Ex. No: 1

Create a database and collection using MongoDB environment. For example a document collection meant for analysing Restaurant records can have fields like restaurant_id, restaurant_name, customer_name, locality, date, cuisine, grade, comments. etc. Create database using INSERT and UPDATE Command.

Ans:

Create a Restaurant database with the fields: (restaurant_id, restaurant_name, customer_name, locality, date, cuisine, grade, comments)

 Database Name: Restaurant

 Collection Name: Restaurant_Records

> use Restaurant

switched to db Restaurant

1. Insert Document:

Using insert you can either insert one document or array of documents

> db.Restaurant_Records.insert({ re

staurant_id: "AEPY06",

restaurant_name: "do it yourself",

customer_name:"abhyash",

locality:"pune",

date:03/03/2023,

cuisine:"korean",

grade:"5 star",

comments: "Good"

})

> **db.Restaurant_Records.insert({ re**

staurant_id: "AEPY07",

restaurant_name: "Spice Magic",

customer_name:"Venkat",

locality:"Mumbai",

date:03/05/2022,

cuisine:"Chinese",

grade:"4 star",

comments: "Excellent"

})

>**db.Restaurant_Records.insertMany([**

{

restaurant_id: "AEPY08",

restaurant_name: "Adurs",

customer_name:"Prabhu",

locality:"Vijayawada",

date:03/05/2021,

cuisine:"Indian",

grade:"4 star",

comments: "Excellent"

},

{

restaurant_id: "AEPY09",

restaurant_name: "chandrika",

customer_name:"Ram",

locality:"Bhimavaram",

date:03/03/2022,

cuisine:"Indian",

grade:"5 star",

comments: "Excellent"

}

]);

MongoDB query to display all the documents in the collection Restaurant_Records.

> db.Restaurant_Records.find();

Output:

{ "_id" : ObjectId("64797bc50655b1cad1d7789d"), "restaurant_id" : "AEPY06", "restaurant_name" :

"do it yourself", "customer_name" : "abhyash", "locality" : "pune", "date" : 0.0004943153732081067,

"cuisine" : "korean", "grade" : "5 star", "comments" : "Good" }

{ "_id" : ObjectId("64797bc50655b1cad1d7789e"), "restaurant_id" : "AEPY07", "restaurant_name" :

"Spice Magic", "customer_name" : "Venkat", "locality" : "Mumbai", "date" : 0.0002967359050445104,

"cuisine" : "Chinese", "grade" : "4 star", "comments" : "Excellent" }

{ "_id" : ObjectId("64797bc50655b1cad1d7789f"), "restaurant_id" : "AEPY08", "restaurant_name" :

"Adurs", "customer_name" : "Prabhu", "locality" : "Vijayawada", "date" : 0.00029688273132112816,

"cuisine" : "Indian", "grade" : "4 star", "comments" : "Excellent" }

{ "_id" : ObjectId("64797bc50655b1cad1d778a0"), "restaurant_id" : "AEPY09", "restaurant_name" :

"chandrika", "customer_name" : "Ram", "locality" : "Bhimavaram", "date" : 0.0004945598417408506,

"cuisine" : "Indian", "grade" : "5 star", "comments" : "Excellent" }

10

2. Updating documents

db.collection.update() : Updates one or more than one document(s) in collection based on

matching document and based on multi option

db.Restaurant_Records.update({'restaurant_id' : "AEPY07"},{$set:{'restaurant_name':'Salem

Briyani Restaurant'}})

Output:

>db.Restaurant_Records.find();

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

{ "_id" : ObjectId("64797e29f81746b9c76c7c5d"), "restaurant_id" : "AEPY06", "restaurant_name" : "do

it yourself", "customer_name" : "abhyash", "locality" : "pune", "date" : 0.0004943153732081067,

"cuisine" : "korean", "grade" : "5 star", "comments" : "Good" }

{ "_id" : ObjectId("64797e29f81746b9c76c7c5e"), "restaurant_id" : "AEPY07", "restaurant_name" :

"Salem Briyani Restaurant", "customer_name" : "Venkat", "locality" : "Mumbai", "date" :

0.0002967359050445104, "cuisine" : "Chinese", "grade" : "4 star", "comments" : "Excellent" }

{ "_id" : ObjectId("64797e29f81746b9c76c7c5f"), "restaurant_id" : "AEPY08", "restaurant_name" :

"Adurs", "customer_name" : "Prabhu", "locality" : "Vijayawada", "date" : 0.00029688273132112816,

"cuisine" : "Indian", "grade" : "4 star", "comments" : "Excellent" }

{ "_id" : ObjectId("64797e29f81746b9c76c7c60"), "restaurant_id" : "AEPY09", "restaurant_name" :

"chandrika", "customer_name" : "Ram", "locality" : "Bhimavaram", "date" : 0.0004945598417408506,

"cuisine" : "Indian", "grade" : "5 star", "comments" : "Excellent" }

EXERCISE-2:

Create a database and collection with some fields and perform simple MongoDB queries suchas displaying all the records, display selected records with conditions.

ANS:
To Practice Simple MongoDB queries such as displaying all the records, display selected records with conditions

Program:

Insert Document

>**db.Restaurant_Records.insertMany([{**

restaurant_id: "AEPY07",

restaurant_name: "Spice Magic",

customer_name:"Venkat",

locality:"Mumbai",

date:03/05/2022,

cuisine:"Chinese",

grade:"4 star",

comments: "Excellent"

},

{

restaurant_id: "AEPY08",

restaurant_name: "Adurs",

customer_name:"Prabhu",

locality:"Vijayawada",

date:03/05/2021,

cuisine:"Indian",

grade:"4 star",

comments: "Excellent"

},

{

restaurant_id: "AEPY09",

restaurant_name: "chandrika",

customer_name:"Ram",

locality:"Bhimavaram",

date:03/03/2022,

cuisine:"Indian",

grade:"5 star",

comments: "Excellent"

},

{

restaurant_id: "AEPY06",

restaurant_name: "do it yourself",

customer_name:"abhyash",

locality:"pune",

16

date:03/03/2023,

cuisine:"korean",

grade:"5 star",

comments: "Good"

}

])

Queries

1. Write a MongoDB query to display all the documents in the collection restaurants.

>**db.restaurants.find();**

Output:

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("647998c388f1b5a8bbfe7efc"),

ObjectId("647998c388f1b5a8bbfe7efd"),

ObjectId("647998c388f1b5a8bbfe7efe"),

ObjectId("647998c388f1b5a8bbfe7eff")

]

}

{ "_id" : ObjectId("647998c388f1b5a8bbfe7efc"), "restaurant_id" : "AEPY07",

"restaurant_name" : "Spice Magic", "customer_name" : "Venkat", "locality" : "Mumbai", "date" :

0.0002967359050445104, "cuisine" : "Chinese", "grade" : "4 star", "comments" : "Excellent" }

{ "_id" : ObjectId("647998c388f1b5a8bbfe7efd"), "restaurant_id" : "AEPY08",

"restaurant_name" : "Adurs", "customer_name" : "Prabhu", "locality" : "Vijayawada", "date" :

0.00029688273132112816, "cuisine" : "Indian", "grade" : "4 star", "comments" : "Excellent" }

{ "_id" : ObjectId("647998c388f1b5a8bbfe7efe"), "restaurant_id" : "AEPY09",

"restaurant_name" : "chandrika", "customer_name" : "Ram", "locality" : "Bhimavaram", "date" :

0.0004945598417408506, "cuisine" : "Indian", "grade" : "5 star", "comments" : "Excellent" }

{ "_id" : ObjectId("647998c388f1b5a8bbfe7eff"), "restaurant_id" : "AEPY06",

"restaurant_name" : "do it yourself", "customer_name" : "abhyash", "locality" : "pune", "date" :

0.0004943153732081067, "cuisine" : "korean", "grade" : "5 star", "comments" : "Good" }

2. Write a MongoDB query to display the fields restaurant_id, restaurant_name, and cuisine for all the documents in the collection restaurant.

> db.Restaurant_Records.find({},{"restaurant_id" : 1,"restaurant_name":1,"cuisine" :1});

Output:

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("647999dc5f862d472d99f416"),

ObjectId("647999dc5f862d472d99f417"),

ObjectId("647999dc5f862d472d99f418"),

ObjectId("647999dc5f862d472d99f419")

]

}

{ "_id" : ObjectId("647999dc5f862d472d99f416"), "restaurant_id" : "AEPY07",

"restaurant_name" : "Spice Magic", "cuisine" : "Chinese" }

{ "_id" : ObjectId("647999dc5f862d472d99f417"), "restaurant_id" : "AEPY08",

"restaurant_name" : "Adurs", "cuisine" : "Indian" }

{ "_id" : ObjectId("647999dc5f862d472d99f418"), "restaurant_id" : "AEPY09",

"restaurant_name" : "chandrika", "cuisine" : "Indian" }

{ "_id" : ObjectId("647999dc5f862d472d99f419"), "restaurant_id" : "AEPY06",

"restaurant_name" : "do it yourself", "cuisine" : "korean" }

3. Write a MongoDB query to display the restaurant which is in the cuisine Indian.

> db.Restaurant_Records.find({"cuisine": "Indian"});

Output:

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("64799aeb1ff0d3240df31074"),

ObjectId("64799aeb1ff0d3240df31075"),

ObjectId("64799aeb1ff0d3240df31076"),

ObjectId("64799aeb1ff0d3240df31077")

]

}

{ "_id" : ObjectId("64799aeb1ff0d3240df31075"), "restaurant_id" : "AEPY08",

"restaurant_name" : "Adurs", "customer_name" : "Prabhu", "locality" : "Vijayawada", "date" :

0.00029688273132112816, "cuisine" : "Indian", "grade" : "4 star", "comments" : "Excellent" }

{ "_id" : ObjectId("64799aeb1ff0d3240df31076"), "restaurant_id" : "AEPY09",

"restaurant_name" : "chandrika", "customer_name" : "Ram", "locality" : "Bhimavaram", "date" :

0.0004945598417408506, "cuisine" : "Indian", "grade" : "5 star", "comments" : "Excellent" }

EXPERIMENT-3:

To perform MongoDB comparison and logical query operators - $gt, $gte, $lt, $lte, $in,

#nin,$ne, $and, $or, $not

Ans:

$gt : It is used to match values of the fields that are greater than a specified value

$gte: It is used to match values of the fields that are greater than equal to the specified value

$lt : It is used to match values of the fields that are less than a specified valueo

$lte : It is used to match values of the fields that are less than equals to the specified value

$in : It is used to match any of the values specified in an array.

$nin : It is used to match none of the values specified in an array.

$ne: It is used to match all values of the field that are not equal to a specified value
$and: Performs a logical AND operation on an array of expressions.
$or: Performs a logical OR operation on an array of expressions.
$not: Performs a logical NOT operation on a specified expression.

In the following examples, we are working with:

Database: Sample

Collection: contributor

Document: three documents that contain the details of the contributors in the form of field-value pairs.

>use Sample

1. **Switched to db Sample**

>**db.contributor.find()**

{

"_id":Objectid("5e6f7a6692e6dfa3fc48ddbe"),

"name":"Rohit",

"branch":"CSE",

"joiningYear":2018,

"language":["C#","Python","Java"],

"personal":{"contactinfo":0,"state":"Delhi","age":24,"semesterMarks":[70,73.3,76.5,78.6] },

"salary":1000

}

{

"_id":Objectid("5e7b9f0a92e6dfa3fc48ddbf"),

"name":"Amit",

"branch""ECE",

"joiningYear":2017,

"language":["Python","C#"],

"personal":{"contactinfo":234556789,"state":"UP","age":25,"semesterMarks":[80,80.1,98,70 },

"salary":10000

}

{

"_id":Objectid("5e7b9f0a92e6dfa3fc48ddc0"),

"name":"Sumit",

"branch":"CSE",

"joiningYear":2017,

"language":["Java","Perl"],

"personal":{"contactinfo":2300056789,"state":"MP","age":24,"semesterMarks":[89,80.1,78,71]},

"salary":15000

}

1. Matching values using $nin operator:

In this example, we are retrieving only those employee's documents whose name is not Amit or Suman.

>db.contributor.find({name: {$nin: ["Amit", "Suman"]}}).pretty()

{

"_id":Objectid("5e6f7a6692e6dfa3fc48ddbe"),

"name":"Rohit",

"branch":"CSE",

"joiningYear":2018,

"language":["C#","Python","Java"],

"personal":{"contactinfo":0,"state":"Delhi","age":24,"semesterMarks":[70,73.3,76.5,78.6] },

"salary":1000

}

{

"_id":Objectid("5e7b9f0a92e6dfa3fc48ddc0"),

"name":"Sumit",

"branch":"CSE",

"joiningYear":2017,

"language":["Java","Perl"],

"personal":{"contactinfo":2300056789,"state":"MP","age":24,"semesterMarks":[89,80.1,78,71]},

"salary":15000

}

20

2. Matching values using $in operator:

In this example, we are retrieving only those employee's documents whose name is either Amit or Suman.

>db.contributor.find({name: {$in: ["Amit", "Suman"]}}).pretty()

{

"_id":Objectid("5e7b9f0a92e6dfa3fc48ddbf"),

"name":"Amit",

"branch""ECE",

"joiningYear":2017,

"language":["Python","C#"],

"personal":{"contactinfo":234556789,"state":"UP","age":25,"semesterMarks":[80,80.1,98,70 },

"salary":10000

}

3. Matching values using $lt operator:

In this example, we are selecting those documents where the value of the salary field is less than 2000.

>db.contributor.find({salary: {$lt: 2000}}).pretty()

{
"_id":Objectid("5e6f7a6692e6dfa3fc48ddbe"),
"name":"Rohit",
"branch":"CSE",
"joiningYear":2018,
"language":["C#","Python","Java"],
"personal":{"contactinfo":0,"state":"Delhi","age":24,"semesterMarks":[70,73.3,76.5,78.6] },
"salary":1000
}

4. Matching values using $eq operator:

In this example, we are selecting those documents where the value of the branch field is equal to CSE.

>db.contributor.find({branch: {$eq: "CSE"}}).pretty()

{
"_id":Objectid("5e6f7a6692e6dfa3fc48ddbe"),
"name":"Rohit",
"branch":"CSE",
"joiningYear":2018,
"language":["C#","Python","Java"],
"personal":{"contactinfo":0,"state":"Delhi","age":24,"semesterMarks":[70,73.3,76.5,78.6] },
"salary":1000
}
{
"_id":Objectid("5e7b9f0a92e6dfa3fc48ddc0"),
"name":"Sumit",

"branch":"CSE",
"joiningYear":2017,
"language":["Java","Perl"],
"personal":{"contactinfo":2300056789,"state":"MP","age":24,"semesterMarks":[89,80.1,78,71]},
"salary":15000

}

5. Matching values using $ne operator:

In this example, we are selecting those documents where the value of the branch field is not equal to

CSE.

>db.contributor.find({branch: {$ne: "CSE"}}).pretty()

{

"_id":Objectid("5e7b9f0a92e6dfa3fc48ddbf"),

"name":"Amit",

"branch""ECE",

"joiningYear":2017,

"language":["Python","C#"],

"personal":{"contactinfo":234556789,"state":"UP","age":25,"semesterMarks":[80,80.1,98,70 },

"salary":10000

}

6. Matching values using $gt operator:

In this example, we are selecting those documents where the value of the salary field is greater than

1000.

>db.contributor.find({salary: {$gt: 1000}}).pretty()

{

"_id":Objectid("5e7b9f0a92e6dfa3fc48ddbf"),

"name":"Amit",

"branch""ECE",

"joiningYear":2017,

"language":["Python","C#"],

"personal":{"contactinfo":234556789,"state":"UP","age":25,"semesterMarks":[80,80.1,98,70 },

"salary":10000

}

{

"_id":Objectid("5e7b9f0a92e6dfa3fc48ddc0"),

"name":"Sumit",

"branch":"CSE",

"joiningYear":2017,

"language":["Java","Perl"],

"personal":{"contactinfo":2300056789,"state":"MP","age":24,"semesterMarks":[89,80.1,78,71]},

"salary":15000

}

22

7. Matching values using $gte operator:

In this example, we are selecting those documents where the value of the joining Year field is greater

than equals to 2018.

>db.contributor.find({joiningYear: {$gte: 2018}})

{

"_id":Objectid("5e6f7a6692e6dfa3fc48ddbe"),

"name":"Rohit",

"branch":"CSE",

"joiningYear":2018,

"language":["C#","Python","Java"],

"personal":{"contactinfo":0,"state":"Delhi","age":24,"semesterMarks":[70,73.3,76.5,78.6] },

"salary":1000}

8. Less Than or Equal To ($lte) Operator:

operator selects documents where the value of a field is less than or equal to a specified value.

=> > **db.contributor.find({salary: {$lte: 2000}}).pretty()**

9. **Logical AND ($and) Operator**:

This operator performs a logical AND operation on an array of two or more expressions and selects the documents that satisfy all the expressions.

=> **> db.contributor.find({$and: [{name: "Amit"}, {branch: "ECE"}]}).pretty()**

10. Logical NOT ($not) Operator:

This operator performs a logical NOT operation on the specified expression and selects the documents that do not match the expression.

=> **> db.contributor.find({name: {$not: {$eq: "Rohit"}}}).pretty()**

11. **Logical OR ($or) Operator**:

This operator performs a logical OR operation on an array of two or more expressions and selects the documents that satisfy at least one of the expressions.

> **db.contributor.find({$or: [{name: "Amit"}, {name: "Sumit"}]}).pretty()**

EXPERIMENT: 4

To perform MongoDB array based and evaluation query operators -$exists, $type,

$mod, $regex, $all, $elemMatch, $size

Ans:

Example documents:

Database: Sample

Collection: contributor

Document: three documents that contain the details of the contributors in the form of field-value pairs.

> use Sample

1.Switched to db Sample

>db.contributor.insertMany([

{

name: "John",

age: 30,

address: { city: "New York", state: "NY" }

},

{

name: "Jane",

age: 25

},

{

name: "Bob",

age: 40, address:

"bvrm"

}

]);

Query examples:

1. $exists: find all documents where the address field exists

db.contributor.find({ address: { $exists: true } })

Output:

{ "_id" : ObjectId("6479a0ff2615b529741554d8"), "name" : "John", "age" : 30, "address" :

{"city" : "New York", "state" : "NY" } }

{ "_id" : ObjectId("6479a0ff2615b529741554da"), "name" : "Bob", "age" : 40, "address" :

"bvrm" }

2. $type: find all documents where the address field is of type "object"

db.users.find({ address: { $type: "object" } })

Output:

{ "_id" : ObjectId("6479a0ff2615b529741554d8"), "name" : "John", "age" : 30, "address" :

{"city" : "New York", "state" : "NY" } }

3.$mod Operator: mod operator allows the user to get those documents from a collection where

the specific field when divided by a divisor has an even or odd remainder. This operator works like the WHERE clause of the SQL programming language. The $mod operator works only on the integer values and here is what the query syntax will look like.

Syntax

> db.collection_name.find({ <field_name>: { $mod: [divisor, remainder] } })

Where:

 field_name is the attribute name on which the documents are fetched from a collection

 divisor and remainder are the input arguments to perform a modulo operation

Query 1:

>db.editors.find({ age: { $mod: [5, 0] } }).pretty()

{

"acknowledged" : true,

"insertedIds" : [

"1001",

"1002",

"1003",

"1004",

"1005",

"1006",

"1007",

"1008",

"1009",

"1010"

]

}

{

"_id" : "1004",

"name" : {

"first" : "John",

"last" : "Gordon"

},

"age" : 20

}

{

"_id" : "1005",

"name" : {

"first" : "Rick",

"last" : "Ford"

},

"age" : 25,

"grades" : {

"JavaCodeGeek" : "A+",

"WebCodeGeek" : "A+",

"DotNetCodeGeek" : "A"

}

}

{

"_id" : "1008",

"name" : {

"first" : "Arya",

"last" : "Stark"

},

"age" : 25

}

4.$regex Operator:

In the Mongo universe, the $regex operator allows the user to get those documents from a

collection where a string matches a specified pattern.

Syntax:

db.collection_name.find({ <field_name>: { $regex: /pattern/, $options: '<options>' } })

Where:

 field_name is the attribute name on which the documents are fetched from a collection

 A pattern is a regular expression for a complex search

Query:

>db.editors.find({ "name.first" : { $regex: 'A.*' } }).pretty()

Output:

{

"acknowledged" : true,

"insertedIds" : ["1001","1002","1003","1004","1005","1006",

"1007","1008","1009","1010"

]

}

{

"_id" : "1008",

"name" : {

"first" : "Arya",

"last" : "Stark"

},

"age" : 25

}

{

"_id" : "1009",

"name" : {

"first" : "April",

"last" : "Paul"

},

"age" : 28,

"grades" : {

"JavaCodeGeek" : "A+",

"WebCodeGeek" : "A",

"DotNetCodeGeek" : "A+"

}

}

Array Operators

- ⇨ **Array operators in MongoDB are used to query documents that include arrays. The array operators are**

1. $all

2. $size

3. $elemMatch

Example documents:

{ name: "John", hobbies: ["reading", "hiking"] }

{ name: "Jane", hobbies: ["swimming", "yoga"] }

{ name: "Bob", hobbies: null }

Query examples:

1. $all:

find all documents where hobbies include both "reading" and "hiking"

db.users.find({ hobbies: { $all: ["reading", "hiking"] } })

2.$elemMatch:

selects documents if element(s) in an array field match the specified conditions.

db.users.find({

$elemMatch : { name: “John”, hobbies:[“reading”, “hiking]}

})

2. **$size:**

selects documents if the array field is a specified size.

db.users.find({hobbies : {$size:2}})

EXPERIMENT-5

To create database in Cassandra using – Create, Alter, Drop and Truncate commands

How to run Cassandra .. ?

Goto Downloads > open Cassandra folder > open bin folder

> right click on whitespace and open terminal there means at bin folder open terminal

> Run Cassandra now by : .\cassandra it runs continuously don't close it

> now open another terminal at the bin folder now run cqlsh by : .\cqlsh

Now you are ready use cqlsh.

CREATE

```
cqlsh> CREATE keyspace cloudduggu
   ... WITH REPLICATION = {'class':'SimpleStrategy', 'replication_factor' : 1};
```

```
cqlsh> describe keyspaces;
```

ALTER

```
cqlsh> describe cloudduggu;
cqlsh> ALTER keyspace cloudduggu
   ... WITH REPLICATION = {'class':'NetworkTopologyStrategy', 'datacenter1' : 2
,'replication_factor' : 2};
cqlsh> describe cloudduggu;
```

DROP

Command:

```
cqlsh> describe keyspaces;
cqlsh> DROP keyspace cloudduggu;
cqlsh> describe keyspaces;
```

TRUNCATE

```
cqlsh> USE cloudduggu;
```

```
cqlsh> CREATE TABLE cloudduggu.emp_detail (
   emp_id INT PRIMARY KEY,
   emp_first_name TEXT,
   emp_last_name TEXT,
   emp_state TEXT,
   zip INT
) ;

cqlsh> INSERT INTO cloudduggu.emp_detail (emp_id, emp_first_name, emp_last_name
, emp_state, zip  ) VALUES (1001, 'Ram','Kumar','MP', 564567);
```

```
cqlsh:cloudduggu> select * from Cloudduggu.emp_name;
```

```
cloudduggu@ubuntu: ~/apache-cassandra-4.0.1/bin
cqlsh> TRUNCATE  TABLE  Cloudduggu.emp_name;
cqlsh> select * from Cloudduggu.emp_name;

 id | first_name | last_name
----+------------+-----------

(0 rows)
cqlsh>
```

Exercise -6 To create database in Cassandra using – Create, Insert, Update, Delete commands

```
cqlsh> USE cloudduggu;
```

```
cqlsh> CREATE TABLE cloudduggu.emp_detail (
   emp_id INT PRIMARY KEY,
   emp_first_name TEXT,
   emp_last_name TEXT,
   emp_state TEXT,
   zip INT
) ;

cqlsh> INSERT INTO cloudduggu.emp_detail (emp_id, emp_first_name, emp_last_name
, emp_state, zip  ) VALUES (1001, 'Ram','Kumar','MP', 564567);
```

```
cqlsh> SELECT * FROM cloudduggu.emp_detail;
```

```
cqlsh> SELECT * FROM cloudduggu.emp_detail;
cqlsh> UPDATE cloudduggu.emp_detail SET emp_state='AP' WHERE emp_id=1001;
cqlsh> SELECT * FROM cloudduggu.emp_detail;
```

Command:

```
cqlsh> SELECT * FROM cloudduggu.emp_detail;
cqlsh> DELETE FROM cloudduggu.emp_detail WHERE emp_id=1001;
cqlsh> SELECT * FROM cloudduggu.emp_detail;
```

Exercise -7 To create a database in Cassandra and performing queries such as selecting records, select records with specific condition

CREATE A KEYSPACE
USE KEYSPACE
CREATE A TABLE
INSERT SOME DATA INTO TABLE

Command:

```
cqlsh> SELECT * FROM cloudduggu.emp_detail;
```

Verify Output:

```
cloudduggu@ubuntu: ~/apache-cassandra-4.0.1/bin
cqlsh> Select * from Cloudduggu.emp_detail;

 emp_id | emp_first_name | emp_last_name | emp_state | zip
--------+----------------+---------------+-----------+--------
   1004 |        Animesh |         Dutta |        JK | 923487
   1005 |          Manoj |          Gopa |       GUJ | 563456
   1001 |            Ram |         Kumar |        MP | 564567
   1003 |           Ritu |           Raj |       MAH | 345234
   1002 |          Mohan |        Sharma |        UP | 786789

(5 rows)
cqlsh>
```

Select Specific Column From Table

We can read particular columns data by mentioning only those columns in the select list. In t below example we will access only **emp_first_name** and **emp_last_name** columns of the tab **Cloudduggu.emp_detail**.

Command:

```
cqlsh> SELECT emp_first_name, emp_last_name FROM cloudduggu.emp_detail;
```

Verify Output:

```
cloudduggu@ubuntu: ~/apache-cassandra-4.0.1/bin
cqlsh> Select emp_first_name, emp_last_name from Cloudduggu.emp_detail;

 emp_first_name | emp_last_name
----------------+---------------
        Animesh |         Dutta
          Manoj |          Gopa
            Ram |         Kumar
           Ritu |           Raj
          Mohan |        Sharma

(5 rows)
cqlsh>
```

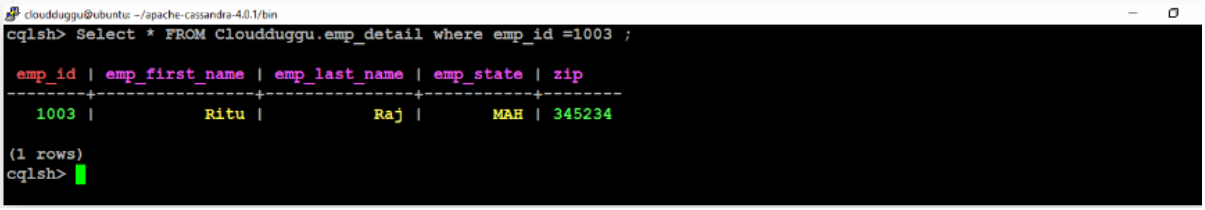
Select Data with Where Clause

Using the **Where Clause** in the select command, we can restrict the row access and fetch onl required rows. In the below example, we will fetch the record of an employee havin empid=1003.

Command:

```
cqlsh> SELECT * FROM cloudduggu.emp_detail WHERE emp_id =1003 ;
```

Verify Output:



Exercise -8

Design a Neo4j graph model for a social network database including nodes, relationships, and properties. Write a Cypher query to

▯ find all users who are friends with a specific user in a social network graph.

▯ find common friends between two specific users in the social network

▯ find all pairs of users who are friends with the same set of users.

▯ Group users by age and count the number of users in each age group.

```
// Creating nodes for users
    CREATE (:User {name: "Alice"})
    CREATE (:User {name: "Bob"})
    CREATE (:User {name: "Charlie"})
    CREATE (:User {name: "David"})
    CREATE (:User {name: "Eve"})
    CREATE (:User {name: "Frank"})
    CREATE (:User {name: "Grace"})

// Creating FRIEND relationships between users
    MATCH (a:User {name: "Alice"}), (b:User {name: "Bob"})
    CREATE (a)-[:FRIEND]->(b)
```

```
MATCH (a:User {name: "Alice"}), (c:User {name: "Charlie"})
CREATE (a)-[:FRIEND]->(c)
MATCH (b:User {name: "Bob"}), (c:User {name: "Charlie"})
CREATE (b)-[:FRIEND]->(c)
MATCH (c:User {name: "Charlie"}), (d:User {name: "David"})
CREATE (c)-[:FRIEND]->(d)
MATCH (a:User {name: "Alice"}), (e:User {name: "Eve"})
CREATE (a)-[:FRIEND]->(e)
MATCH (b:User {name: "Bob"}), (d:User {name: "David"})
CREATE (b)-[:FRIEND]->(d)
MATCH (c:User {name: "Charlie"}), (f:User {name: "Frank"})
CREATE (c)-[:FRIEND]->(f)
MATCH (d:User {name: "David"}), (g:User {name: "Grace"})
CREATE (d)-[:FRIEND]->(g)
```

// Query to find all friends of a specific user

```
MATCH (specific_user:User {name: "Alice"})-[:FRIEND]-(friend:User)
RETURN friend.name AS friend_name
```

// Query to find common friend

```
MATCH (u1:User {name: "Alice"})-[:FRIEND]-(friend:User)-[:FRIEND]-(u2:User {name:
"Bob"})
RETURN friend.name AS common_friend
```

// Query to find users with more friends

```
MATCH (u:User)-[:FRIEND]-()
RETURN u.name AS user_name, COUNT(*) AS num_friends
ORDER BY num_friends DESC
LIMIT 1
```

```
// Query to find users by age group
        MATCH (u:User)
        RETURN CASE
            WHEN u.age <= 20 THEN 'Under 20'
            WHEN u.age <= 30 THEN '20-30'
            WHEN u.age <= 40 THEN '31-40'
            ELSE 'Over 40'
          END AS age_group,
          COUNT(*) AS num_users
```

Exercise - 9

Design a Neo4j graph model for a movie database including nodes, relationships, and properties. Write a Cypher query to

⍰ Find Movies Directed by a Specific Director.

⍰ Find Actors Who Acted in a Specific Movie.

⍰ Find Movies Released in a Specific Year.

⍰ Find Actors Who Worked with a Specific Director.

⍰ Find Actors Who Worked

```
// Creating nodes for movies
CREATE (:Movie {title: "Inception", release_year: 2010, genre: "Sci-Fi", rating: 8.8})
CREATE (:Movie {title: "The Dark Knight", release_year: 2008, genre: "Action", rating: 9.0})
CREATE (:Movie {title: "Interstellar", release_year: 2014, genre: "Sci-Fi", rating: 8.6})

// Creating nodes for actors
CREATE (:Actor {name: "Leonardo DiCaprio", birth_date: "1974-11-11", nationality: "American"})
CREATE (:Actor {name: "Tom Hardy", birth_date: "1977-09-15", nationality: "British"})
CREATE (:Actor {name: "Anne Hathaway", birth_date: "1982-11-12", nationality: "American"})

// Creating nodes for directors
CREATE (:Director {name: "Christopher Nolan", birth_date: "1970-07-30", nationality: "British"})
CREATE (:Director {name: "James Cameron", birth_date: "1954-08-16", nationality: "Canadian"})

// Creating relationships
MATCH (a:Actor {name: "Leonardo DiCaprio"}), (m1:Movie {title: "Inception"})
CREATE (a)-[:ACTED_IN]->(m1)

MATCH (a:Actor {name: "Tom Hardy"}), (m1:Movie {title: "Inception"})
CREATE (a)-[:ACTED_IN]->(m1)

MATCH (a:Actor {name: "Leonardo DiCaprio"}), (m2:Movie {title: "The Dark Knight"})
CREATE (a)-[:ACTED_IN]->(m2)

MATCH (a:Actor {name: "Tom Hardy"}), (m2:Movie {title: "The Dark Knight"})
```

```
CREATE (a)-[:ACTED_IN]->(m2)

MATCH (a:Actor {name: "Anne Hathaway"}), (m3:Movie {title: "Interstellar"})
CREATE (a)-[:ACTED_IN]->(m3)

MATCH (d:Director {name: "Christopher Nolan"}), (m:Movie)
WHERE m.title IN ["Inception", "The Dark Knight", "Interstellar"]
CREATE (d)-[:DIRECTED]->(m)

MATCH (d:Director {name: "James Cameron"}), (m:Movie {title: "Avatar"})
CREATE (d)-[:DIRECTED]->(m)
```

Query to Find Movies Directed by a Specific Director:

```
MATCH (d:Director {name: "Christopher Nolan"})-[:DIRECTED]->(m:Movie)
RETURN m.title AS movie_title
```

Query to Find Actors Who Acted in a Specific Movie:

```
MATCH (a:Actor)-[:ACTED_IN]->(m:Movie {title: "Inception"})
RETURN a.name AS actor_name
```

Query to Find Movies Released in a Specific Year:

```
MATCH (m:Movie)
WHERE m.release_year = 2000
RETURN m.title AS movie_title
```

Query to Find Actors Who Worked with a Specific Director:

```
MATCH (d:Director {name: "Christopher Nolan"})-[:DIRECTED]->(m:Movie)<-[:ACTED_IN]-
(a:Actor)
RETURN DISTINCT a.name AS actor_name
```

Query to Find Actors Who Worked with Multiple Directors:

```
MATCH (a:Actor)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(d:Director)
WITH a, COUNT(DISTINCT d) AS num_directors
WHERE num_directors > 1
RETURN a.name AS actor_name, num_directors
```