



G H Raison College of Engineering  
& Management, Wagholi, Pune



**Department of Computer Engineering**

**D-19**

**Lab Manual**  
**(2022-23)**

**Class: TY Computer Term: VI**

**UCOP305: Unstructured Database Management  
LAB**

**Faculty Name: Mrs. Sunita Nandgave**



# G H Raisoni College of Engineering & Management, Wagholi, Pune



## Department of Computer Engineering

### Database Management Systems (BCOP19303)

#### Course Outcome

<b>CO1</b>	To Perform CRUD operations (create, read, update and delete) on data in NoSQL environment
<b>CO2</b>	To Define, compare and use the four types of NoSQL Databases (Document-oriented, Key Value Pairs, Column-oriented and Graph).
<b>CO3</b>	To Demonstrate an understanding of the detailed architecture, define objects, load data, query data and performance tune Column-oriented NoSQL databases.
<b>CO4</b>	To Perform CRUD operations (create, read, update and delete) on data in NoSQL environment

#### List of Experiment

<b>Sr. No.</b>	<b>Name of Experiment</b>	<b>CO Mapped</b>
<b>1</b>	Installation and setup of MongoDB client and server	<b>CO1, CO2</b>
<b>2</b>	Create a database and collection using MongoDB environment. For example, a document collection meant for analyzing Restaurant records can have fields like restaurant_id, restaurant_name, customer_name, locality, date, cuisine, grade, comments. etc. Create database using INSERT, UPDATE, UPSERTS, DELETE and INDEX. Practice writing simple MongoDB queries such as displaying all the records, display selected records with conditions	<b>CO1, CO2</b>
<b>3</b>	Experiment with MongoDB comparison and logical query operators - \$gt, \$gte, \$lt, \$lte, \$in, \$nin, \$ne, \$and, \$or, \$not	<b>CO3</b>
<b>4</b>	Practice exercise on element, array based and evaluation query operators - \$exists, \$type, \$mod, \$regex	<b>CO3</b>
<b>5</b>	Exercise on MongoDB shell commands and user management	<b>CO2, CO3</b>
<b>6</b>	Installation and configuration of Cassandra. Find out two use cases where Cassandra is preferred over MongoDB	<b>CO2, CO3</b>
<b>7</b>	Create database in Casandra using - Create, Alter and Drop. Add records using Inset, Update, Delete and Truncate.	<b>CO4</b>
<b>8</b>	Exercise based on Cassandra Query Language i.e. selecting records, select records with specific conditions	<b>CO4</b>
<b>Content Beyond Syllabus</b>		
<b>9</b>	Open ended practical Mini-project	<b>CO2, CO3</b>

# Practical NO 01

**Aim:** Installation of MongoDB.

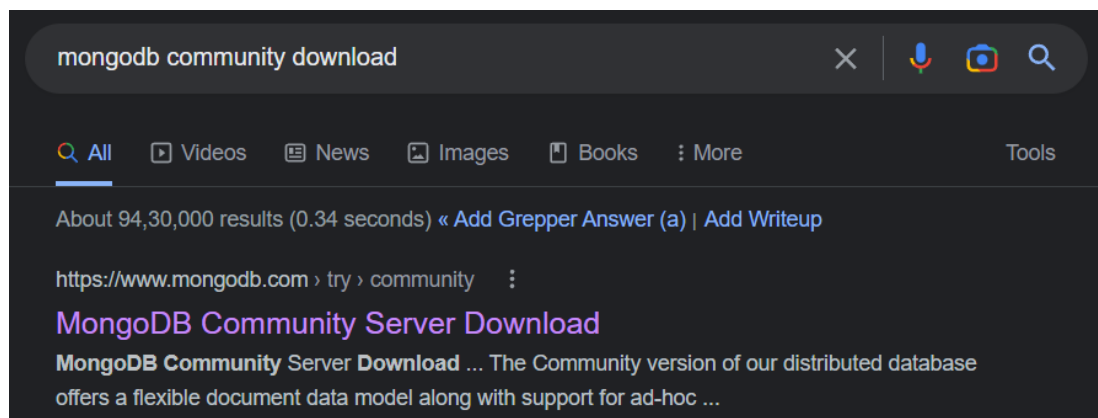
## Theory:

MongoDB is an open-source document oriented database and leading NoSQL database. MongoDB is written in C++. This tutorial will give you great understanding on MongoDB concepts needed to create and deploy a highly scalable and performance-oriented database.

### Step to install MongoDB:

- 1 Search for “Mongodb download” and go for first link.
- 2 Select the Operating system as Windows and Package as zip and then download
- 3 Extract the zip. Mongodb folder will be appear.
4. Open that extracted setup folder.
5. Copy all files under “bin” folder.
4. Create folder on your machine with name “mongodb”.
5. Paste all files under that folder.
6. Create folder with name “data” in “mongodb” folder.
- 7 Create one more folder with name “db” in “data” folder.
- 8 Start mongodb server:  
Open command prompt and navigate to the “mongodb” folder and run “mongod”. It will start your mongodb server.
- 9 Connect to the server:  
Open another command prompt and navigate to the “mongodb” folder and run “mongo”. It will connect to your mongodb server and you can check or perform the db related activites here.

### Screenshots:



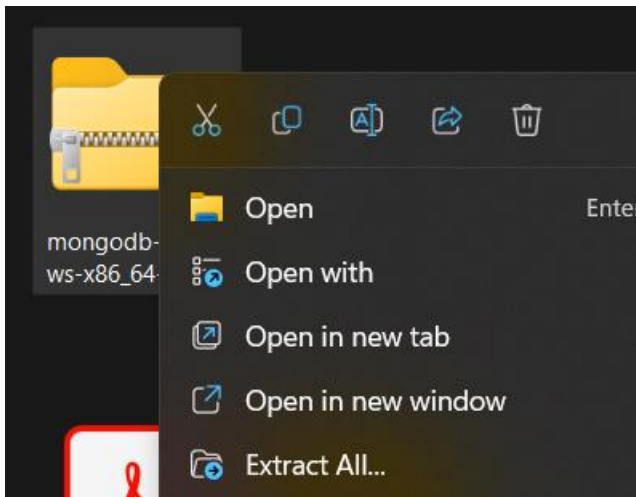
Version  
4.4.18








Platform  
Windows

Package  
zip

Download 

More Options 



Downloads > mongodb-windows-x86_64-4.4.18 > mongodb-win32-x86_64-windows-4.4.18 > bin				
Name	Date modified	Type	Size	
Today				
 mongos.pdb	28-12-2022 11:33 AM	PDB File	2,67,476 KB	
 mongod.pdb	28-12-2022 11:33 AM	PDB File	3,90,124 KB	
 mongos	28-12-2022 11:33 AM	Application	27,107 KB	
 mongod	28-12-2022 11:33 AM	Application	37,945 KB	
 mongo.pdb	28-12-2022 11:33 AM	PDB File	1,79,172 KB	
 Install-Compass	28-12-2022 11:33 AM	Windows PowerSh...	2 KB	
 mongo	28-12-2022 11:33 AM	Application	21,112 KB	

```
Microsoft Windows [Version 10.0.22621.963]
(c) Microsoft Corporation. All rights reserved.
```

```
C:\>mkdir mongodb
```

```
C:\>cd mongodb
```

```
C:\>mkdir "data/db"
```

```
C:\mongodb>mongod
{"t":{"$date":"2022-12-28T11:41:10.133+05:30"},"s":"I", "c":"CONTROL", "id":23285, "ctx":"main","msg":"Automatically
disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2022-12-28T11:41:10.635+05:30"},"s":"I", "c":"NETWORK", "id":4648602, "ctx":"main","msg":"Implicit TCP
FastOpen in use."}
{"t":{"$date":"2022-12-28T11:41:10.637+05:30"},"s":"I", "c":"STORAGE", "id":4615611, "ctx":"initandlisten","msg":"Mong
oDB starting", "attr":{"pid":1968, "port":27017, "dbPath":"C:/data/db/", "architecture":"64-bit", "host":"Shubham-Laptop"}}
{"t":{"$date":"2022-12-28T11:41:10.637+05:30"},"s":"I", "c":"CONTROL", "id":23398, "ctx":"initandlisten","msg":"Targ
et operating system minimum version", "attr":{"targetMinOS":"Windows 7/Windows Server 2008 R2"}}
{"t":{"$date":"2022-12-28T11:41:10.638+05:30"},"s":"I", "c":"CONTROL", "id":23403, "ctx":"initandlisten","msg":"Buil
d Info", "attr":{"buildInfo":{"version":"4.4.18", "gitVersion":"8ed32b5c2c68ebe7f8ae2ebe8d23f36037a17dea", "modules":[], "al
locator":"tcmalloc", "environment":{"distmod":"windows", "distarch":"x86_64", "target_arch":"x86_64"}}}}
{"t":{"$date":"2022-12-28T11:41:10.638+05:30"},"s":"I", "c":"CONTROL", "id":51765, "ctx":"initandlisten","msg":"Oper
ating System", "attr":{"os":{"name":"Microsoft Windows 10", "version":"10.0 (build 22621)"}}}
{"t":{"$date":"2022-12-28T11:41:10.638+05:30"},"s":"I", "c":"CONTROL", "id":21951, "ctx":"initandlisten","msg":"Opti
ons set by command line", "attr":{"options":{}}}
{"t":{"$date":"2022-12-28T11:41:10.641+05:30"},"s":"I", "c":"STORAGE", "id":22315, "ctx":"initandlisten","msg":"Open
ing WiredTiger", "attr":{"config":{"create, cache_size=3253M, session_max=33000, eviction=(threads_min=4, threads_max=4), confi
g_base=false, statistics=(fast), log=(enabled=true, archive=true, path=journal, compressor=snappy), file_manager=(close_idle_t
ime=100000, close_scan_interval=10, close_handle_minimum=250), statistics_log=(wait=0), verbose=[recovery_progress, checkpoin
t_progress, compact_progress], "}}}
```

```
C:\mongodb>mongo
MongoDB shell version v4.4.18
connecting to: mongod://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongod
Implicit session: session { "id" : UUID("78c28fdb-4267-40cc-9a53-0279d7981468") }
MongoDB server version: 4.4.18
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
  https://community.mongodb.com
---
The server generated these startup warnings when booting:
  2022-12-28T11:41:10.720+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
  2022-12-28T11:41:10.721+05:30: This server is bound to localhost. Remote systems will be unable to connect to this server. Start the server with --b
ind_ip <address> to specify which IP addresses it should serve responses from, or with --bind_ip_all to bind to all interfaces. If this behavior is desired,
start the server with --bind_ip 127.0.0.1 to disable this warning
---
  Enable MongoDB's free cloud-based monitoring service, which will then receive and display
  metrics about your deployment (disk utilization, CPU, operation statistics, etc).

  The monitoring data will be available on a MongoDB website with a unique URL accessible to you
  and anyone you share the URL with. MongoDB may use this information to make product
  improvements and to suggest MongoDB products and deployment options to you.

  To enable free monitoring, run the following command: db.enableFreeMonitoring()
  To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> |
```

## Conclusion:

Hence, with this practical we able to install and run mongodb in our machine

# Practical NO 02

**Aim:** Create a database and collection using MongoDB environment. For example, a document collection meant for analyzing Restaurant records can have fields like restaurant\_id, restaurant\_name, customer\_name, locality, date, cuisine, grade, comments. etc. Create database using INSERT, UPDATE, UPSERTS, DELETE and INDEX. Practice writing simple MongoDB queries such as displaying all the records, display selected records with conditions

## Theory:

### CRUD Operations On MongoDB:

#### 1. Create Operations –

The create or insert operations are used to insert or add new documents in the collection. If a collection does not exist, then it will create a new collection in the database. You can perform, create operations using the following methods provided by the MongoDB:

Method:

- a. db.collection.insertOne() :  
It is used to insert a single document in the collection
- b. db.collection.insertMany() :  
It is used to insert multiple documents in the collection
- c. db.createCollection():  
It is used to create an empty collection

#### 2. Read Operations –

The Read operations are used to retrieve documents from the collection, or in other words, read operations are used to query a collection for a document. You can perform read operation using the following method provided by the MongoDB:

- a. db.collection.find() :  
it is used to retrieve documents from the collection

#### 3. Update Operations –

The update operations are used to update or modify the existing document in the collection. You can perform update operations using the following methods provided by the MongoDB:

- a. db.collection.updateOne() :  
It is used to update a single document in the collection that satisfy the given criteria
- b. db.collection.updateMany() :

1. Create database for restaurant

```
> use restaurantDB
switched to db restaurantDB
> db
restaurantDB
> |
```

2. Create Collection for restaurant

```
> db.createCollection("restaurant")
{ "ok" : 1 }
```

3. Performing INSERT, UPDATE, UPSERTS, DELETE and INDEX commands on restaurant

```
> db.restaurant.insert({
... restaurant_id : 137,
... restaurant_name : "shubham's kichen",
... customer_names : ["sandeep", "himang", "vishal", "niyamat", "faizan", "tanishka", "prachi"],
... locality : "pune",
... grade : "A+",
... comments : ["best restaurant in pune", "Food Quality is good"]
... })
WriteResult({ "nInserted" : 1 })
```

```
> db.restaurant.update(
... {restaurant_id : 137}, {$set: {grade : "B+"}}
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

```
> db.restaurant.update(
... {restaurant_id : 140},
... {$set : { restaurant_name : "abc", locality: "mumbai" }},
... {upsert: true}
... )
WriteResult({
  "nMatched" : 0,
  "nUpserted" : 1,
  "nModified" : 0,
  "_id" : ObjectId("63ac43fe3bd403bc70bda012")
})
```

```
> db.restaurant.createIndex({"restaurant_id" : 1})
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
> |
```

```
> db.restaurant.deleteOne({restaurant_id: 140})
{ "acknowledged" : true, "deletedCount" : 1 }
> |
```

#### 4. Displaying all records

```
> db.restaurant.find().pretty()
{
  "_id" : ObjectId("63ac42e7d6638fce7be82b6c"),
  "restaurant_id" : 137,
  "restaurant_name" : "shubham's kichen",
  "customer_names" : [
    "sandeep",
    "himang",
    "vishal",
    "niyamat",
    "faizan",
    "tanishka",
    "prachi"
  ],
  "locality" : "pune",
  "grade" : "B+",
  "comments" : [
    "best restaurant in pune",
    "Food Quality is good"
  ]
}
```

#### 5. Displaying selected records with conditions

```
> db.restaurant.find(
... {
... locality: "pune"
... })
{ "_id" : ObjectId("63ac42e7d6638fce7be82b6c"), "restaurant_id" : 137, "restaurant_name" : "shubham's kichen", "customer_names" : [ "sandeep", "himang", "vishal", "niyamat", "faizan", "tanishka", "prachi" ], "locality" : "pune", "grade" : "B+", "comments" : [ "best restaurant in pune", "Food Quality is good" ] }
> |
```

**Conclusion:** Hence, we successfully created restaurant database in mongodb and performed create, read, update and delete on it.



## Practical NO 03

**Aim:** Experiment with MongoDB comparison and logical query operators - \$gt, \$gte, \$lt, \$lte, \$in, #nin, \$ne, \$and, \$or, \$not

### Theory:

#### Mongoddb Comparison Operators:

MongoDB uses various comparison query operators to compare the values of the documents. The following table contains the comparison query operators

1. \$eq: It is used to match the values of the fields that are equal to a specified value.

```
> db.students.find({city: {$eq: "latur"}})
{ "_id" : ObjectId("63ad2051bfff340c9213f2bbd"), "rollno" : 131, "name" : "prachi", "city" : "latur", "marks" : 92.93 }
{ "_id" : ObjectId("63ad21c9bfff340c9213f2bc3"), "rollno" : 116, "name" : "vidya", "city" : "latur", "marks" : 85.33 }
```

2. \$ne: It is used to match values of the field that are not equal to a specified value.

```
> db.students.find({city: {$ne: "latur"}})
{ "_id" : ObjectId("63ad2002bfff340c9213f2bba"), "rollno" : 128, "name" : "Himang", "city" : "bandara", "marks" : 89.99 }
{ "_id" : ObjectId("63ad201dbfff340c9213f2bbb"), "rollno" : 129, "name" : "Faizan", "city" : "pune", "marks" : 91.54 }
{ "_id" : ObjectId("63ad2034bfff340c9213f2bbc"), "rollno" : 130, "name" : "Niyamat", "city" : "pune", "marks" : 90.22 }
{ "_id" : ObjectId("63ad2070bfff340c9213f2bbe"), "rollno" : 133, "name" : "sami", "city" : "pune", "marks" : 71.33 }
{ "_id" : ObjectId("63ad209fbfff340c9213f2bbf"), "rollno" : 134, "name" : "sandeep", "city" : "mumbai", "marks" : 81.41 }
{ "_id" : ObjectId("63ad20d2bfff340c9213f2bc0"), "rollno" : 137, "name" : "shubham", "city" : "mumbai", "marks" : 39.8 }
{ "_id" : ObjectId("63ad20efbfff340c9213f2bc1"), "rollno" : 139, "name" : "tanishka", "city" : "jalna", "marks" : 93.33 }
{ "_id" : ObjectId("63ad2104bfff340c9213f2bc2"), "rollno" : 139, "name" : "vishal", "city" : "jalgaon", "marks" : 96.56,
"roll" : 140 }
{ "_id" : ObjectId("63ad2215bfff340c9213f2bc4"), "rollno" : 110, "name" : "vaibhav", "city" : "buldana", "marks" : 70 }
```

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

```
> db.students.find({marks: {$gt: 70.00}})
{ "_id" : ObjectId("63ad2002bfff340c9213f2bba"), "rollno" : 128, "name" : "Himang", "city" : "bandara", "marks" : 89.99 }
{ "_id" : ObjectId("63ad201dbfff340c9213f2bbb"), "rollno" : 129, "name" : "Faizan", "city" : "pune", "marks" : 91.54 }
{ "_id" : ObjectId("63ad2034bfff340c9213f2bbc"), "rollno" : 130, "name" : "Niyamat", "city" : "pune", "marks" : 90.22 }
{ "_id" : ObjectId("63ad2051bfff340c9213f2bbd"), "rollno" : 131, "name" : "prachi", "city" : "latur", "marks" : 92.93 }
{ "_id" : ObjectId("63ad2070bfff340c9213f2bbe"), "rollno" : 133, "name" : "sami", "city" : "pune", "marks" : 71.33 }
{ "_id" : ObjectId("63ad209fbfff340c9213f2bbf"), "rollno" : 134, "name" : "sandeep", "city" : "mumbai", "marks" : 81.41 }
{ "_id" : ObjectId("63ad20efbfff340c9213f2bc1"), "rollno" : 139, "name" : "tanishka", "city" : "jalna", "marks" : 93.33 }
{ "_id" : ObjectId("63ad2104bfff340c9213f2bc2"), "rollno" : 139, "name" : "vishal", "city" : "jalgaon", "marks" : 96.56,
"roll" : 140 }
{ "_id" : ObjectId("63ad21c9bfff340c9213f2bc3"), "rollno" : 116, "name" : "vidya", "city" : "latur", "marks" : 85.33 }
> |
```

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

```
> db.students.find({marks: {$gte: 70.00}})
{ "_id" : ObjectId("63ad2002bfff340c9213f2bba"), "rollno" : 128, "name" : "Himang", "city" : "bandara", "marks" : 89.99 }
{ "_id" : ObjectId("63ad201dbfff340c9213f2bbb"), "rollno" : 129, "name" : "Faizan", "city" : "pune", "marks" : 91.54 }
{ "_id" : ObjectId("63ad2034bfff340c9213f2bbc"), "rollno" : 130, "name" : "Niyamat", "city" : "pune", "marks" : 90.22 }
{ "_id" : ObjectId("63ad2051bfff340c9213f2bbd"), "rollno" : 131, "name" : "prachi", "city" : "latur", "marks" : 92.93 }
{ "_id" : ObjectId("63ad2070bfff340c9213f2bbe"), "rollno" : 133, "name" : "sami", "city" : "pune", "marks" : 71.33 }
{ "_id" : ObjectId("63ad209fbfff340c9213f2bbf"), "rollno" : 134, "name" : "sandeep", "city" : "mumbai", "marks" : 81.41 }
{ "_id" : ObjectId("63ad20efbfff340c9213f2bc1"), "rollno" : 139, "name" : "tanishka", "city" : "jalna", "marks" : 93.33 }
{ "_id" : ObjectId("63ad2104bfff340c9213f2bc2"), "rollno" : 139, "name" : "vishal", "city" : "jalgaon", "marks" : 96.56,
"roll" : 140 }
{ "_id" : ObjectId("63ad21c9bfff340c9213f2bc3"), "rollno" : 116, "name" : "vidya", "city" : "latur", "marks" : 85.33 }
{ "_id" : ObjectId("63ad2215bfff340c9213f2bc4"), "rollno" : 110, "name" : "vaibhav", "city" : "buldana", "marks" : 70 }
>
```

5. \$lt: It is used to match values of the fields that are less than a specified value.

```
> db.students.find({marks: {$lt: 70.00}})
{ "_id" : ObjectId("63ad20d2bfff340c9213f2bc0"), "rollno" : 137, "name" : "shubham", "city" : "mumbai", "marks" : 39.8 }
>
```

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

```
> db.students.find({marks: {$lte: 70.00}})
{ "_id" : ObjectId("63ad20d2bfff340c9213f2bc0"), "rollno" : 137, "name" : "shubham", "city" : "mumbai", "marks" : 39.8 }
{ "_id" : ObjectId("63ad2215bfff340c9213f2bc4"), "rollno" : 110, "name" : "vaibhav", "city" : "buldana", "marks" : 70 }
>
```

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

```
> db.students.find({city: {$in: ["latur", "jalna", "pune"]}})
{ "_id" : ObjectId("63ad201dbff340c9213f2bbb"), "rollno" : 129, "name" : "Faizan", "city" : "pune", "marks" : 91.54 }
{ "_id" : ObjectId("63ad2034bff340c9213f2bbc"), "rollno" : 130, "name" : "Niyamat", "city" : "pune", "marks" : 90.22 }
{ "_id" : ObjectId("63ad2051bff340c9213f2bbd"), "rollno" : 131, "name" : "prachi", "city" : "latur", "marks" : 92.93 }
{ "_id" : ObjectId("63ad2070bff340c9213f2bbe"), "rollno" : 133, "name" : "sami", "city" : "pune", "marks" : 71.33 }
{ "_id" : ObjectId("63ad209fbff340c9213f2bbf"), "rollno" : 134, "name" : "sandeep", "city" : "mumbai", "marks" : 81.41 }
{ "_id" : ObjectId("63ad20d2bff340c9213f2bc0"), "rollno" : 137, "name" : "shubham", "city" : "mumbai", "marks" : 39.8 }
{ "_id" : ObjectId("63ad20efbff340c9213f2bc1"), "rollno" : 139, "name" : "tanishka", "city" : "jalna", "marks" : 93.33 }
{ "_id" : ObjectId("63ad21c9bff340c9213f2bc3"), "rollno" : 116, "name" : "vidya", "city" : "latur", "marks" : 85.33 }
>
```

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

```
> db.students.find({city: {$nin: ["latur", "jalna", "pune"]}})
{ "_id" : ObjectId("63ad2002bff340c9213f2bba"), "rollno" : 128, "name" : "Himang", "city" : "bandara", "marks" : 89.99 }
{ "_id" : ObjectId("63ad209fbff340c9213f2bbf"), "rollno" : 134, "name" : "sandeep", "city" : "mumbai", "marks" : 81.41 }
{ "_id" : ObjectId("63ad2051bff340c9213f2bbd"), "rollno" : 131, "name" : "prachi", "city" : "latur", "marks" : 92.93 }
{ "_id" : ObjectId("63ad2070bff340c9213f2bbe"), "rollno" : 133, "name" : "sami", "city" : "pune", "marks" : 71.33 }
{ "_id" : ObjectId("63ad20d2bff340c9213f2bc0"), "rollno" : 137, "name" : "shubham", "city" : "mumbai", "marks" : 39.8 }
{ "_id" : ObjectId("63ad2104bff340c9213f2bc2"), "rollno" : 139, "name" : "vishal", "city" : "jalgaon", "marks" : 96.56,
"roll" : 140 }
{ "_id" : ObjectId("63ad21c9bff340c9213f2bc3"), "rollno" : 116, "name" : "vidya", "city" : "latur", "marks" : 85.33 }
{ "_id" : ObjectId("63ad2215bff340c9213f2bc4"), "rollno" : 110, "name" : "vaibhav", "city" : "buldana", "marks" : 70 }
>
```

## MongoDB logical Operators:

MongoDB supports logical query operators. These operators are used for filtering the data and getting precise results based on the given conditions.

The following table contains the comparison query operators:

1. \$and: It is used to join query clauses with a logical AND and return all documents that match the given conditions of the both clauses.

```
> db.students.find({$and: [{marks: {$gt: 70.00}}, {marks: {$lt: 90.00}}]})
{ "_id" : ObjectId("63ad2002bff340c9213f2bba"), "rollno" : 128, "name" : "Himang", "city" : "bandara", "marks" : 89.99 }
{ "_id" : ObjectId("63ad201dbff340c9213f2bbb"), "rollno" : 129, "name" : "Faizan", "city" : "pune", "marks" : 91.54 }
{ "_id" : ObjectId("63ad2034bff340c9213f2bbc"), "rollno" : 130, "name" : "Niyamat", "city" : "pune", "marks" : 90.22 }
{ "_id" : ObjectId("63ad2051bff340c9213f2bbd"), "rollno" : 131, "name" : "prachi", "city" : "latur", "marks" : 92.93 }
{ "_id" : ObjectId("63ad2070bff340c9213f2bbe"), "rollno" : 133, "name" : "sami", "city" : "pune", "marks" : 71.33 }
{ "_id" : ObjectId("63ad209fbff340c9213f2bbf"), "rollno" : 134, "name" : "sandeep", "city" : "mumbai", "marks" : 81.41 }
{ "_id" : ObjectId("63ad21c9bff340c9213f2bc3"), "rollno" : 116, "name" : "vidya", "city" : "latur", "marks" : 85.33 }
>
```

2. \$or: It is used to join query clauses with a logical OR and return all documents that match the given conditions of either clause

```
> db.students.find({$or: [{marks: {$gt: 70.00}}, {marks: {$lt: 90.00}}]})
{ "_id" : ObjectId("63ad2002bff340c9213f2bba"), "rollno" : 128, "name" : "Himang", "city" : "bandara", "marks" : 89.99 }
{ "_id" : ObjectId("63ad201dbff340c9213f2bbb"), "rollno" : 129, "name" : "Faizan", "city" : "pune", "marks" : 91.54 }
{ "_id" : ObjectId("63ad2034bff340c9213f2bbc"), "rollno" : 130, "name" : "Niyamat", "city" : "pune", "marks" : 90.22 }
{ "_id" : ObjectId("63ad2051bff340c9213f2bbd"), "rollno" : 131, "name" : "prachi", "city" : "latur", "marks" : 92.93 }
{ "_id" : ObjectId("63ad2070bff340c9213f2bbe"), "rollno" : 133, "name" : "sami", "city" : "pune", "marks" : 71.33 }
{ "_id" : ObjectId("63ad209fbff340c9213f2bbf"), "rollno" : 134, "name" : "sandeep", "city" : "mumbai", "marks" : 81.41 }
{ "_id" : ObjectId("63ad20d2bff340c9213f2bc0"), "rollno" : 137, "name" : "shubham", "city" : "mumbai", "marks" : 39.8 }
{ "_id" : ObjectId("63ad20efbff340c9213f2bc1"), "rollno" : 139, "name" : "tanishka", "city" : "jalna", "marks" : 93.33 }
{ "_id" : ObjectId("63ad2104bff340c9213f2bc2"), "rollno" : 139, "name" : "vishal", "city" : "jalgaon", "marks" : 96.56,
"roll" : 140 }
{ "_id" : ObjectId("63ad21c9bff340c9213f2bc3"), "rollno" : 116, "name" : "vidya", "city" : "latur", "marks" : 85.33 }
{ "_id" : ObjectId("63ad2215bff340c9213f2bc4"), "rollno" : 110, "name" : "vaibhav", "city" : "buldana", "marks" : 70 }
>
```

3. \$not: It is used to invert the effect of the query expressions and return documents that do not match the query expression

```
> db.students.find({marks: {$not: {$gt: 70.00}}})
{ "_id" : ObjectId("63ad20d2bff340c9213f2bc0"), "rollno" : 137, "name" : "shubham", "city" : "mumbai", "marks" : 39.8 }
{ "_id" : ObjectId("63ad2215bff340c9213f2bc4"), "rollno" : 110, "name" : "vaibhav", "city" : "buldana", "marks" : 70 }
>
```

4. \$nor: It is used to join query clauses with a logical NOR and return all documents that fail to match both clauses

```
> db.students.find({$nor: [{marks: {$gt: 70.00}}, {city: "buldana"}]})
{ "_id" : ObjectId("63ad20d2bff340c9213f2bc0"), "rollno" : 137, "name" : "shubham", "city" : "mumbai", "marks" : 39.8 }
>
```

**Conclusion:** Hence, we successfully learn about the comparison and logical operators in MongoDB.

## Practical no 04

**Aim:** Practice exercise on element, array based and evaluation query operators - \$exists, \$type, \$mod, \$regex

### Theory:

#### Element Query Operators:

Element Operators return data based on field existence or data types

- a. Exists: Matches documents that have the specified field

```
> db.students.find({rollno: {$exists: true, $nin: [133, 139, 140]}})
{ "_id" : ObjectId("63ad2002bfff340c9213f2bba"), "rollno" : 128, "name" : "Himang", "city" : "bandara", "marks" : 89.99 }
{ "_id" : ObjectId("63ad201dbfff340c9213f2bbb"), "rollno" : 129, "name" : "Faizan", "city" : "pune", "marks" : 91.54 }
{ "_id" : ObjectId("63ad2034bfff340c9213f2bbc"), "rollno" : 130, "name" : "Niyamat", "city" : "pune", "marks" : 90.22 }
{ "_id" : ObjectId("63ad2051bfff340c9213f2bbd"), "rollno" : 131, "name" : "prachi", "city" : "latur", "marks" : 92.93 }
{ "_id" : ObjectId("63ad209fbfff340c9213f2bbf"), "rollno" : 134, "name" : "sandeep", "city" : "mumbai", "marks" : 81.41 }
{ "_id" : ObjectId("63ad20d2bfff340c9213f2bc0"), "rollno" : 137, "name" : "shubham", "city" : "mumbai", "marks" : 39.8 }
{ "_id" : ObjectId("63ad21c9bfff340c9213f2bc3"), "rollno" : 116, "name" : "vidya", "city" : "latur", "marks" : 85.33 }
{ "_id" : ObjectId("63ad2215bfff340c9213f2bc4"), "rollno" : 110, "name" : "vaibhav", "city" : "buldana", "marks" : 70 }
```

- b. Type: Selects documents if a field is of the specified type

```
> db.addressBook.find({"zipCode": {$type: "string"}})
{ "_id" : 1, "address" : "2030 Martian Way", "zipCode" : "90698345" }
{ "_id" : 5, "address" : "104 Venus Drive", "zipCode" : [ "834847278", "1893289032" ] }
> db.addressBook.find({"zipCode": {$type: 1}})
{ "_id" : 2, "address" : "156 Lunar Place", "zipCode" : 43339374 }
> db.addressBook.find({"zipCode": {$type: "number"}})
{ "_id" : 2, "address" : "156 Lunar Place", "zipCode" : 43339374 }
{ "_id" : 3, "address" : "2324 Pluto Place", "zipCode" : NumberLong(3921412) }
{ "_id" : 4, "address" : "55 Saturn Ring", "zipCode" : 88602117 }
>
```

#### Evaluation Query Operators:

Evaluation operators return data based on evaluations of either individuals' fields or the entire collection's documents.

- a. \$mod: Allows use of aggregation expressions within the query language.

```
> db.students.find({marks: {$mod: [4, 0]}})
{ "_id" : ObjectId("63ad2051bfff340c9213f2bbd"), "rollno" : 131, "name" : "prachi", "city" : "latur", "marks" : 92.93 }
{ "_id" : ObjectId("63ad2104bfff340c9213f2bc2"), "rollno" : 139, "name" : "vishal", "city" : "jalgaon", "marks" : 96.56, "roll" : 140 }
> db.students.find({marks: {$mod: [5, 0]}})
{ "_id" : ObjectId("63ad2034bfff340c9213f2bbc"), "rollno" : 130, "name" : "Niyamat", "city" : "pune", "marks" : 90.22 }
{ "_id" : ObjectId("63ad21c9bfff340c9213f2bc3"), "rollno" : 116, "name" : "vidya", "city" : "latur", "marks" : 85.33 }
{ "_id" : ObjectId("63ad2215bfff340c9213f2bc4"), "rollno" : 110, "name" : "vaibhav", "city" : "buldana", "marks" : 70 }
> db.students.find({marks: {$mod: [5, 2]}})
{ "_id" : ObjectId("63ad2051bfff340c9213f2bbd"), "rollno" : 131, "name" : "prachi", "city" : "latur", "marks" : 92.93 }
>
```

- b. \$regex: Selects documents where values match a regular expression

```
> db.students.find({name: {$not: {$regex: "s.*"}}})
{ "_id" : ObjectId("63ad2002bfff340c9213f2bba"), "rollno" : 128, "name" : "Himang", "city" : "bandara", "marks" : 89.99 }
{ "_id" : ObjectId("63ad201dbfff340c9213f2bbb"), "rollno" : 129, "name" : "Faizan", "city" : "pune", "marks" : 91.54 }
{ "_id" : ObjectId("63ad2034bfff340c9213f2bbc"), "rollno" : 130, "name" : "Niyamat", "city" : "pune", "marks" : 90.22 }
{ "_id" : ObjectId("63ad2051bfff340c9213f2bbd"), "rollno" : 131, "name" : "prachi", "city" : "latur", "marks" : 92.93 }
{ "_id" : ObjectId("63ad21c9bfff340c9213f2bc3"), "rollno" : 116, "name" : "vidya", "city" : "latur", "marks" : 85.33 }
{ "_id" : ObjectId("63ad2215bfff340c9213f2bc4"), "rollno" : 110, "name" : "vaibhav", "city" : "buldana", "marks" : 70 }
> db.students.find({name: {$regex: "s.*"}})
{ "_id" : ObjectId("63ad2070bfff340c9213f2bbe"), "rollno" : 133, "name" : "sami", "city" : "pune", "marks" : 71.33 }
{ "_id" : ObjectId("63ad209fbfff340c9213f2bbf"), "rollno" : 134, "name" : "sandeep", "city" : "mumbai", "marks" : 81.41 }
{ "_id" : ObjectId("63ad20d2bfff340c9213f2bc0"), "rollno" : 137, "name" : "shubham", "city" : "mumbai", "marks" : 39.8 }
{ "_id" : ObjectId("63ad20efbfff340c9213f2bc1"), "rollno" : 139, "name" : "tanishka", "city" : "jalna", "marks" : 93.33 }
{ "_id" : ObjectId("63ad2104bfff340c9213f2bc2"), "rollno" : 139, "name" : "vishal", "city" : "jalgaon", "marks" : 96.56,
```

**Conclusion:** Hence, we Successfully learn about the element based, array based and evaluation based query operators and also performed the operators like \$exists, \$type, \$mod, and \$regex.

## Experiment NO 05

**Aim:** Exercise on MongoDB shell commands and user management

### Theory:

#### MongoDB Shell Commands

MongoDB Shell is the quickest way to connect, configure, query, and work with your MongoDB database. It acts as a command-line client of the MongoDB server.

You can start MongoDB Shell by executing `mongo` or `mongo` command on the command prompt/terminal. `mongo` is the new MongoDB shell with some more features than the old `mongo` shell.

`mongo <commands>`

1. `--help`: The `--help` command display all the commands which you can use with `mongo` or `mongosh`.

```
c:\mongodb>mongo --help
MongoDB shell version v4.4.18
usage: mongo [options] [db address] [file names (ending in .js)]
db address can be:
  foo                foo database on local machine
  192.168.0.5/foo     foo database on 192.168.0.5 machine
  192.168.0.5:9999/foo foo database on 192.168.0.5 machine on port 9999
  mongodb://192.168.0.5:9999/foo connection string URI can also be used
Options:
  --ipv6                enable IPv6 support (disabled by
                        default)
  --host arg            server to connect to
  --port arg            port to connect to
  -h [ --help ]        show this usage information
  --version             show version information
  --verbose             increase verbosity
  --shell              run the shell after executing files
  --nodb               don't connect to mongod on startup - no
                        'db address' arg expected
```

2. `--nodb`: allows you to run MongoDB shell without connecting to a database.

```
c:\mongodb>mongo --nodb
MongoDB shell version v4.4.18
> |
```

3. `--port`: To connect with the local database on a different port.

```
C:\>mongosh --port 23023
```

4. `--version`: Show version information

```
c:\mongodb>mongo --version
MongoDB shell version v4.4.18
Build Info: {
  "version": "4.4.18",
  "gitVersion": "8ed32b5c2c68ebe7f8ae2ebe8d23f36037a17dea",
  "modules": [],
  "allocator": "tcmalloc",
  "environment": {
    "distmod": "windows",
    "distarch": "x86_64",
    "target_arch": "x86_64"
  }
}
```

5. `--verbose`: Increase the verbosity of the output of the shell

```
c:\mongodb>mongo --verbose
{"t":{"$date":"2023-01-03T14:33:55.566Z"},"s":"D1", "c":"CONTROL", "id":22615, "ctx":"main","msg":"Loading library: {
toUtf8String_full_path_c_str","attr":{"toUtf8String_full_path_c_str":"Schannel.dll"}}
{"t":{"$date":"2023-01-03T14:33:55.573Z"},"s":"I", "c":"NETWORK", "id":4648602, "ctx":"main","msg":"Implicit TCP FastO
pen in use."}
MongoDB shell version v4.4.18
{"t":{"$date":"2023-01-03T14:33:55.574Z"},"s":"D1", "c":"EXECUTOR", "id":23104, "ctx":"OCSPManagerHTTP-0","msg":"Start
ing thread","attr":{"threadName":"OCSPManagerHTTP-0","poolName":"OCSPManagerHTTP"}}
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
{"t":{"$date":"2023-01-03T14:33:55.611Z"},"s":"D1", "c":"NETWORK", "id":20109, "ctx":"js","msg":"Creating new connect
ion", "attr":{"hostAndPort":"127.0.0.1:27017"}}
{"t":{"$date":"2023-01-03T14:33:55.613Z"},"s":"D1", "c":"NETWORK", "id":20119, "ctx":"js","msg":"Connected to host", "
attr":{"connString":"127.0.0.1:27017"}}
{"t":{"$date":"2023-01-03T14:33:55.626Z"},"s":"D1", "c":"NETWORK", "id":20110, "ctx":"js","msg":"Connected connection
!"}
Implicit session: session { "id" : UUID("1ab261a2-f07e-4f47-a8b9-ec13a5137024") }
MongoDB server version: 4.4.18
---
```

6. `--norc`: Will not run the `‘.mongoshrc.js’` files on startup

```
c:\mongodb>mongo --norc
MongoDB shell version v4.4.18
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("0db900da-8621-4b86-94a6-62ae18c2140e") }
MongoDB server version: 4.4.18
---
The server generated these startup warnings when booting:
  2023-01-03T18:14:13.974+05:30: Access control is not enabled for the database. Read and write access to data and
configuration is unrestricted
  2023-01-03T18:14:13.975+05:30: This server is bound to localhost. Remote systems will be unable to connect to th
is server. Start the server with --bind_ip <address> to specify which IP addresses it should serve responses from, or wi
th --bind_ip_all to bind to all interfaces. If this behavior is desired, start the server with --bind_ip 127.0.0.1 to di
sable this warning
---
```

7. `--retryWrites`: Automatically retry write operations upon transient network errors

```
c:\mongodb>mongo --retryWrites
MongoDB shell version v4.4.18
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("a4284d7d-433c-43f2-a0bf-947c30154fea") }
MongoDB server version: 4.4.18
---
The server generated these startup warnings when booting:
  2023-01-03T18:14:13.974+05:30: Access control is not enabled for the database. Read and write access to data and
configuration is unrestricted
  2023-01-03T18:14:13.975+05:30: This server is bound to localhost. Remote systems will be unable to connect to th
is server. Start the server with --bind_ip <address> to specify which IP addresses it should serve responses from, or wi
th --bind_ip_all to bind to all interfaces. If this behavior is desired, start the server with --bind_ip 127.0.0.1 to di
sable this warning
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
```

## MongoDB User Management Commands:

1. `db.auth()`: Authenticates a user to a database.

```
> use studentdb
switched to db studentdb
> db.auth("shubhamsapkal", "shubham@123")
1
```

2. db.changeUserPassword(): Changes an existing user's password.

```
> db.changeUserPassword("shubhamsapkal", "shubham@123")
```

3. db.createUser(): Creates a new user.

```
> db.createUser( { user: "shubhamsapkal",
...               pwd: "shubham123",
...               roles: [ { role: "clusterAdmin", db: "admin" },
...                       { role: "readAnyDatabase", db: "admin" },
...                       "readWrite" ]
... })
Successfully added user: {
  "user" : "shubhamsapkal",
  "roles" : [
    {
      "role" : "clusterAdmin",
      "db" : "admin"
    },
    {
      "role" : "readAnyDatabase",
      "db" : "admin"
    },
    "readWrite"
  ]
}
```

4. db.dropUser(): Removes a single user.

```
> db.dropUser("shubhamsapkal")
true
```

5. db.getUsers(): Returns information about all users associated with a database.

```
> db.getUsers()
[
  {
    "_id" : "studentdb.shubhamsapkal",
    "userId" : UUID("ec8b4589-365a-40d5-a1a8-49407c001385"),
    "user" : "shubhamsapkal",
    "db" : "studentdb",
    "roles" : [
      {
        "role" : "clusterAdmin",
        "db" : "admin"
      },
      {
        "role" : "readAnyDatabase",
        "db" : "admin"
      },
      {
        "role" : "readWrite",
        "db" : "studentdb"
      }
    ],
    "mechanisms" : [
      "SCRAM-SHA-1",
      "SCRAM-SHA-256"
    ]
  }
]
```

**Conclusion:** Hence, we successfully learn about the MongoDB Shell and User management Commands.

## Experiment NO 06

**Aim:** Installation and configuration of Cassandra. Find out two use cases where Cassandra is preferred over MongoDB

### Theory:

#### Installation of Apache Cassandra on Ubuntu

**Step 1:** As the java environment is not available on the system, we will install openjdk-8-jdk using the below command.

```
sudo apt install openjdk-8-jdk
```

```
shuuubham@Shubham-Laptop:~$ sudo apt install openjdk-8-jdk
[sudo] password for shuuubham:
Reading package lists... Done
Building dependency tree
Reading state information... Done
openjdk-8-jdk is already the newest version (8u342-b07-0ubuntu1~20.04).
The following packages were automatically installed and are no longer required:
  default-jdk-headless libfwupdplugin1 openjdk-11-jdk openjdk-11-jdk-headless
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 257 not upgraded.
```

**Step 2:** In this step, we will check whether the java environment is set up or not.

```
java --version
```

```
shuuubham@Shubham-Laptop:~$ java -version
openjdk version "1.8.0_342"
OpenJDK Runtime Environment (build 1.8.0_342-8u342-b07-0ubuntu1~20.04-b07)
OpenJDK 64-Bit Server VM (build 25.342-b07, mixed mode)
```

**Step 3:** Install the apt-transport-https package to permit access of repositories via the HTTPS protocol.

```
sudo apt install apt-transport-https
```

```
shuuubham@Shubham-Laptop:~$ sudo apt install apt-transport-https
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  default-jdk-headless libfwupdplugin1 openjdk-11-jdk openjdk-11-jdk-headless
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  apt-transport-https
0 upgraded, 1 newly installed, 0 to remove and 257 not upgraded.
Need to get 1704 B of archives.
After this operation, 162 kB of additional disk space will be used.
0% [Working]
```

**Step 4:** Import the GPG key using the following [wget](#) command shown in the below screenshot.

```
wget -q -O - https://www.apache.org/dist/cassandra/KEYS | sudo apt-key add -
```

```
shuuubham@Shubham-Laptop:~$ curl https://downloads.apache.org/cassandra/KEYS | sudo apt-key add -
% Total    % Received % Xferd Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left     Speed
100 267k  100 267k    0     0  126k      0  0:00:02  0:00:02 --:--:-- 126k
OK
shuuubham@Shubham-Laptop:~$
```

**Step 5:** Now, we will add Apache Cassandra's repository to the system's sources list file by using the below command.

```
echo "deb http://downloads.apache.org/cassandra/debian 40x main" | sudo tee -a
/etc/apt/sources.list.d/cassandra.sources.list
```

```
shuuubham@Shubham-Laptop:~$ echo "deb http://downloads.apache.org/cassandra/debian 40x main" | sudo tee -a /etc/apt/sources.list.d/cassandra.sources.list
deb http://downloads.apache.org/cassandra/debian 40x main
```

**Step 6:** Update the repositories by using the below command.

```
sudo apt-get update
```

```
shuuubham@Shubham-Laptop:~$ sudo apt-get update
Hit:2 http://archive.ubuntu.com/ubuntu focal InRelease
Hit:3 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:4 http://archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:5 http://archive.ubuntu.com/ubuntu focal-backports InRelease
0% [Working]
```

**Step 7:** Now, install the Cassandra via the [apt manager](#). Run the below command to install the application.

```
sudo apt-get install cassandra
```



```
shuuubham@Shubham-Laptop:~$ sudo apt-get install cassandra
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  default-jdk-headless libfwupdplugin1 openjdk-11-jdk openjdk-11-jdk-headless
Use 'sudo apt autoremove' to remove them.
Suggested packages:
  cassandra-tools
The following NEW packages will be installed:
  cassandra
0 upgraded, 1 newly installed, 0 to remove and 281 not upgraded.
Need to get 47.5 MB of archives.
After this operation, 58.8 MB of additional disk space will be used.
0% [Working]
```

**Step 8:** Now start the service of cassandra.

sudo service cassandra start

```
shuuubham@Shubham-Laptop:~$ sudo service cassandra start
shuuubham@Shubham-Laptop:~$ |
```

**Step 9:** Now, verify the stats of your node by running the command.

sudo service cassandra status

```
shuuubham@Shubham-Laptop:~$ sudo service cassandra status
* Cassandra is running
shuuubham@Shubham-Laptop:~$ |
```

**Step 10:** Run following command to start Cassandra database

Sudo cqlsh

```
shuuubham@Shubham-Laptop:~$ sudo cqlsh
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.0.0 | Cassandra 4.0.5 | CQL spec 3.4.5 | Native protocol v5]
Use HELP for help.
cqlsh> |
```

**Conclusion:** Hence, we successfully learn about what is Cassandra database, how to install and configure cassandra database and 2 use cases where Cassandra preferred than mongodb.

## Experiment NO 07

**Aim:** Create database in Casandra using – Create, Alter and Drop. Add records using Insert, Update, Delete and Truncate.

### Theory:

#### Creating keyspace in Cassandra:

Database is also referred as keyspace in cassendra.

A keyspace in Cassandra is a namespace that defines data replication on nodes. A cluster contains one keyspace per node. Given below is the syntax for creating a keyspace using the statement CREATE KEYSPACE.

Syntax

```
CREATE KEYSPACE <identifier> WITH <properties>
```

i.e.

```
CREATE KEYSPACE "testdb2" WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 3};
```

```
cqlsh> CREATE KEYSPACE testdb2 WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 3};  
Warnings :  
Your replication factor 3 for keyspace testdb2 is higher than the number of nodes 1
```

### Altering Keyspace in Cassandra:

ALTER KEYSPACE can be used to alter properties such as the number of replicas and the durable\_writes of a KeySpace. Given below is the syntax of this command.

Syntax

ALTER KEYSPACE <identifier> WITH <properties>

i.e.

ALTER KEYSPACE testdb2 WITH replication = {'class':'NetworkTopologyStrategy', 'replication\_factor' : 1};

```
cqlsh> ALTER KEYSPACE testdb2 WITH replication = {'class':'NetworkTopologyStrategy', 'replication_factor' : 1};  
cqlsh> |
```

### Dropping the keyspace in Cassandra:

You can drop a KeySpace using the command DROP KEYSPACE. Given below is the syntax for dropping a KeySpace.

Syntax

DROP KEYSPACE <identifier>

i.e.

DROP KEYSPACE testdb2

```
cqlsh> drop keyspace testdb2  
... ;  
cqlsh>
```

### Creating a table:

You can create a table using the command CREATE TABLE. Given below is the syntax for creating a table.

Syntax

CREATE (TABLE | COLUMNFAMILY) <tablename>

('<column-definition>', '<column-definition>')

(WITH <option> AND <option>)

```
cqlsh:testdb2> CREATE TABLE students(sid int PRIMARY KEY, sname text, dept text, city text, marks varint );  
cqlsh:testdb2> |
```

### Inserting a record in Cassandra:

You can insert data into the columns of a row in a table using the command INSERT. Given below is the syntax for creating data in a table.

```
INSERT INTO <tablename> (<column1 name>, <column2 name>....) VALUES (<value1>, <value2>....) USING  
<option>
```

```
cqlsh:testdb2> INSERT INTO students(sid, sname, dept, city, marks) VALUES (137, 'shubham', 'comp', 'mumbai',  
92);  
cqlsh:testdb2> |
```

### Updating a record in Cassandra:

UPDATE is the command used to update data in a table. The following keywords are used while updating data in a table –

- Where – This clause is used to select the row to be updated.
- Set – Set the value using this keyword.
- Must – Includes all the columns composing the primary key.

While updating rows, if a given row is unavailable, then UPDATE creates a fresh row. Given below is the syntax of UPDATE command –

```
UPDATE <tablename> SET <column name> = <new value> <column name> <value>.... WHERE  
<condition>
```

```
cqlsh:testdb2> UPDATE students SET marks=93 WHERE sid=137;  
cqlsh:testdb2> |
```

### Deleting data from a table in Cassandra:

You can delete data from a table using the command DELETE. Its syntax is as follows –

```
DELETE FROM <identifier> WHERE <condition>;
```

```
cqlsh:testdb2> DELETE FROM students WHERE sid=137;  
cqlsh:testdb2> |
```

### TRUNCATE Table in Cassandra:

Removes all data from the specified table immediately and irreversibly, and removes all data from any materialized views derived from that table.

Syntax:

```
TRUNCATE [TABLE] [keyspace_name.table_name]
```

```
cqlsh:testdb2> TRUNCATE TABLE students;  
cqlsh:testdb2> |
```

**Conclusion:** Hence, we successfully learn to create database in cassandra and inserting, updating, deleting, and truncating data in cassandra database.

## Experiment NO 08

**Aim:** Exercise based on Cassandra Query Language i.e. selecting records, select records with specific conditions

### Theory:

#### Selecting Records from table in Cassandra

SELECT clause is used to read data from a table in Cassandra. Using this clause, you can read a whole table, a single column, or a particular cell. Given below is the syntax of SELECT clause.

```
SELECT FROM <tablename>
```

```
cqlsh:testdb2> SELECT * FROM students;
```

sid	city	dept	marks	sname
128	Bandara	ENTC	91	Himang
140	jalgaon	ENTC	94	Vishal
129	pune	IT	89	faizan
137	mumbai	comp	92	shubham
134	Mumbai	IT	90	Sandeep
139	buldana	Mech	98	Vaibhav
130	pune	comp	88	Niyamat

### Selecting Records from table with specific condition in cassandra:

Using WHERE clause, we can able to specify condition on which data is going to retrieve

```
SELECT FROM <table name> WHERE <condition>;
```

Note – A WHERE clause can be used only on the columns that are a part of primary key or have a secondary index on them.

```
cqlsh:testdb2> select * from students where sid = 137;
```

sid	city	dept	marks	sname
137	mumbai	comp	92	shubham

```
(1 rows)
```

**Conclusion:** Hence, we successfully able learn how to select records from Cassandra table and also selected records from table with specific condition.