sam |        6
    104 |   CS |    Jack |        2
    102 | BCOM |     Joy |        2
    101 |   CS | Michael |        2
    103 | MECH |    Avin |        4

(5 rows)
token@cqlsh:college>
```

4. DISPLAYING TABLE CONTENTS USING ‘WHERE’ CLAUSE:

Procedure:

SELECT * from college.stud_details WHERE Dep =’CS’ ALLOW FILTERING;

Output:

```
token@cqlsh:college> SELECT * from college.stud_details WHERE Dep='CS' ALLOW FILTERING;

 rollno | dep | name    | semester
--------+-----+---------+----------
    104 |  CS |    Jack |        2
    101 |  CS | Michael |        2

(2 rows)
token@cqlsh:college>
```

5. **TABLE UPDATION:**

Procedure:

UPDATE college.stud_details SET Name = 'Avinash' WHERE Rollno=103;

Output:

```
token@cqlsh:college> UPDATE college.stud_details SET Name='Avinash' WHERE Rollno=103;
token@cqlsh:college> SELECT * FROM college.stud_details;

 rollno | dep  | name    | semester
--------+------+---------+----------
    105 | MECH |     Sam |        6
    104 |   CS |    Jack |        2
    102 | BCOM |     Joy |        2
    101 |   CS | Michael |        2
    103 | MECH | Avinash |        4

(5 rows)
token@cqlsh:college>
```

6. **ALTERING THE TABLE:**

Procedure:

ALTER TABLE college.stud_details ADD Mark int;

Output:

```
token@cqlsh:college> ALTER TABLE college.stud_details ADD Mark int;
token@cqlsh:college> SELECT * FROM college.stud_details;

 rollno | dep  | mark | name    | semester
--------+------+------+---------+----------
    105 | MECH | null |     Sam |        6
    104 |   CS | null |    Jack |        2
    102 | BCOM | null |     Joy |        2
    101 |   CS | null | Michael |        2
    103 | MECH | null | Avinash |        4

(5 rows)
token@cqlsh:college>
```

7. DROPPING A COLUMN:

Procedure:

ALTER TABLE college.stud_details DROP Mark;

Output:

```
token@cqlsh:college> ALTER TABLE college.stud_details DROP Mark;
token@cqlsh:college> SELECT * FROM college.stud_details;

 rollno | dep  | name    | semester
--------+------+---------+----------
    105 | MECH |     Sam |        6
    104 |   CS |    Jack |        2
    102 | BCOM |     Joy |        2
    101 |   CS | Michael |        2
    103 | MECH | Avinash |        4

(5 rows)
token@cqlsh:college>
```

8. RETRIEVING TIMESTAMPS:

Timestamps:

Each time we write data into Cassandra, a timestamp is generated for each column value that is updated. Internally, Cassandra uses these timestamps for resolving any conflicting changes that are made to the same value.

Procedure:

SELECT Rollno, Name, writetime(Name) FROM stud_details;

Output:

```
token@cqlsh:college> SELECT Rollno,Name,writetime(Name) FROM stud_details;

 rollno | name    | writetime(name)
--------+---------+------------------
    105 |     Sam | 1630429021375831
    104 |    Jack | 1630428951838381
    102 |     Joy | 1630428827179346
    101 | Michael | 1630428791519440
    103 | Avinash | 1630430640227531

(5 rows)
token@cqlsh:college>
```

9. **DELETING FROM THE TABLE:**

Procedure:

DELETE FROM college.stud_details WHERE rollno=105;

Output:

```
token@cqlsh:college> DELETE FROM college.stud_details WHERE rollno=105;
token@cqlsh:college> SELECT * FROM college.stud_details;

 rollno | dep  | name    | semester
--------+------+---------+----------
    104 |   CS |    Jack |        2
    102 | BCOM |     Joy |        2
    101 |   CS | Michael |        2
    103 | MECH | Avinash |        4

(4 rows)
token@cqlsh:college>
```

10. QUERY TO ADD ANOTHER EMAIL ADDRESS WITHOUT REPLACING THE WHOLE SET BY USING CONCATENATION

ALTER TABLE college.stud_details ADD Emails set<text>;
UPDATE stud_details SET emails={'jack12@gmail.com'} WHERE Rollno=104;

```
token@cqlsh:college> ALTER TABLE college.stud_details ADD Emails set<text>;
token@cqlsh:college> UPDATE stud_details SET emails={'jack12@gmail.com'} WHERE Rollno=104;
token@cqlsh:college> SELECT Emails from stud_details WHERE Rollno=104;

 emails
---------------------
 {'jack12@gmail.com'}

(1 rows)
token@cqlsh:college>
```

Procedure:

UPDATE stud_details SET Emails+{'jackjohn@gmail.com} WHERE Rollno=104;

Output:

```
token@cqlsh:college> UPDATE stud_details SET Emails=Emails+{'jackjohn@gmail.com'} WHERE Rollno=104;
token@cqlsh:college> SELECT Emails from stud_details WHERE Rollno=104;

 emails
-------------------------------------------
 {'jack12@gmail.com', 'jackjohn@gmail.com'}

(1 rows)
token@cqlsh:college>
```