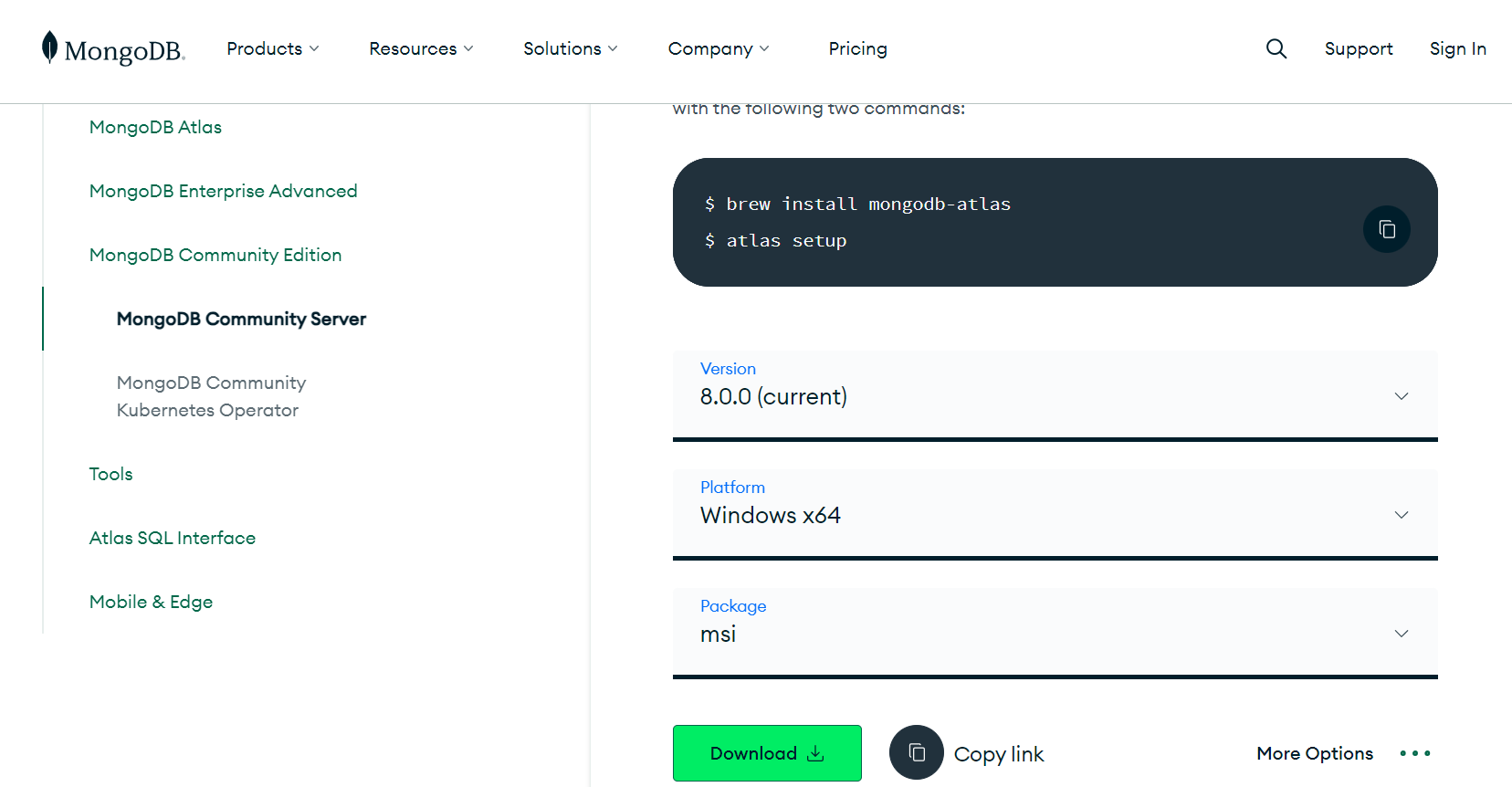
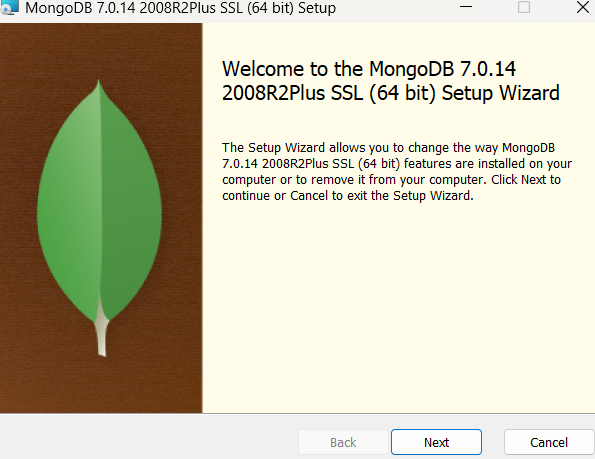
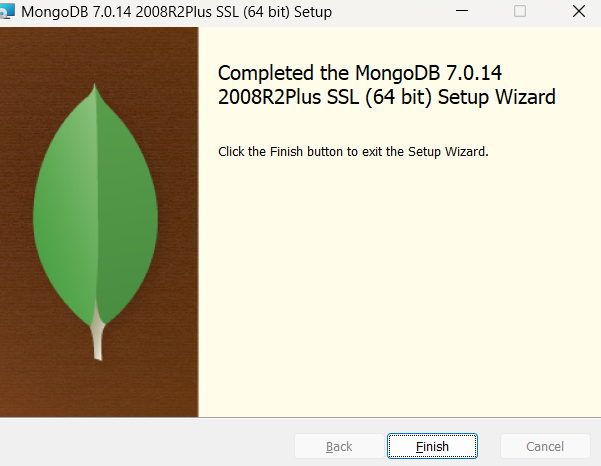
| **Practical-1** | **Lab Exercise: Setting up and Exploring MongoDB** |
| --- | --- |
| **a** | ***Install MongoDB on your local machine or lab server.*** |
| **b** | ***Create a new MongoDB database and collection.*** |
| **c** | ***Insert sample data into collection*** |
| **d** | ***Retrieve and display data from the collection using MongoDB queries.*** |

1. ***Install MongoDB on your local machine or lab server.***

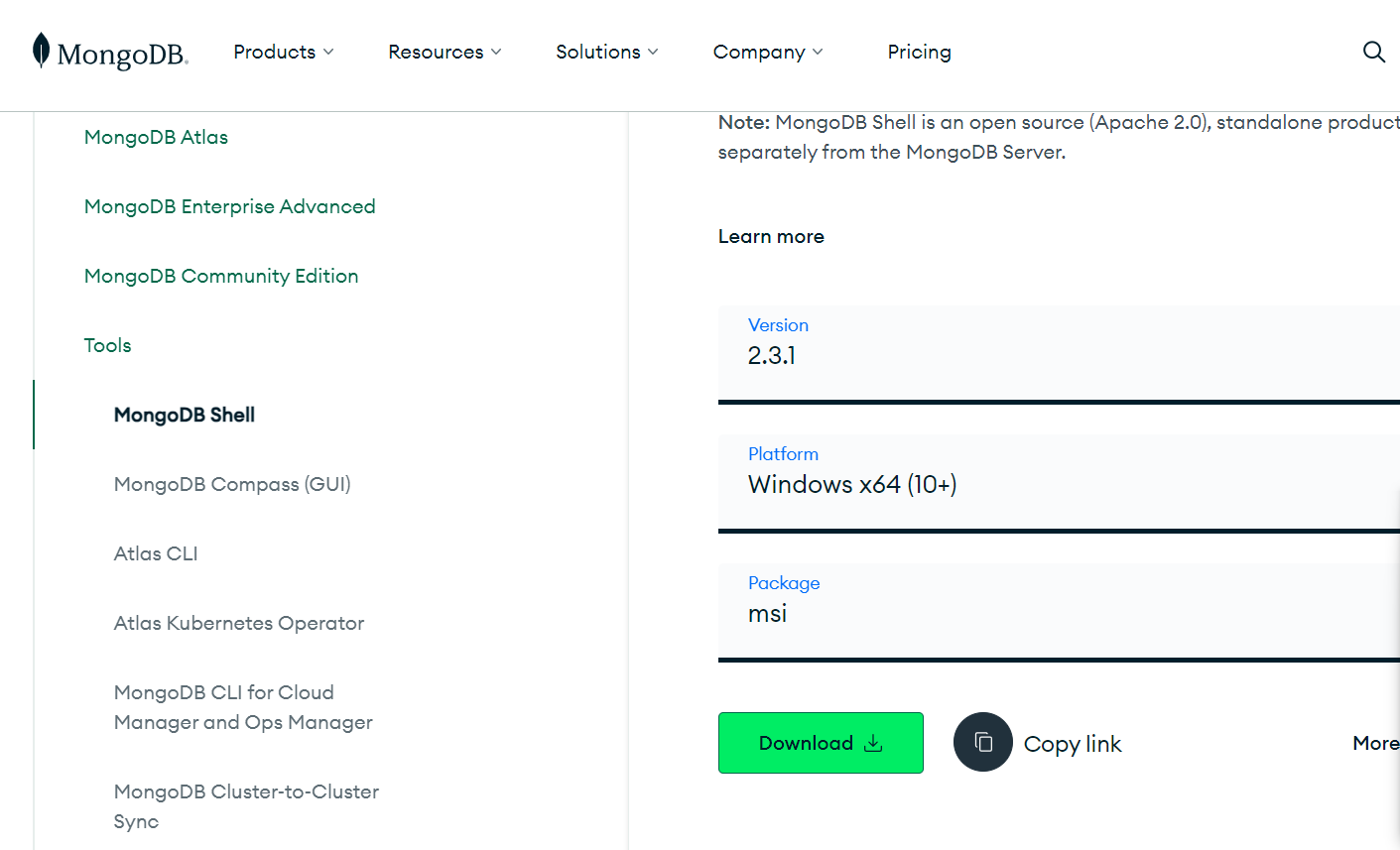
***Steps:*** 1. DOWNLOAD MONGODB COMMUNITY SERVER

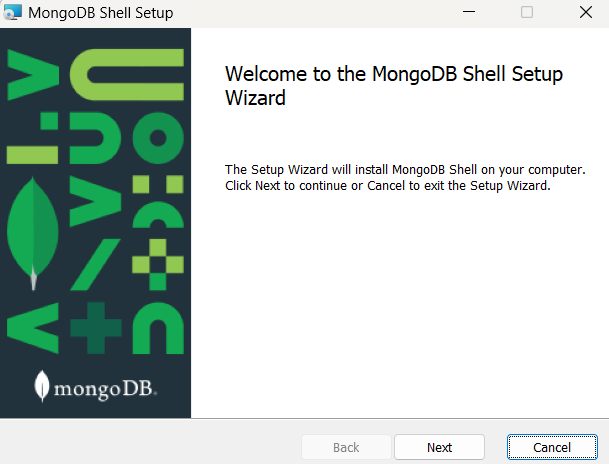


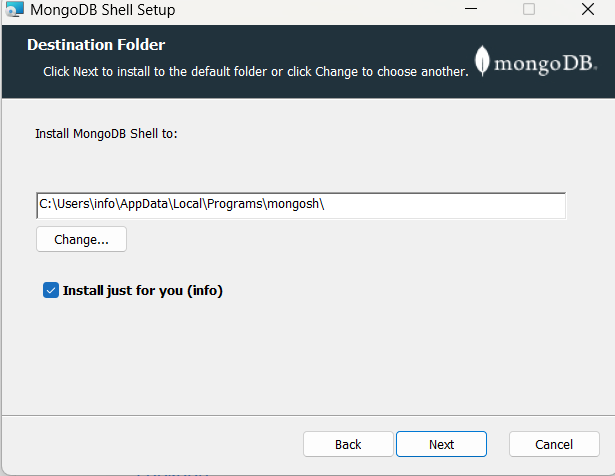


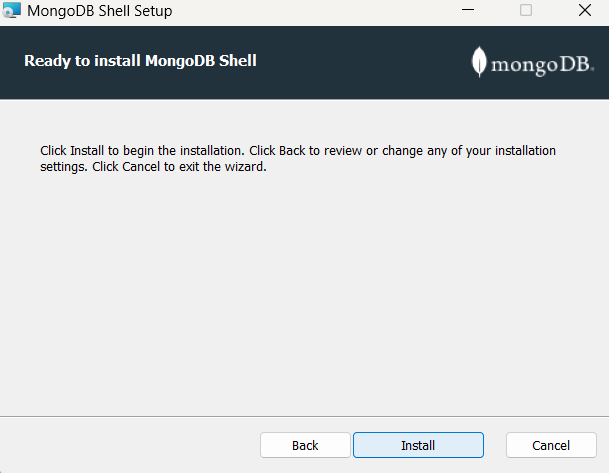


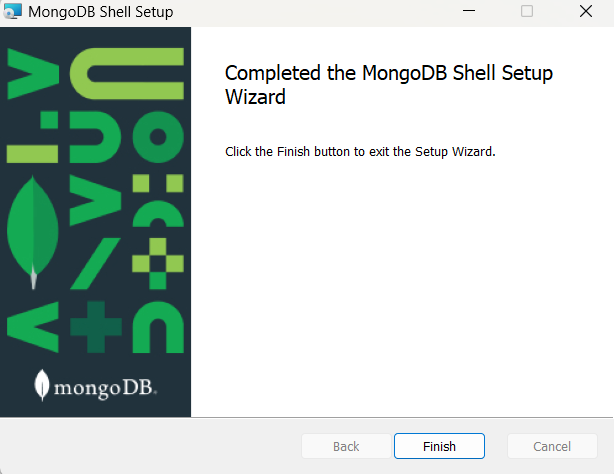
1. Download MongoDB Shell



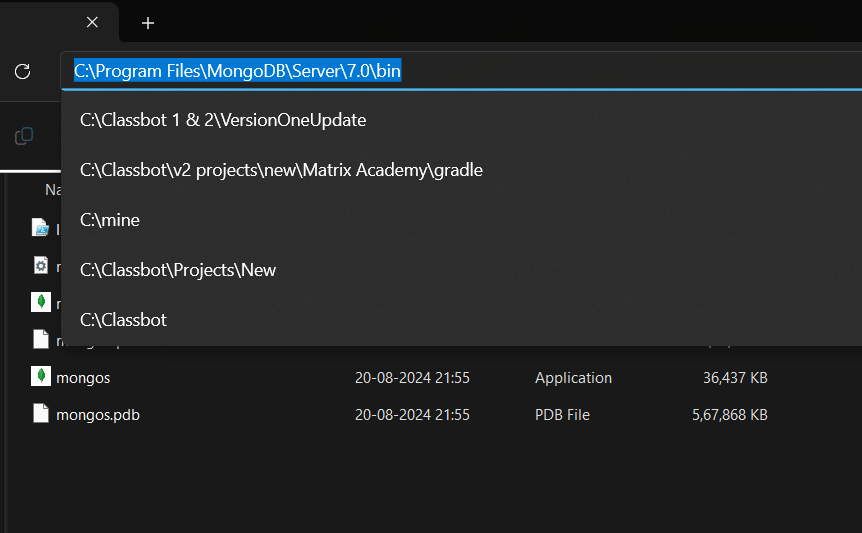




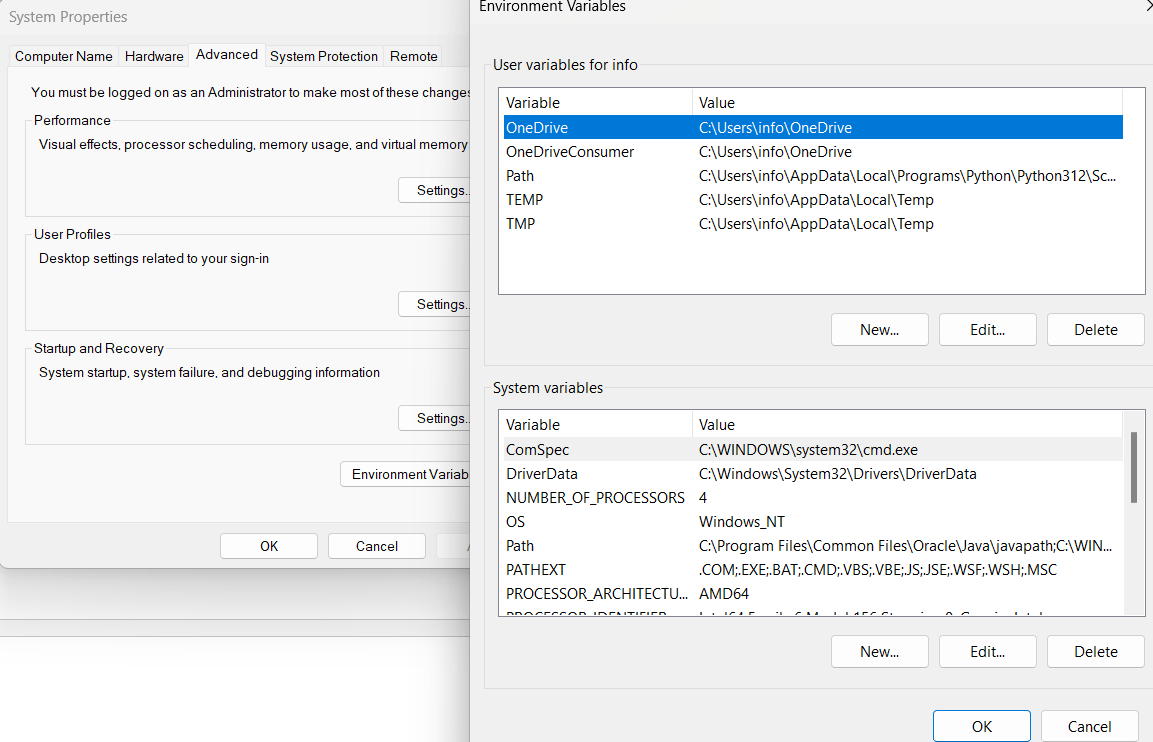




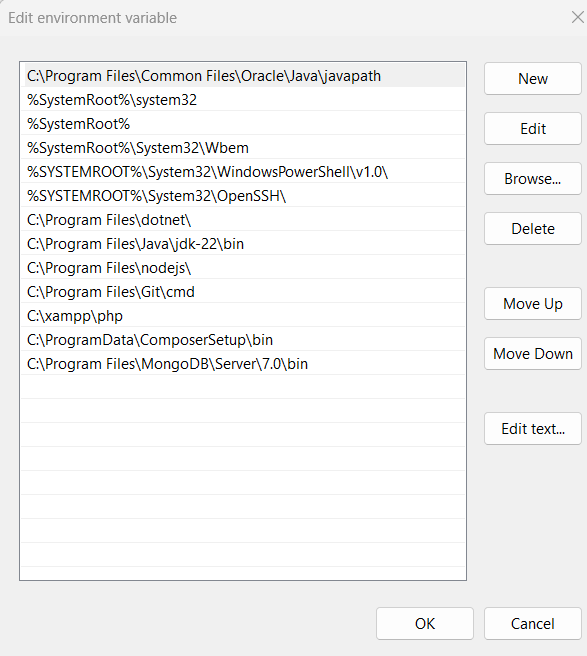
Copy bin path C:\Program Files\MongoDB\Server\7.0\bin



Search environment variables

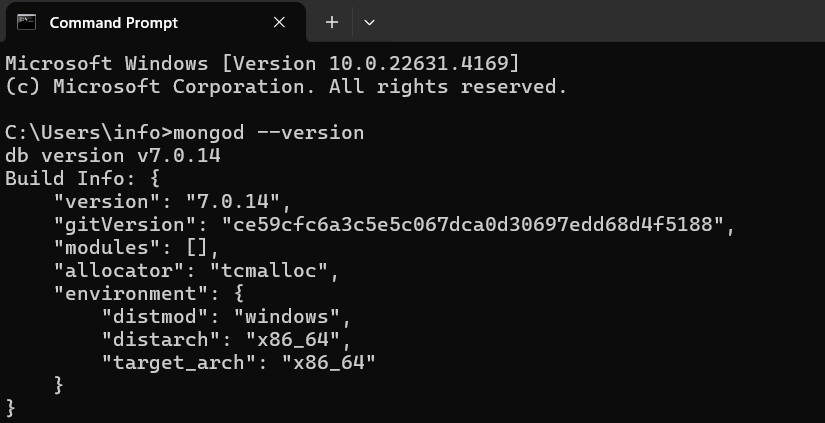


Double click on path in system variables and paste the copied address

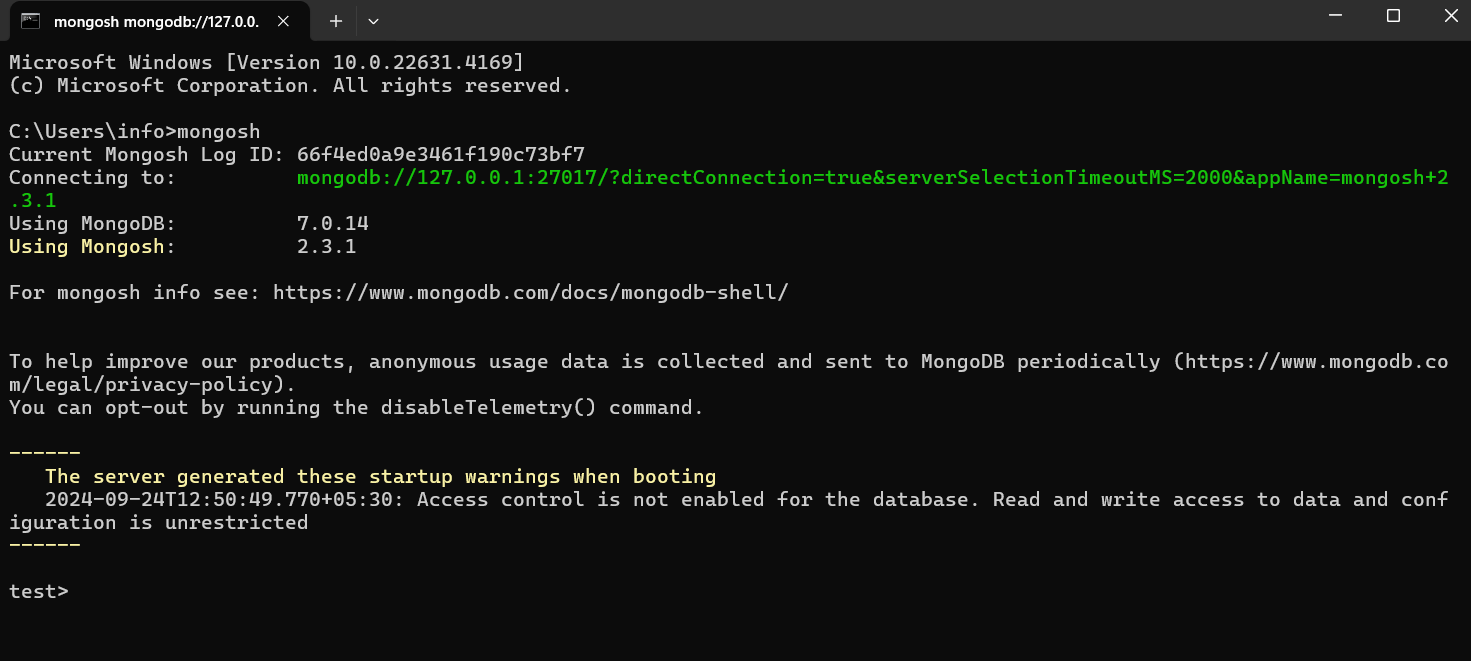


Click on ok to save the details

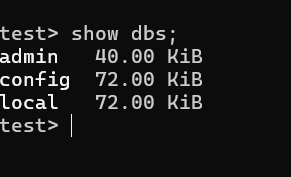
Now open command prompt and type mongod --version to check if it is properly installed.



Now open new command Prompt and check if shell is installed properly. Type mongosh to check.



Now type show dbs to check current databases and check if things are working properly.



Now the initial setups to use the mongo db client and server is done. We can start working on the practicals from now onwards.

1. ***Create a new MongoDB database and collection.***

Steps:

1. start server and shell:

Server (Mongod.exe)

1. Before starting the server make sure following things:
   1. Path of bin directory (located in C:\Program Files\MongoDB\Server\7.0\bin)
   2. Path of db directory (you must explicitly create this directory in C:\data\db)
2. Open command prompt
3. Change directory : C:>**cd C:\Program Files\MongoDB\Server\7.0\bin**
4. Start server: C:\Program Files\….....\bin\>**mongod.exe --dbpath C:\data\db**

Shell (Mongo.exe)

1. Open new command prompt
2. Change directory : C:>**cd C:\Program Files\MongoDB\Server\7.0\bin**
3. Start shell: C:\Program Files\ **MongoDB\Server\7.0**\bin\>**mongosh.exe**

Consider Example:

1. Database Name: MscCS
2. Collection Name: Books

**Execute following commands on shell.**

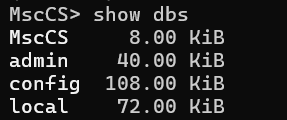
* 1. **To create Database**

use MscCS



**2. To show Database**

show dbs



**3. To create collection**

db.createCollection("books")

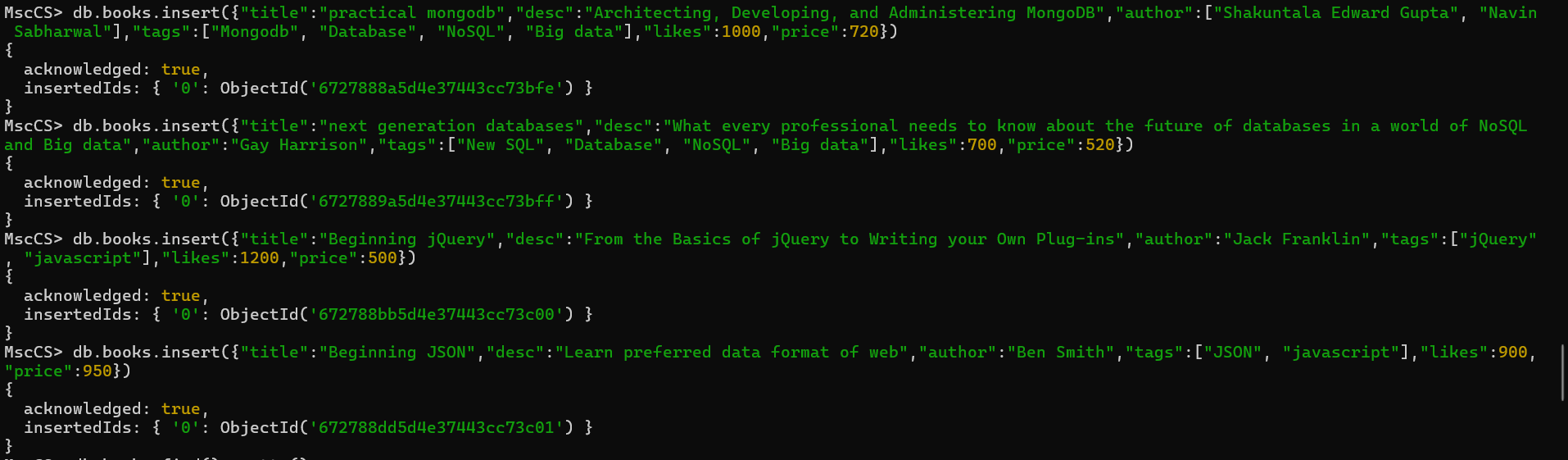


**4. To show collections**

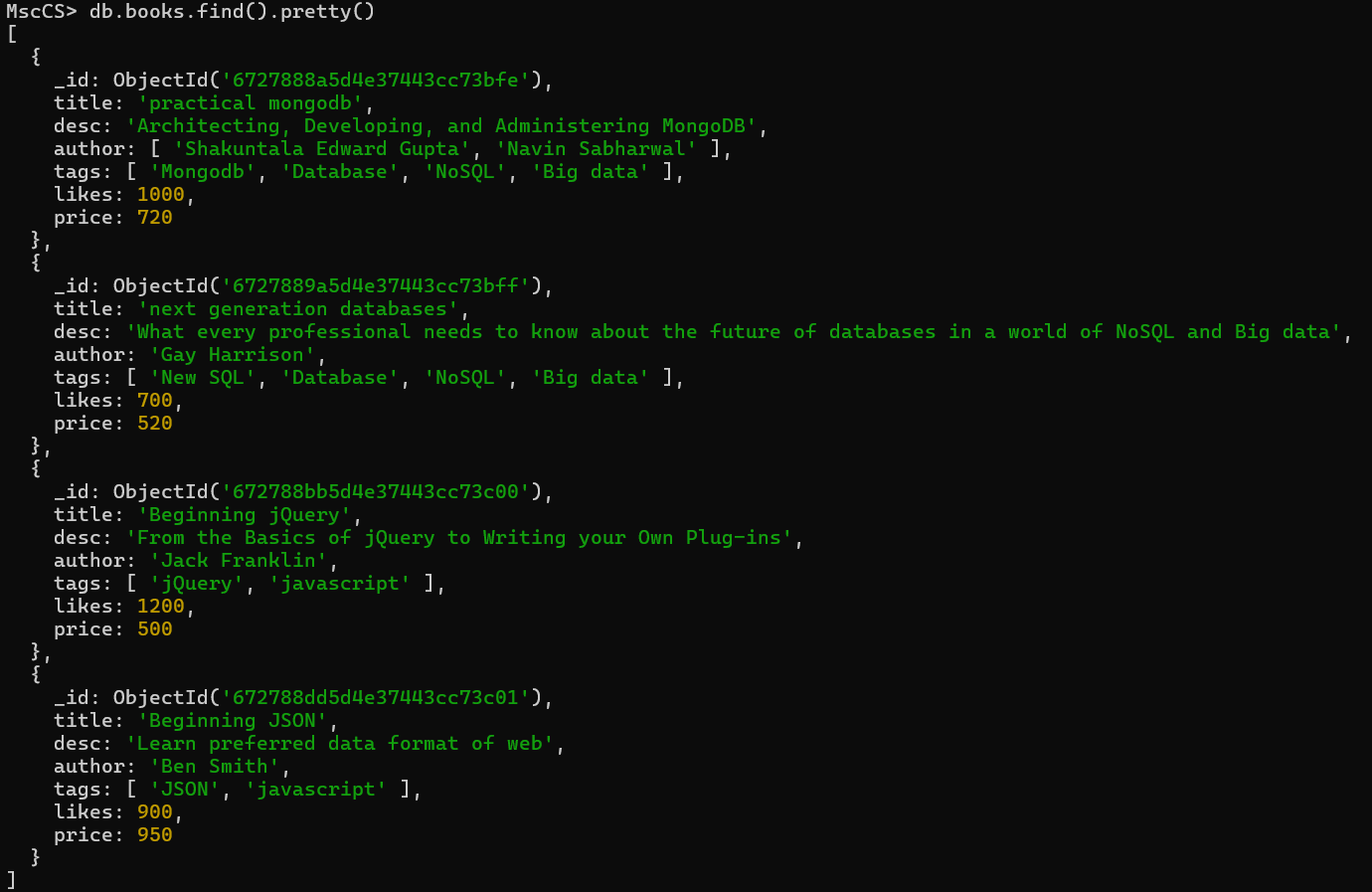
show collections



1. ***Insert sample data into the collection.***
2. db.books.insert({"title":"practical mongodb","desc":"Architecting, Developing, and Administering MongoDB","author":["Shakuntala Edward Gupta", "Navin Sabharwal"],"tags":["Mongodb", "Database", "NoSQL", "Big data"],"likes":1000,"price":720})
3. db.books.insert({"title":"next generation databases","desc":"What every professional needs to know about the future of databases in a world of NoSQL and Big data","author":"Gay Harrison","tags":["New SQL", "Database", "NoSQL", "Big data"],"likes":700,"price":520})
4. db.books.insert({"title":"Beginning jQuery","desc":"From the Basics of jQuery to Writing your Own Plug-ins","author":"Jack Franklin","tags":["jQuery", "javascript"],"likes":1200,"price":500})
5. db.books.insert({"title":"Beginning JSON","desc":"Learn preferred data format of web","author":"Ben Smith","tags":["JSON", "javascript"],"likes":900,"price":950})

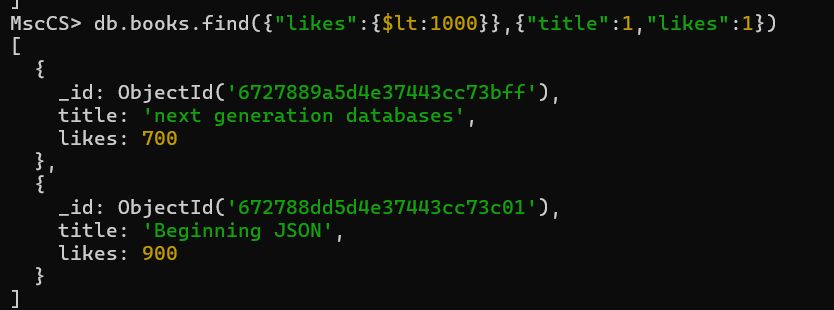


1. ***Retrieve and display data from the collection using MongoDB queries***

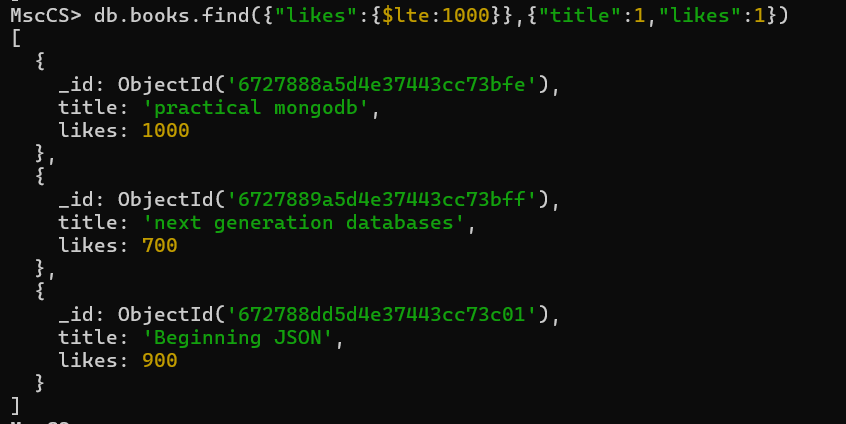
***Retrieve Data***: db.books.find().pretty()

***display data from the collection using MongoDB queries***

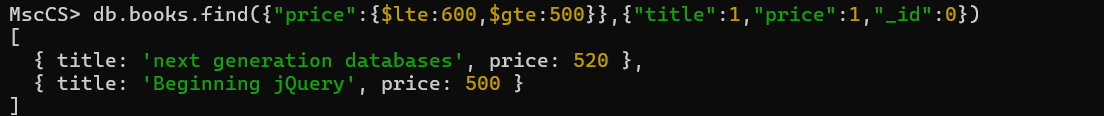
db.books.find({"likes":{$lt:1000}},{"title":1,"likes":1})



db.books.find({"likes":{$lte:1000}},{"title":1,"likes":1})

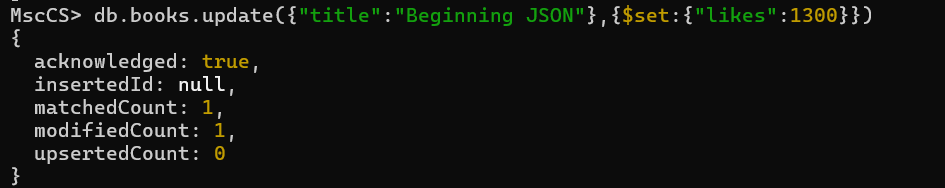


db.books.find({"price":{$lte:600,$gte:500}},{"title":1,"price":1,"\_id":0})



**To update data from collection**

db.books.update({"title":"Beginning JSON"},{$set:{"likes":1300}})



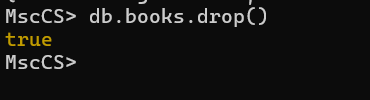
**To delete data from collection**

db.book.remove({"title":"learning mongodb"})



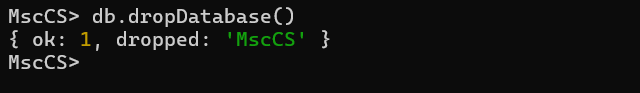
**To drop collection**

db.books.drop()



**To drop Database**

db.dropDatabase()



| **Practical No. 2** | **Interacting with Redis** |
| --- | --- |
| **a)** | ***Install Redis on your lab server or local machine.*** |
| **b)** | ***Store and retrieve data in Redis using various data structures like strings,***  ***lists, and sets.*** |
| **c)** | ***Implement basic Redis commands for data manipulation and retrieval*** |

Setup :

Install Redis Using Windows Subsystem for Linux (WSL)

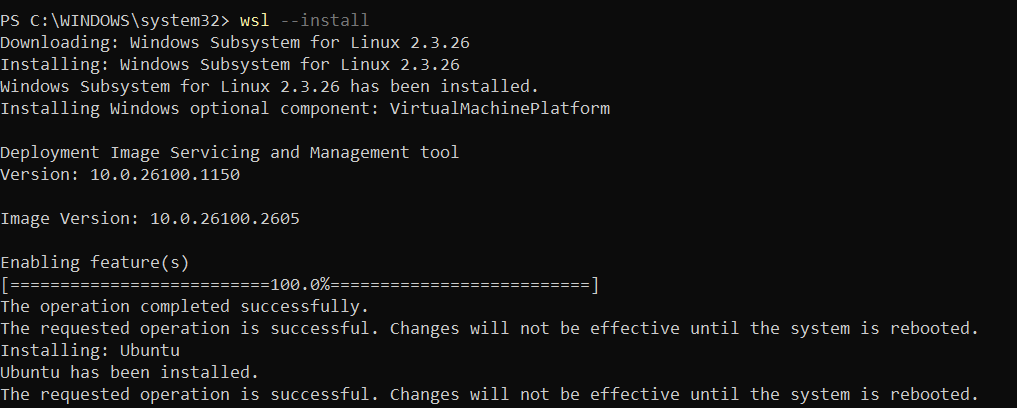
This method runs Redis on a Linux environment inside Windows.

Enable WSL:

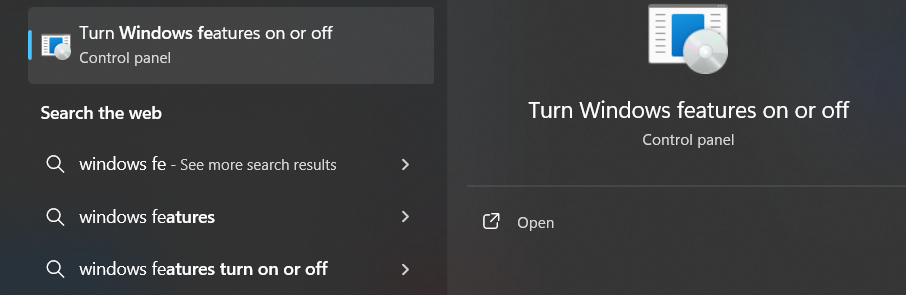
Open PowerShell as Administrator and run:

wsl --install

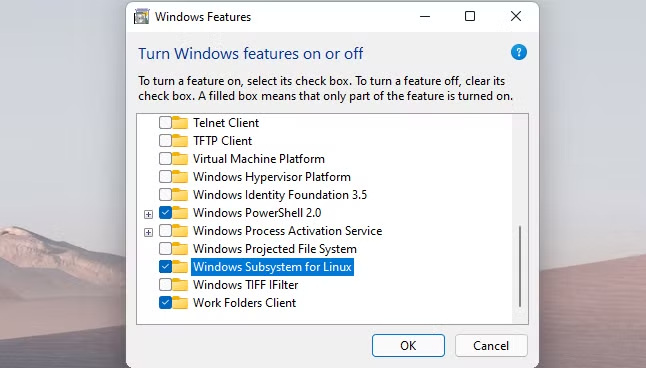
This installs WSL with a default Linux distribution (like Ubuntu).



After installing wsl , go to windows search windows features ,



And then select windows subsystem for linux and save



> sudo apt update sudo apt install redis-server

Start Redis:

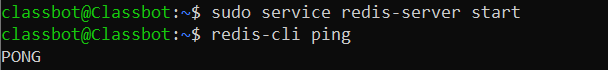
Run the Redis server:

> sudo service redis-server start

Verify it's running:

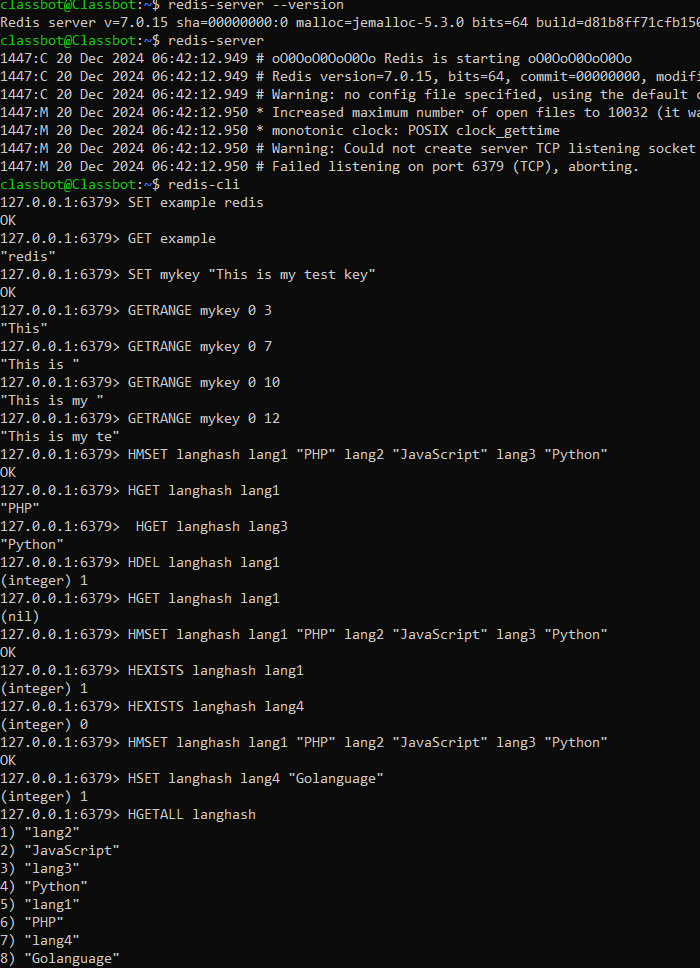
> redis-cli ping

You should see PONG.



Now start the ubuntu kernel and start the server and client and perform

operations.



**String Data Type in Redis**

127.0.0.1:6379> SET example redis

OK

127.0.0.1:6379> GET example

"redis"

127.0.0.1:6379> SET mykey "This is my test key"

OK

127.0.0.1:6379> GETRANGE mykey 0 3

"This"

## Hash Data type in Redis

**Redis HMSET & HGET**

127.0.0.1:6379> HMSET langhash lang1 "PHP" lang2 "JavaScript" lang3 "Python"

OK

127.0.0.1:6379> HGET langhash lang1

"PHP"

127.0.0.1:6379> HGET langhash lang2

"JavaScript"

127.0.0.1:6379> HGET langhash lang3

"Python"

127.0.0.1:6379> HDEL langhash lang1

(integer) 1

127.0.0.1:6379> HGET langhash lang1

(nil)

127.0.0.1:6379> HGET langhash lang2

"JavaScript"

127.0.0.1:6379> HGET langhash lang3

"Python"

**Redis HDEL**

127.0.0.1:6379> HDEL langhash lang1 lang2 lang3

(integer) 3

127.0.0.1:6379> HGET langhash lang1

(nil)

127.0.0.1:6379> HGET langhash lang2

(nil)

127.0.0.1:6379> HGET langhash lang3

(nil)

**Redis HEXISTS**

127.0.0.1:6379> HMSET langhash lang1 "PHP" lang2 "JavaScript" lang3 "Python"

OK

127.0.0.1:6379> HEXISTS langhash lang1

(integer) 1

127.0.0.1:6379> HEXISTS langhash lang4

(integer) 0

**Redis HGETALL**

127.0.0.1:6379> HMSET langhash lang1 "PHP" lang2 "JavaScript" lang3 "Python"

OK

127.0.0.1:6379> HSET langhash lang4 "Golanguage"

(integer) 1

127.0.0.1:6379> HGETALL langhash

1) "lang1"

2) "PHP"

3) "lang2"

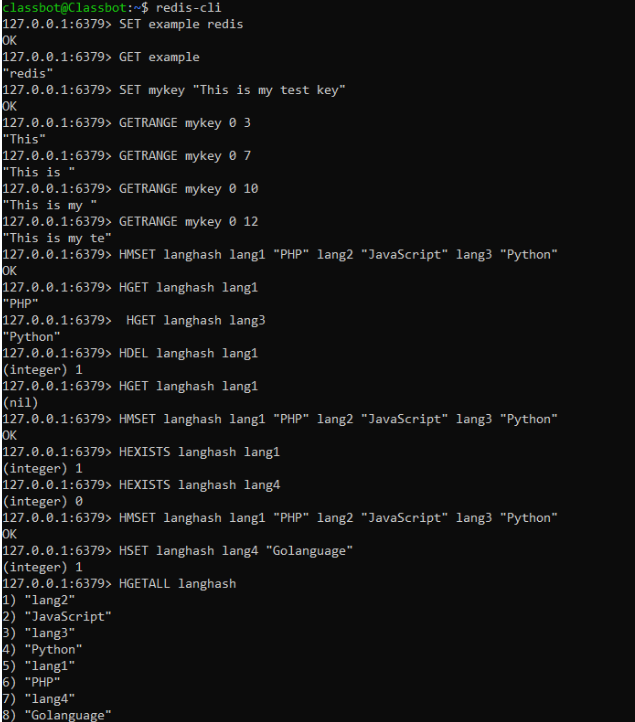
4) "JavaScript"

5) "lang3"

6) "Python"

7) "lang4"

8) "Golanguage"



**Redis HGETALL another example**

127.0.0.1:6379> HSET user email example@gmail.com

(integer) 1

127.0.0.1:6379> HSET user lang English

(integer) 1

127.0.0.1:6379> HSET user gender Male

(integer) 1

127.0.0.1:6379> HGETALL user

1) "email"

2) "example@gmail.com"

3) "lang"

4) "English"

5) "gender"

6) "Male"

**Redis HKEYS**

127.0.0.1:6379> HSET langhash lang1 "PHP"

(integer) 1

127.0.0.1:6379> HSET langhash lang2 "Javascript"

(integer) 1

127.0.0.1:6379> HSET langhash lang3 "Python"

(integer) 1

127.0.0.1:6379> HSET langhash lang4 "Golanguage"

(integer) 1

127.0.0.1:6379> HKEYS langhash

1) "lang1"

2) "lang2"

3) "lang3"

4) "lang4"

## **List Data Type in Redis**

**Redis LPUSH**

127.0.0.1:6379> LPUSH mycolor1 white black

(integer) 2

127.0.0.1:6379> LRANGE mycolor1 0 -1

1) "black"

2) "white"

127.0.0.1:6379> LPUSH mycolor1 red blue

(integer) 4

127.0.0.1:6379> LRANGE mycolor1 0 -1

1) "blue"

2) "red"

3) "black"

4) "white"

**Redis LPOP**

127.0.0.1:6379> LPUSH mycolor1 white black red blue

(integer) 4

127.0.0.1:6379> LRANGE mycolor1 0 -1

1) "blue"

2) "red"

3) "black"

4) "white"

127.0.0.1:6379> LPOP mycolor1

"blue"

127.0.0.1:6379> LRANGE mycolor1 0 -1

1) "red"

2) "black"

3) "white"

127.0.0.1:6379> RPOP mycolor1

"white"

**Redis RPUSH**

127.0.0.1:6379> RPUSH mycolor white black

(integer) 2

127.0.0.1:6379> RPUSH mycolor red blue

(integer) 4

127.0.0.1:6379> LRANGE mycolor 0 -1

1) "white"

2) "black"

3) "red"

4) "blue"

**Redis RPOP**

127.0.0.1:6379> LPUSH mycolor1 white black red blue

(integer) 4

127.0.0.1:6379> LRANGE mycolor1 0 -1

1) "blue"

2) "red"

3) "black"

4) "white"

127.0.0.1:6379> RPOP mycolor1

"white"

127.0.0.1:6379> LRANGE mycolor1 0 -1

1) "blue"

2) "red"

3) "black"

127.0.0.1:6379> RPOP mycolor1

"black"

127.0.0.1:6379> LPOP mycolor1

"blue"

**Redis LRANGE**

127.0.0.1:6379> LPUSH mycolor1 white black red blue

(integer) 4

127.0.0.1:6379> LRANGE mycolor1 0 -1

1) "blue"

2) "red"

3) "black"

4) "white"

127.0.0.1:6379> LRANGE mycolor1 0 1

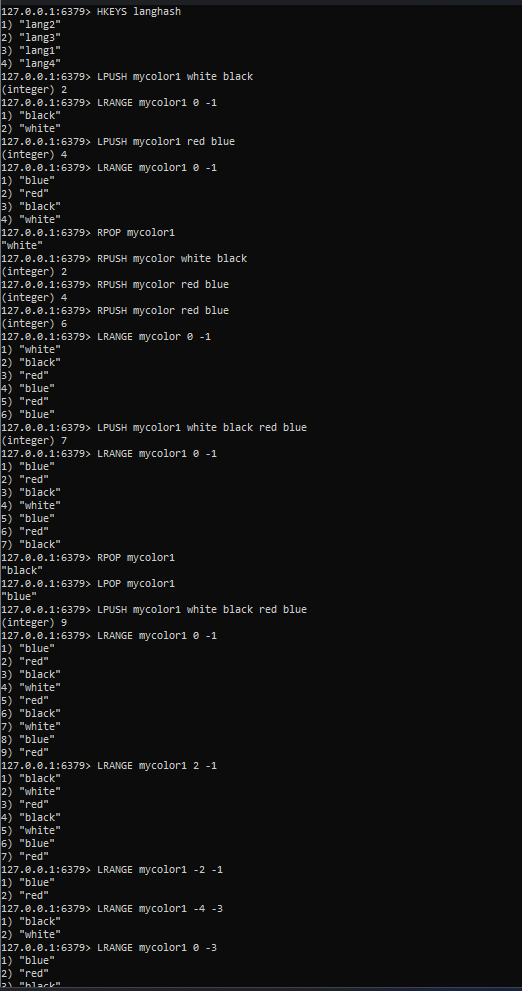
1) "blue"

2) "red"

127.0.0.1:6379> LRANGE mycolor1 2 -1

1) "black"

2) "white"



**Redis LRANGE: using negative right data**

127.0.0.1:6379> LRANGE mycolor1 -2 -1

1) "black"

2) "white"

127.0.0.1:6379> LRANGE mycolor1 -4 -3

1) "blue"

2) "red"

127.0.0.1:6379> LRANGE mycolor1 0 -3

1) "blue"

2) "red"

## **Set Data type in Redis:**

**Redis SADD**

127.0.0.1:6379> SADD mycolor "White"

(integer) 1

127.0.0.1:6379> SADD mycolor "Yellow" "Green"

(integer) 2

127.0.0.1:6379> SADD mycolor "Red" "Blue" "Orange"

(integer) 3

127.0.0.1:6379> SMEMBERS mycolor

1) "Yellow"

2) "White"

3) "Blue"

4) "Green"

5) "Red"

6) "Orange

**Redis SCARD**

127.0.0.1:6379> SADD mycolor "Red" "Green"

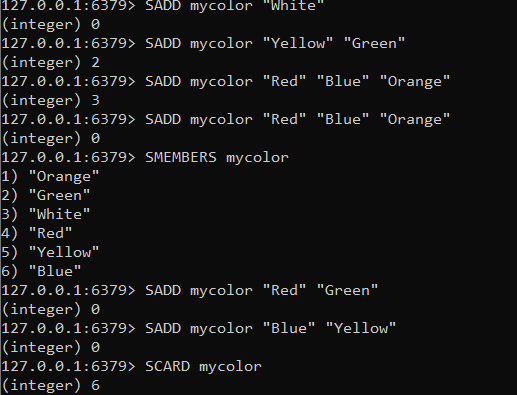
(integer) 2

127.0.0.1:6379> SADD mycolor "Blue" "Yellow"

(integer) 2

127.0.0.1:6379> SCARD mycolor

(integer) 4



**Redis SDIFF**

127.0.0.1:6379> SADD mycolor1 R G B

(integer) 3

127.0.0.1:6379> SADD mycolor2  G Y

(integer) 2

127.0.0.1:6379> sdiff mycolor1 mycolor2

1) "R"

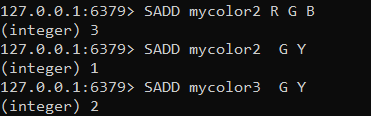
2) "B"

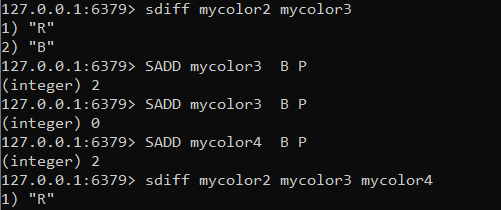
127.0.0.1:6379> SADD mycolor3  B P

(integer) 2

127.0.0.1:6379> sdiff mycolor1 mycolor2 mycolor3

1) "R"





**Redis SISMEMBER**

127.0.0.1:6379> SADD mycolor "red" "green" "blue"

(integer) 3

127.0.0.1:6379> SISMEMBER mycolor "green"

(integer) 1

127.0.0.1:6379> SISMEMBER mycolor "orange"

(integer) 0

**Redis SMEMBER**S

127.0.0.1:6379> SADD mycolor1 "red" "green" "blue"

(integer) 3

127.0.0.1:6379> SADD mycolor1 "orange" "yellow"

(integer) 2

127.0.0.1:6379> SMEMBERS  mycolor1

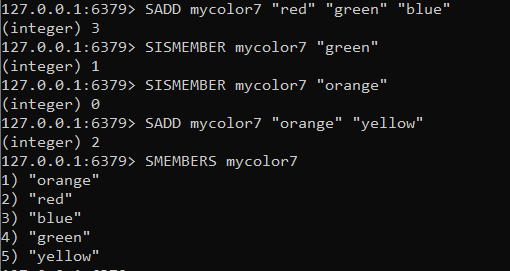
1) "blue"

2) "orange"

3) "green"

4) "red"

5) "yellow"



**Redis SPOP**

127.0.0.1:6379> SADD mycolor1 "red" "green" "blue"

(integer) 3

127.0.0.1:6379> SPOP mycolor1

"red"

127.0.0.1:6379> SMEMBERS mycolor1

1) "green"

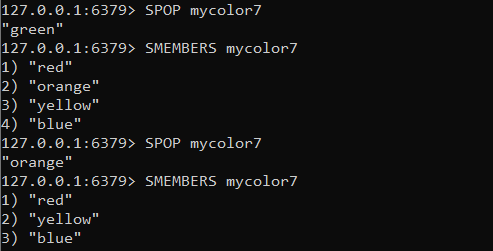
2) "blue"

127.0.0.1:6379> SPOP mycolor1

"green"

127.0.0.1:6379> SMEMBERS mycolor1

1. "blue"



**Redis SINTER of two sets**

127.0.0.1:6379> SADD mycolor1 R G B

(integer) 3

127.0.0.1:6379> SADD mycolor2 G B Y

(integer) 3

127.0.0.1:6379> SINTER mycolor1 mycolor2

1) "G"

2) "B"

## IMG_256

## **Sorted Set in Redis:**

**Redis ZADD**

127.0.0.1:6379> ZADD mycolorset 1 white

(integer) 1

127.0.0.1:6379> ZADD mycolorset 2 black

(integer) 1

127.0.0.1:6379> ZADD mycolorset 3 red

(integer) 1

127.0.0.1:6379> ZRANGE mycolorset 0 -1

1) "white"

2) "black"

3) "red"

127.0.0.1:6379> ZRANGE mycolorset 0 -1 WITHSCORES

1) "white"

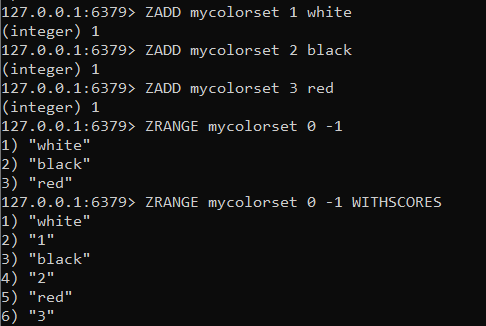
2) "1"

3) "black"

4) "2"

5) "red"

6) "3"



**Redis ZCARD**

127.0.0.1:6379> ZADD mycolorset 10 white 12 black 14 red 16 blue

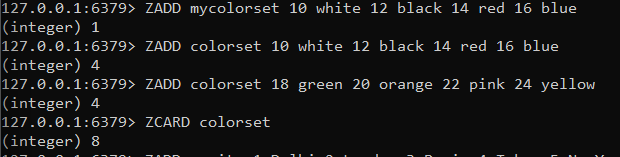
(integer) 4

127.0.0.1:6379> ZADD mycolorset 18 green 20 orange 22 pink 24 yellow

(integer) 4

127.0.0.1:6379> ZCARD mycolorset

(integer) 8



**Redis ZRANGE**

127.0.0.1:6379> ZADD mycity 1 Delhi 2 London 3 Paris 4 Tokyo 5 NewYork 6 Seoul

(integer) 6

127.0.0.1:6379> ZRANGE mycity 0 -1

1) "Delhi"

2) "London"

3) "Paris"

4) "Tokyo"

5) "NewYork"

6) "Seoul"

127.0.0.1:6379> ZRANGE mycity 0 -1 WITHSCORES

 1) "Delhi"

 2) "1"

 3) "London"

 4) "2"

 5) "Paris"

 6) "3"

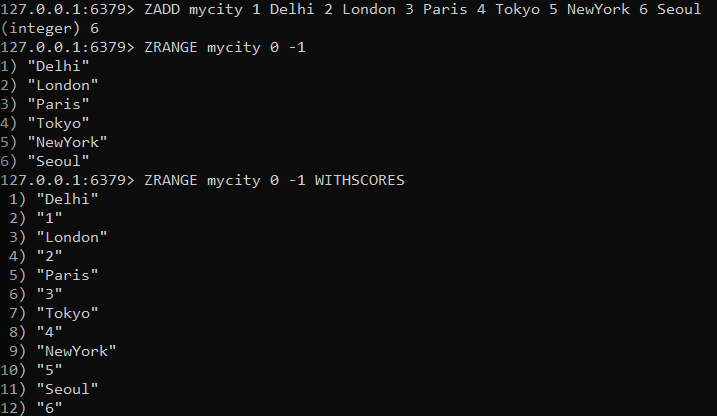
 7) "Tokyo"

 8) "4"

 9) "NewYork"

10) "5"

11) "Seoul"

12) "6"

| **Practical: 03** | **Working with HBase** |
| --- | --- |
| **a)** | **Set up an HBase cluster in a lab environment.** |
| **b)** | **Create an HBase table and define column families.** |
| **c)** | **Insert sample data into the table.** |
| **d)** | **Perform CRUD operations and retrieval of data in HBase.** |

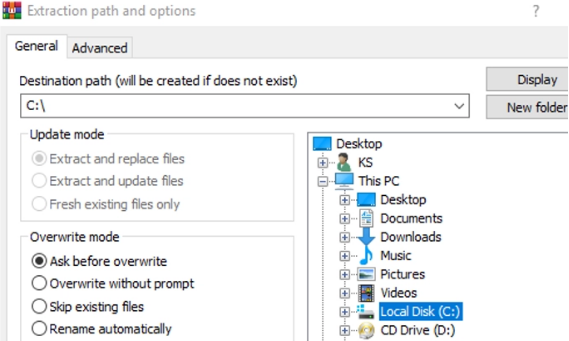
**Hbase Installation**

**Pre-requisites:** We are going to make a standalone setup of HBase in our machine which requires us to

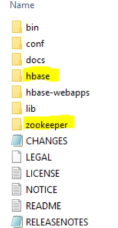
* Install Java JDK 1.8 - We can download and install it from <https://www.oracle.com/java/technologies/downloads/> and set JAVA\_HOME in environment variable.
* Download Hbase-bin - Download Apache Hbase-2.2.5 from <https://archive.apache.org/dist/hbase/2.2.5/>

**Steps:**

* **Step 1 – Extract all files from the archive**

****

* **Step 2 – Create folders named "hbase" and "zookeeper"**



* **Step 3 – Deleting line in HBase.cmd**

Open C:\hbase-2.2.5\bin\hbase.cmd in any text editor.

Search for line **%HEAP\_SETTINGS% and remove the**

**highlighted part.**

set java\_arguments=%HEAP\_SETTINGS%

%HBASE\_OPTS% -classpath "%CLASSPATH%" %CLASS%

%hbase-command-arguments%

* **Step 4 – Adding lines in hbase-env.cmd**

Open C:\hbase-2.2.5\conf\hbase-env.cmd in any text editor.

Add the below lines in the file after the comment session.

set JAVA\_HOME=%JAVA\_HOME%

set HBASE\_CLASSPATH=%HBASE\_HOME%\lib\client-facing-thirdparty\\*

set HBASE\_HEAPSIZE=8000

set HBASE\_OPTS="-XX:+UseConcMarkSweepGC" "-Djava.net.preferIPv4Stack=true"

set SERVER\_GC\_OPTS="-verbose:gc" "-XX:+PrintGCDetails" "-XX:+PrintGCDateStamps" %HBASE\_GC\_OPTS%

set HBASE\_USE\_GC\_LOGFILE=true

set HBASE\_JMX\_BASE="-Dcom.sun.management.jmxremote.ssl=false" "-Dcom.sun.management.jmxremote.authenticate=false"

set HBASE\_MASTER\_OPTS=%HBASE\_JMX\_BASE% "-Dcom.sun.management.jmxremote.port=10101"

set HBASE\_REGIONSERVER\_OPTS=%HBASE\_JMX\_BASE% "-Dcom.sun.management.jmxremote.port=10102"

set HBASE\_THRIFT\_OPTS=%HBASE\_JMX\_BASE% "-Dcom.sun.management.jmxremote.port=10103"

set HBASE\_ZOOKEEPER\_OPTS=%HBASE\_JMX\_BASE% -Dcom.sun.management.jmxremote.port=10104"

set HBASE\_REGIONSERVERS=%HBASE\_HOME%\conf\regionservers

set HBASE\_LOG\_DIR=%HBASE\_HOME%\logs

set HBASE\_IDENT\_STRING=%USERNAME%

set HBASE\_MANAGES\_ZK=true

* **Step 5 – Adding lines in hbase-site.xml**

Open C:\hbase-2.2.5\conf\hbase-site.xml in any text editor.

Add the below lines inside the <configuration> tag.

<property>

    <name>hbase.rootdir</name>

    <value>file:///C:/Documents/hbase-2.2.5/hbase</value> </property>

 <property>

    <name>hbase.zookeeper.property.dataDir</name>

    <value>/C:/Documents/hbase-2.2.5/zookeeper</value>

 </property>

 <property>

     <name> hbase.zookeeper.quorum</name>

    <value>localhost</value>

 </property>

* **Step 6 – Setting Environment Variables:**

Input the following:

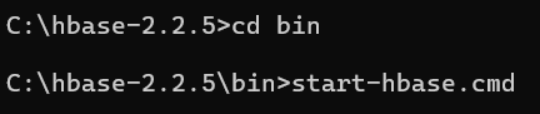
* Variable name: HBASE\_HOME
* Variable Value: Put the path of the Hbase folder.

**Starting the HBASE shell:**

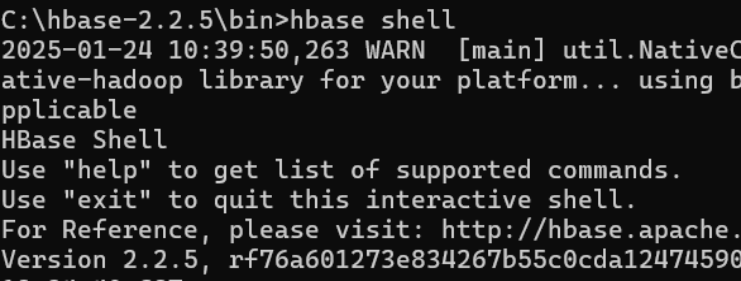
* **Step 1 – Use start-hbase.cmd:**

Go to the bin folder of hbase and start a cmd prompt there

Then type: **start-hbase.cmd**

****

* **Step 2 – Start hbase shell:**  Type: **hbase shell**



In HBase, interactive shell mode is used to interact with HBase for table operations, table management, and data modeling.

**Hbase Commands**

**General commands**

In Hbase, general commands are categorized into following commands

* **Status**
* **Version**
* **Table\_help ( scan, drop, get, put, disable, etc.)**
* **Whoami**

To get enter into HBase shell command, first of all, we have to execute the code as mentioned below

**hbase Shell**

Once we get to enter into HBase shell, we can execute all shell commands mentioned below. With the help of these commands, we can perform all types of table operations in the HBase shell mode.

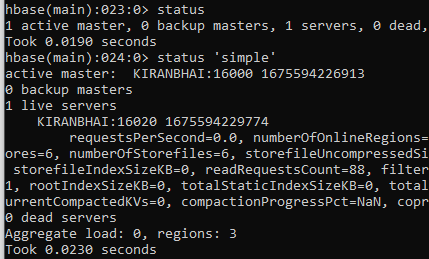
Let us look into all of these commands and their usage one by one with an example.

**Status**

Syntax: status

This command will give details about the system status like a number of servers present in the cluster, active server count, and average load value. You can also pass any particular parameters depending on how detailed status you want to know about the system. The parameters can be **‘summary’, ‘simple’, or ‘detailed’**, the default parameter provided is “summary”.

**Example:**



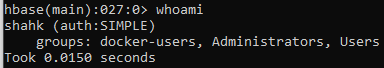
**whoami**

Syntax:

Syntax: Whoami

This command “whoami” is used to return the current HBase user information from the HBase cluster.

Example:



**Tables Managements commands**

These commands will allow programmers to create tables and table schemas with rows and column families.

The following are Table Management commands

* Create
* List
* Describe
* Disable
* Disable\_all
* Enable
* Enable\_all
* Drop
* Drop\_all
* Show\_filters
* Alter
* Alter\_status

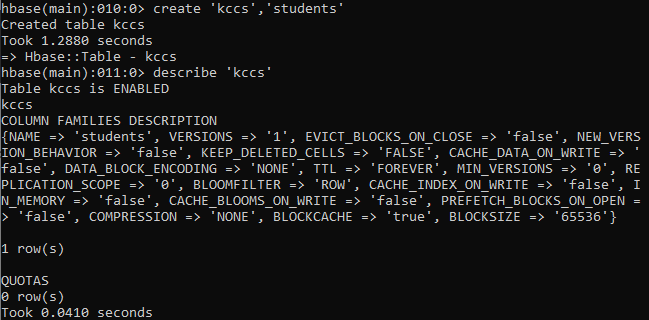
Let us look into various command usage in HBase with an example.

**Create**

Syntax: create <table name>, <column family name>

**Example:**

Creating a table named ‘kccs’ with column family ‘students’

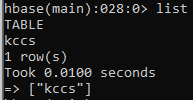


The above example explains how to create a table in HBase with the specified name given according to the dictionary or specifications as per column family. In addition to this we can also pass some table-scope attributes as well into it.

In order to check whether the table ‘kccs’ is created or not, we have to use the **“list”** command as mentioned below.

**List**

Syntax:list

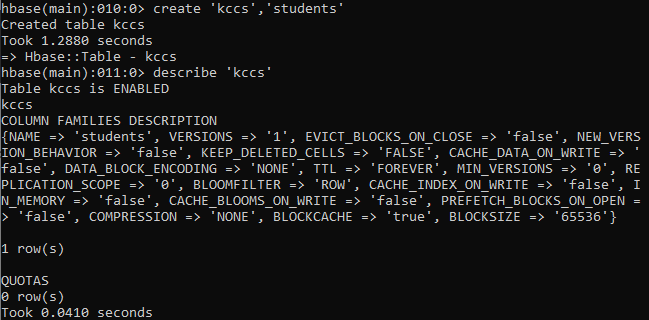


* “List” command will display all the tables that are present or created in HBase
* The output showing in above screen shot is currently showing the existing tables in HBase
* We can filter output values from tables by passing optional regular expression parameters

**Describe**

Syntax:describe <table name>

**Example:**



hbase(main):011:0>describe 'kccs'

This command describes the named table.

* It will give more information about column families present in the mentioned table
* In our case, it gives the description about table “kccs”
* It will give information about table names with column families, associated filters, versions and some more details.

**disable**

Syntax: disable <tablename>

**Example:** disable 'kccs'





* This command will start disabling the named table
* If table needs to be deleted or dropped, it has to disabled first

**disable\_all**

Syntax: disable\_all<"matching regex">

* This command will disable all the tables matching the given regex.
* The implementation is same as delete command (Except adding regex for matching)
* Once the table gets disable the user can able to delete the table from HBase
* Before delete or dropping table, it should be disabled first

**Enable**

Syntax: enable <tablename>

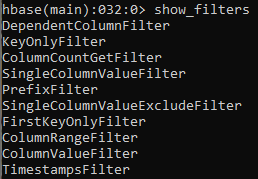
**Example:** enable 'kccs’



* This command will start enabling the named table
* Whichever table is disabled, to retrieve back to its previous state we use this command
* If a table is disabled in the first instance and not deleted or dropped, and if we want to re-use the disabled table then we have to enable it by using this command.

**show\_filters**

Syntax: show\_filters

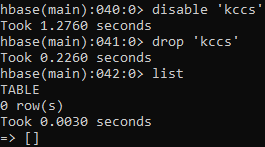


This command displays all the filters present in HBase like ColumnPrefix Filter, TimestampsFilter, PageFilter, FamilyFilter, etc.

**drop**

Syntax:drop <table name>

**Example:**



We have to observe below points for drop command

* To delete the table present in HBase, first we have to disable it
* To drop the table present in HBase, first we have to disable it
* So either table to drop or delete first the table should be disable using disable command
* Before execution of this command, it is necessary that we disable table “kccs”

**drop\_all**

Syntax: drop\_all<"regex">

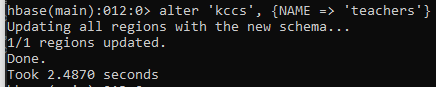
* This command will drop all the tables matching the given regex
* Tables have to disable first before executing this command using disable\_all
* Tables with regex matching expressions are going to drop from HBase

**alter**

Syntax: alter <table name>, NAME=><column family name>, VERSIONS=>5

This command alters the column family schema. To understand what exactly it does, we have explained it here with an example.

**Example:** Altering table kccs, by adding a new column family named ‘teachers’



**Data manipulation commands**

These commands will work on the table related to data manipulations such as putting data into a table, retrieving data from a table and deleting schema, etc.

The commands come under these are

* Count
* Put
* Get
* Delete
* Delete all
* Truncate
* Scan

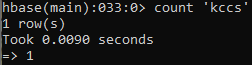
Let's look into these commands usage with an example.

**Count**

Syntax: count <'tablename'>, CACHE =>1000

* The command will retrieve the count of a number of rows in a table. The value returned by this one is the number of rows.
* Current count is shown per every 1000 rows by default.
* Count interval may be optionally specified.
* Default cache size is 10 rows.
* Count command will work fast when it is configured with the right Cache.

**Example:**



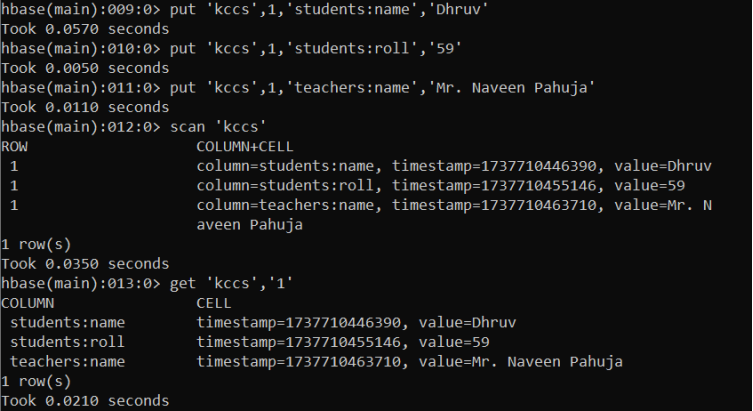
**Put**

Syntax: put <'table name'>,<'rowname'>,<'column value'>,<'value'>

This command is used for following things

* It will put a cell ‘value’ at a defined or specified table or row or column.
* It will optionally coordinate time stamps.

**Example:**



* hbase> put 'kccs',1,'students:name','Dhruv'

Here we are placing values into table “kccs” under row 1, column family ‘students’ and column ‘name’

* hbase> put 'kccs',1,'students:roll','59'

Here we are placing values into table “kccs” under row 1, column family ‘students’ and column ‘roll’

* hbase> put 'kccs',1,'teachers:name','Mr. Naveen Pahuja'

Here we are placing values into table “kccs” under row 1, column family ‘teachers’ and column ‘name’

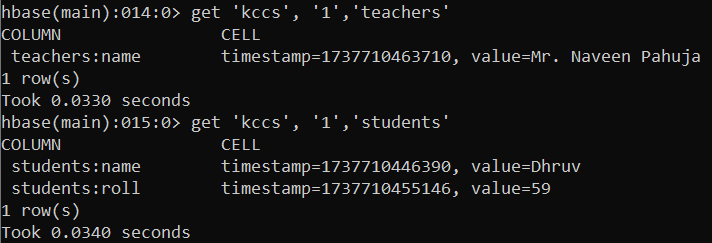
**Get**

Syntax: get <'table name'>, <'rowname'>, {< Additional parameters>}

Here <Additional Parameters> include TIMERANGE, TIMESTAMP, VERSIONS and FILTERS.

By using this command, you will get a row or cell contents present in the table. In addition to that you can also add additional parameters to it like TIMESTAMP, TIMERANGE,VERSIONS, FILTERS, etc. to get a particular row or cell content.

**Example:**

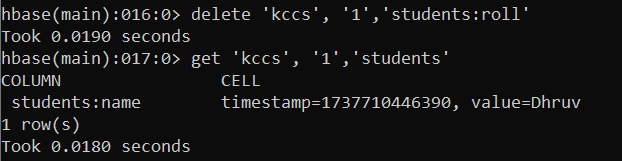


**Delete**

Syntax:delete <'tablename'>,<'row name'>,<'column name'>

* This command will delete the cell value at a defined table of row or column.
* Delete must and should match the deleted cells coordinates exactly.
* When scanning, delete cell suppresses older versions of values.

**Example:** hbase> delete 'kccs',1,'students:roll'



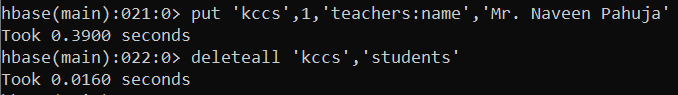
* The above execution will delete column ‘roll’ in the column family ‘students’ from the table ‘kccs’

**deleteall**

Syntax: deleteall <'tablename'>, <'rowname'>

* This Command will delete all cells in a given row.
* We can optionally define column names and time stamps to the syntax.
* Optionally we can mention column names in that.

**Example:**



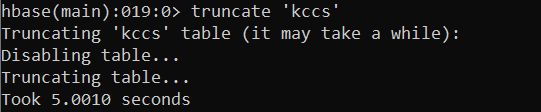
**Truncate**

Syntax: truncate <tablename>

After truncating an hbase table, the schema will present but not the records. This command performs 3 functions; those are listed below

* Disables table if it already presents
* Drops table if it already presents
* Recreates the mentioned table

**Example:**

****

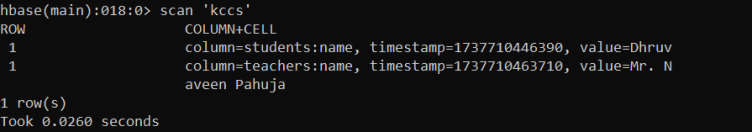
**Scan**

Syntax: scan <'tablename'>, {Optional parameters}

This command scans the entire table and displays the table contents.

* We can pass several optional specifications to this scan command to get more information about the tables present in the system.
* Scanner specifications may include one or more of the following attributes.
* These are TIMERANGE, FILTER, TIMESTAMP, LIMIT, MAXLENGTH, COLUMNS, CACHE, STARTROW and STOPROW.

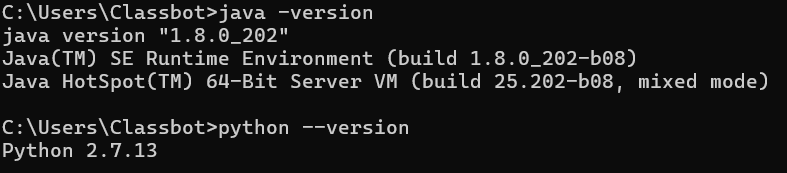
**Example:**



| **Practical No. 4** | **Apache Cassandra Operations** |
| --- | --- |
| **a** | **Install and configure Apache Cassandra in a lab environment.** |
| **b** | **Create a keyspace and define a table schema.** |
| **c** | **Insert data into the table.** |
| **d** | **Perform CRUD operations and query data from Apache Cassandra.** |

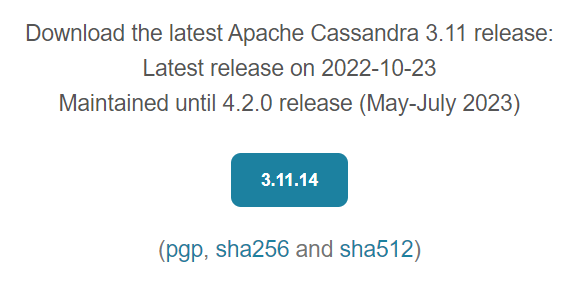
**STEPS:**

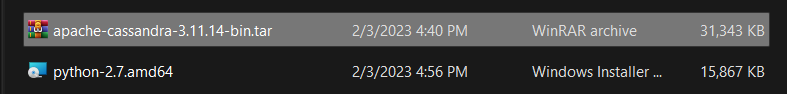
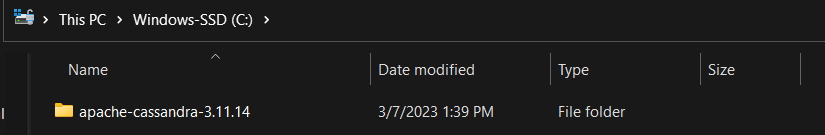
Prerequisites:

1. Python 2.7
2. Java jdk 1.8 

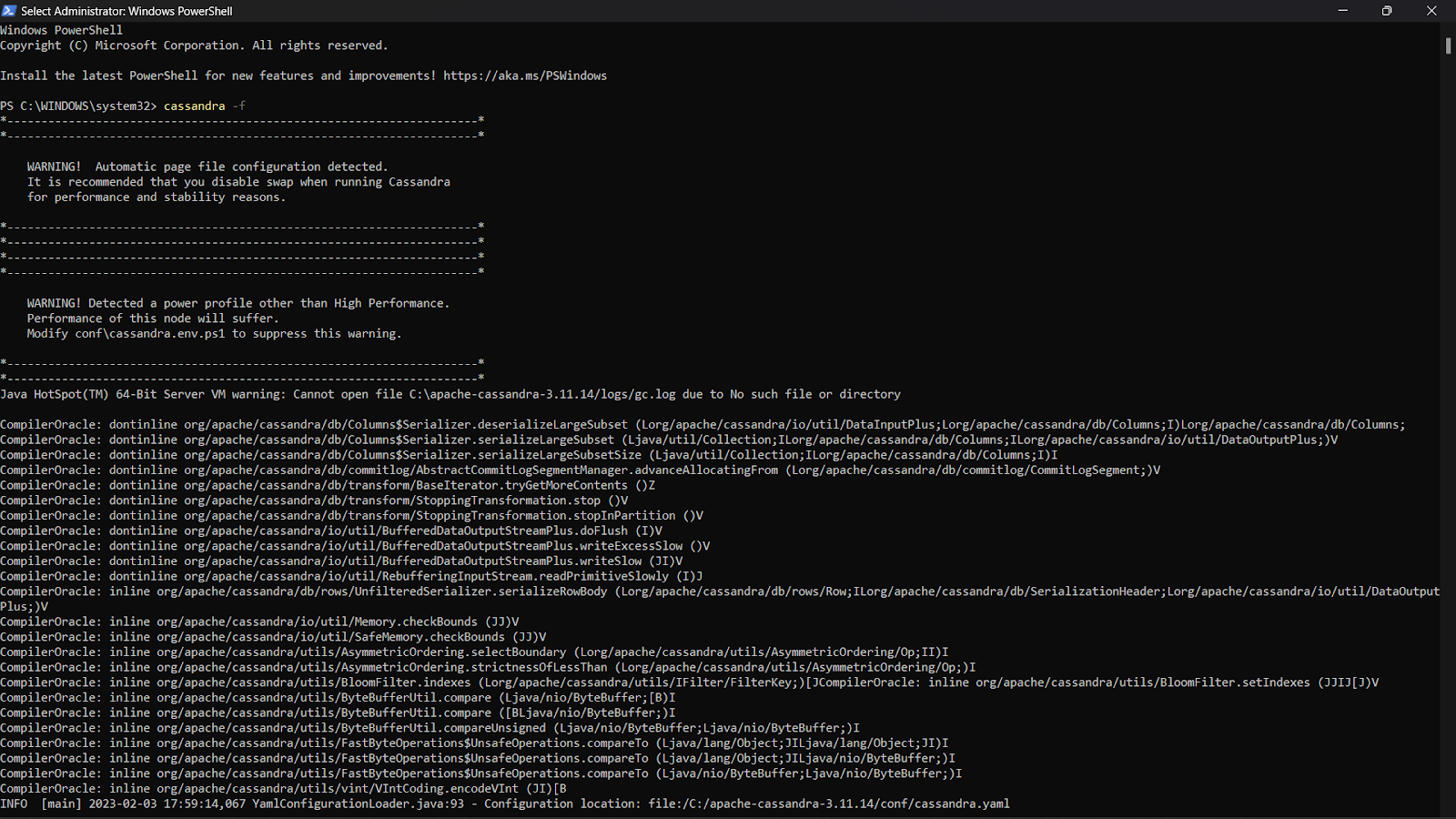
Add directory path in user variables.

1. C:\Program Files\Java\jdk1.8.0\_121\bin
2. C:\Python27

[https://cassandra.apache.org/\_/download.html](https://cassandra.apache.org/_/download.html*)

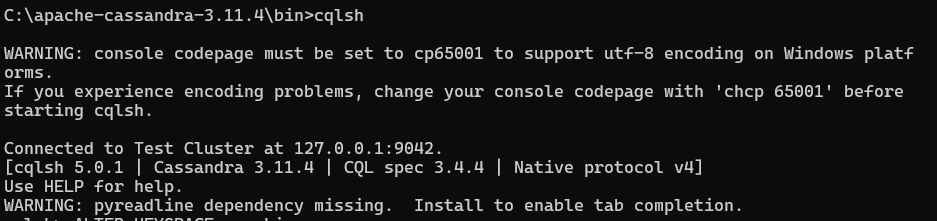
Compress the tar file. Unzip it into the c drive C:\apache-cassandra-3.11.14\bin

Add the above path to user variables as well.

**Type: Cassandra -f in command prompt**

Now keep the current prompt on, while opening a new command prompt.

**Type: cqlsh**



**1. Keyspace Operations:**

**Create Keyspace:**

cqlsh> CREATE KEYSPACE tutorialspoint WITH replication = {'class':'SimpleStrategy', 'replication\_factor' : 1};

cqlsh> CREATE KEYSPACE ayushi  WITH replication = {'class':'SimpleStrategy', 'replication\_factor' : 1};

**Verification:**

cqlsh> DESCRIBE KEYSPACES;

ayushi          system\_schema  system              system\_traces

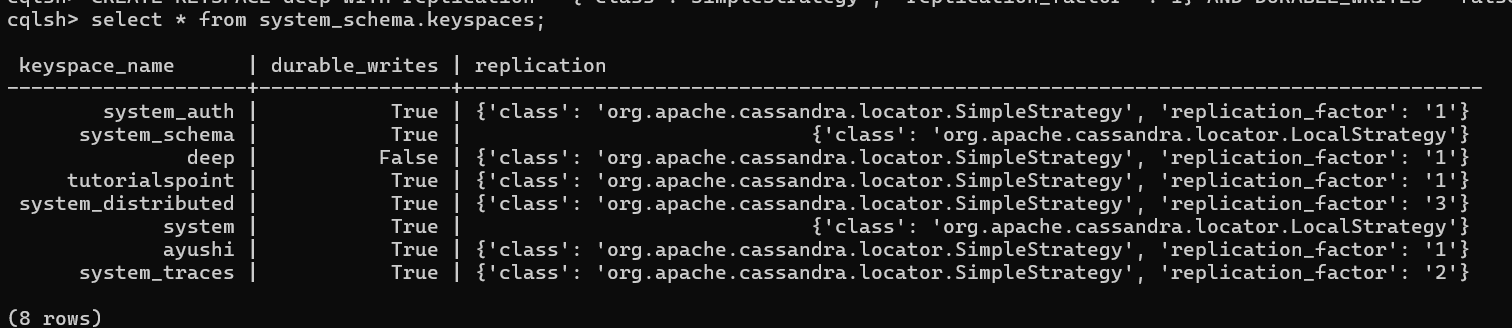
tutorialspoint  system\_auth    system\_distributed

**Durable\_writes:**

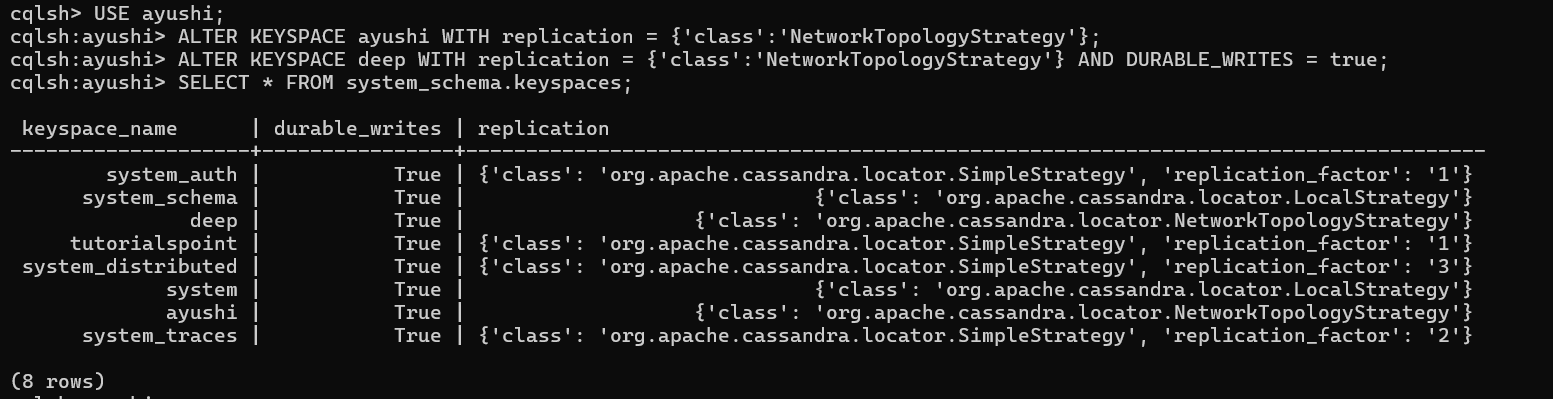
By default, the durable\_writes properties of a table is set to true, however it can be set to false.

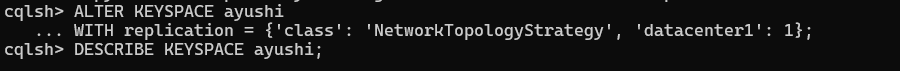
cqlsh> CREATE KEYSPACE deep WITH replication = {'class':'SimpleStrategy', 'replication\_factor' : 1} AND DURABLE\_WRITES = false;

**Verification:**

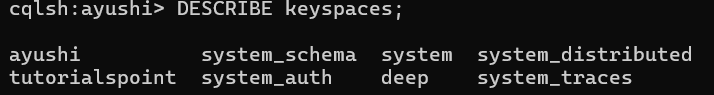


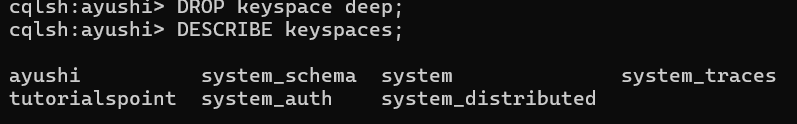
**Using a Keyspace:**



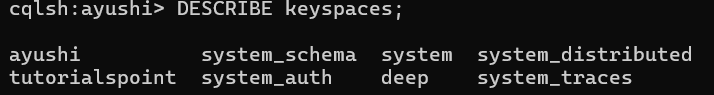


**Dropping a Keyspace:**



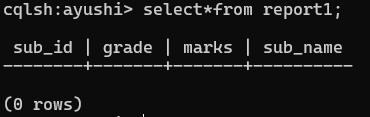


**2. Table Operations**

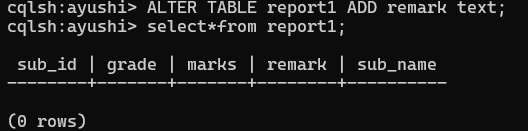


**Creating a Table:**

CREATE TABLE report(sub\_id int PRIMARY KEY,sub\_name text,marks varint,grade text);

****

**Altering a Table:**

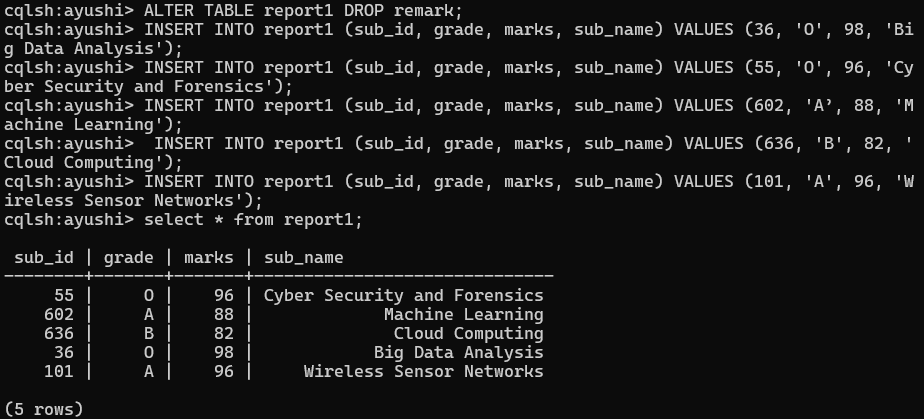
****

**Dropping a Column:**

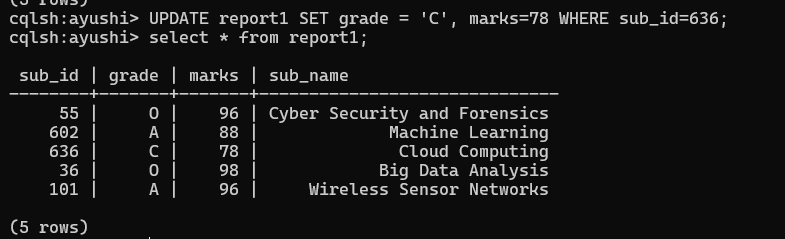
cqlsh:ayushi> ALTER TABLE report DROP remark;

**3. CRUD Operations:**

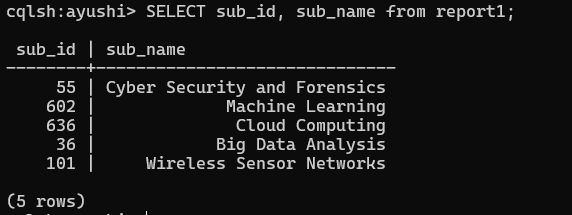
**Creating Data in Table:**

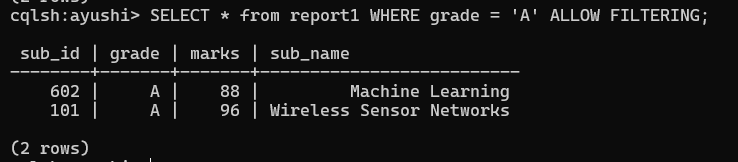
****

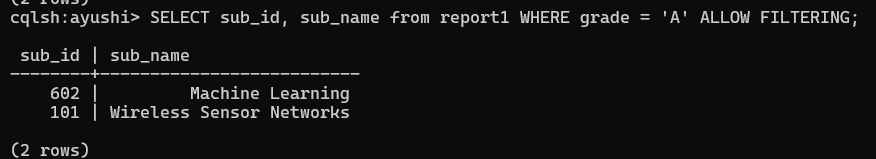
**Updating Data in a Table:**

****

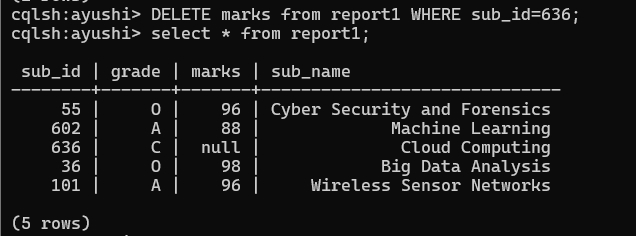
**Reading Required Columns:**

****

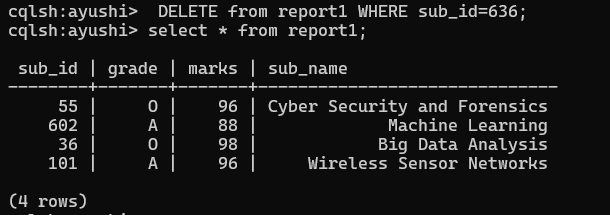
****

****

**Deleting Datafrom a Table:**

****

**Deleting an Entire Row:**

****

| **Practical No 5** | **Querying MongoDB** |
| --- | --- |
| A. | **Write and Execute MongoDB queries to retrieves specific data from collection.** |

**Note: start server and shell.**

**Import given database file.**

Server (Mongod.exe)

1. Before starting the server make sure following things:
   1. Path of bin directory (located in C:\Program Files\MongoDB\Server\7.0\bin)
   2. Path of db directory (you must explicitly create this directory in **C:\MSC-CS\sem-one\noSql-Practicals\data\db**)
2. Open command prompt
3. Change directory :C:>**cd C:\Program Files\MongoDB\Server\7.0\bin**
4. Start server:C:\Program Files\….....\bin\>**mongod.exe --dbpath C:\MSC-CS\sem-one\noSql-Practicals\restaurants.json**
5. Shell (Mongo.exe)
6. Open new command prompt
7. Change directory : C:>**cd C:\Program Files\MongoDB\Server\7.0\bin**
8. Start shell: C:\Program Files\ \bin\>**mongo.exe**

Import given “restaurants.json” file (mongoimport.exe)

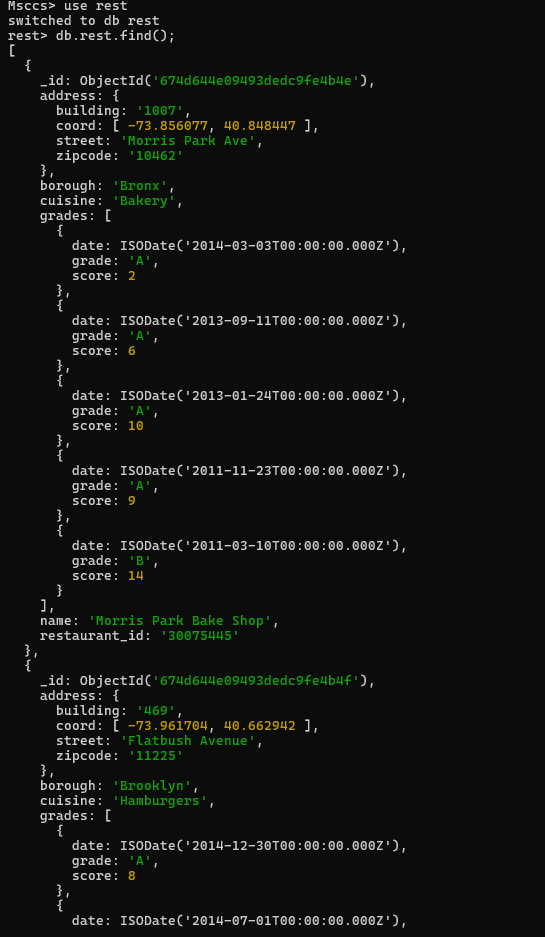
1. Open new command prompt
2. Change directory : C:>**cd C:\Program Files\MongoDB\Server\7.0\bin**
3. C:\……..\bin>mongoimport --db rest --collection rest <C:\MSC-CS\sem-one\noSql-Practicals/restaurants.json
4. Execute following commands on shell.

>use rest

//Switched to database rest

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

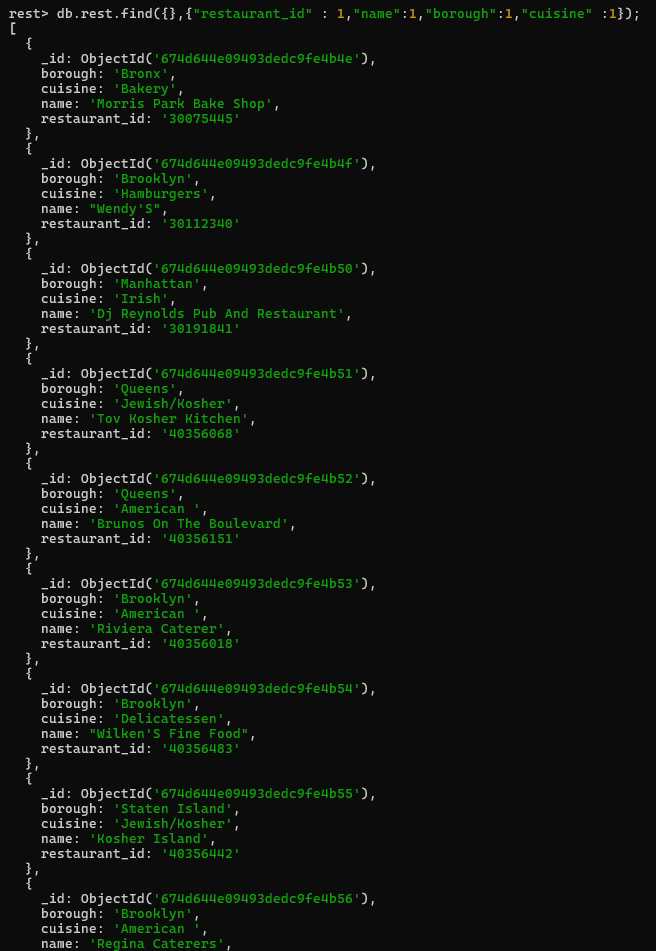
db.rest.find();

****

Type it to load more.

1. Write a MongoDB query to display the fields, restaurant\_id, name, borough and cuisine for all the documents in the collection restaurant.

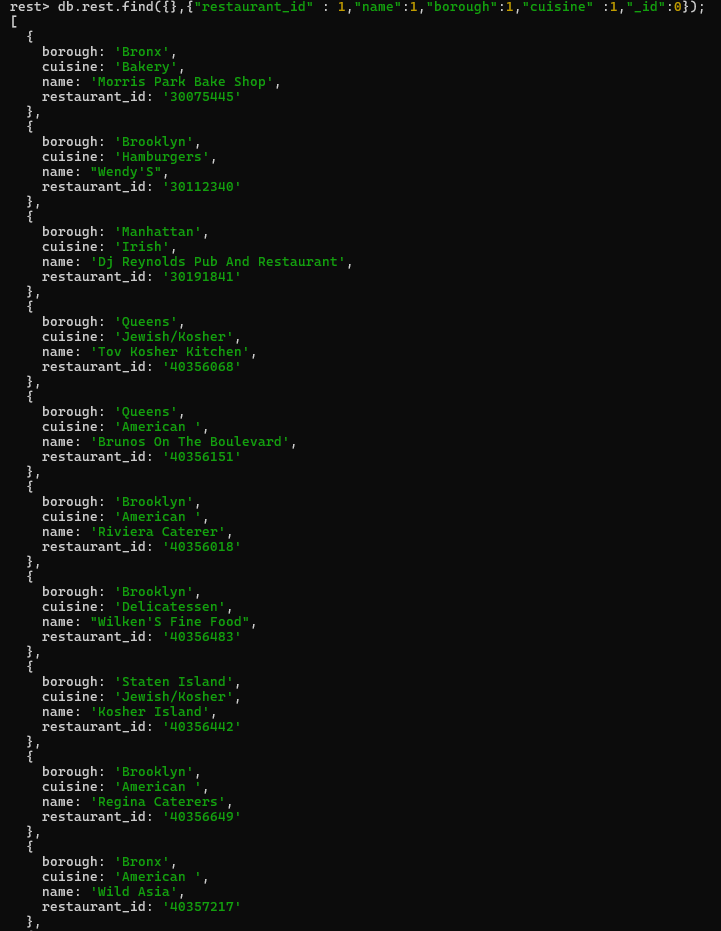
db.rest.find({},{"restaurant\_id" : 1,"name":1,"borough":1,"cuisine" :1});



Type it to load more.

1. Write a MongoDB query to display the fields restaurant\_id, name, borough and cuisine, but exclude the field \_id for all the documents in the collection restaurant.

db.rest.find({},{"restaurant\_id" : 1,"name":1,"borough":1,"cuisine" :1,"\_id":0});



Type it to load more.

1. Write a MongoDB query to display the fields restaurant\_id, name, borough and zip code, but exclude the field \_id for all the documents in the collection restaurant.

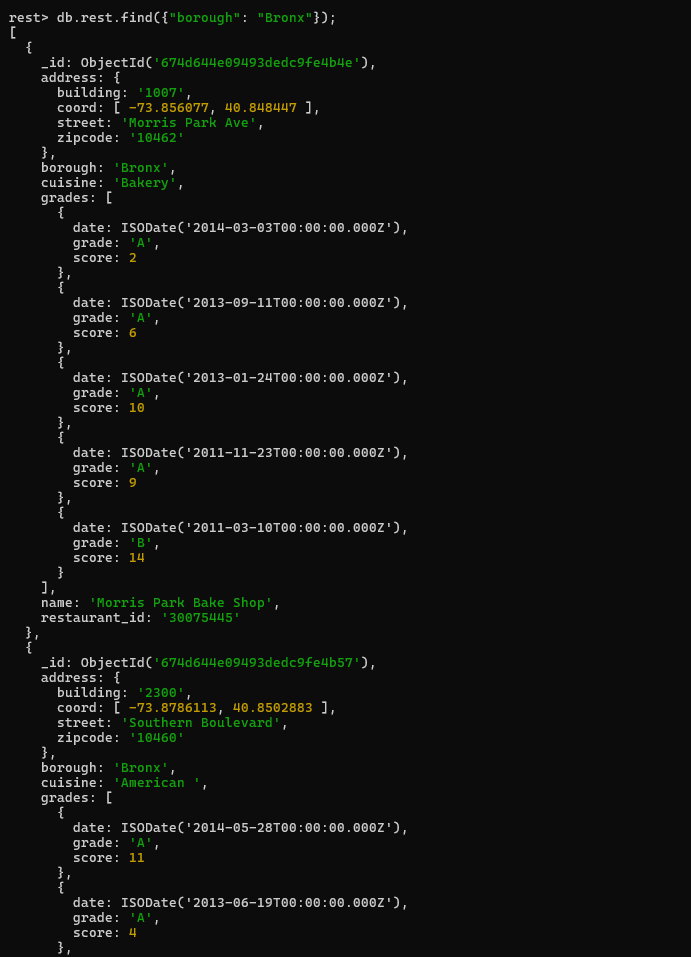
db.rest.find({},{"restaurant\_id" :1,"name":1,"borough":1,"address.zipcode" :1, "\_id":0});



Type it to load more.

1. Write a MongoDB query to display all the restaurants which are in the borough Bronx.

db.rest.find({"borough": "Bronx"});

****

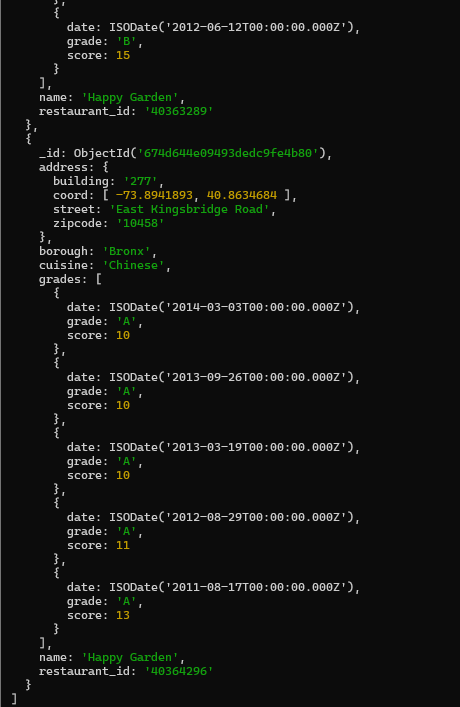
Type it to load more.

1. Write a MongoDB query to display the first 5 restaurants which are in the borough Bronx.

db.rest.find({"borough": "Bronx"}).limit(5);







1. Write a MongoDB query to display the next 5 restaurants after skipping the first 5 which are in the borough Bronx.

db.rest.find({“borough”: “Bronx”}).skip(5).limit(5);

****

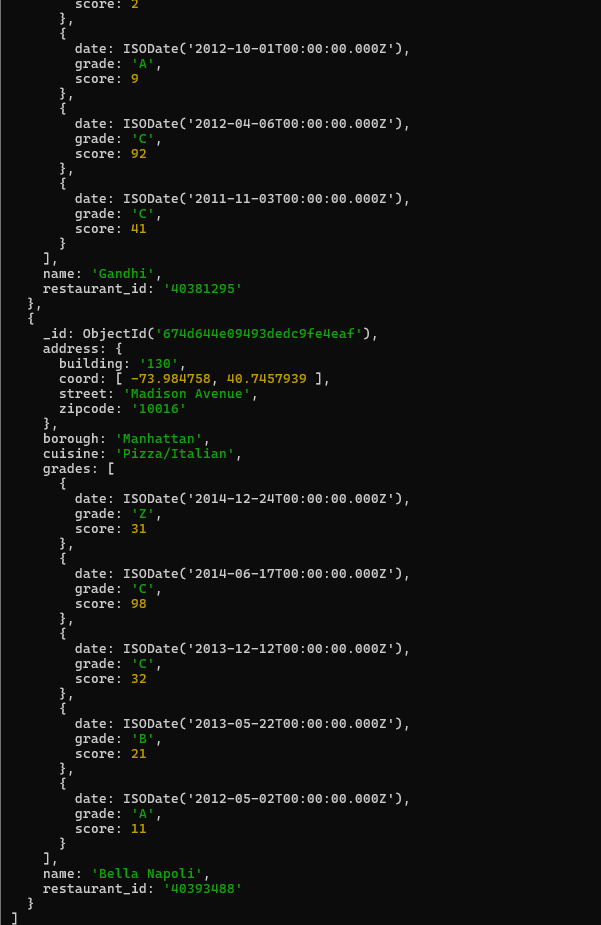
****

****

1. Write a MongoDB query to find the restaurants who achieved a score more than 90.

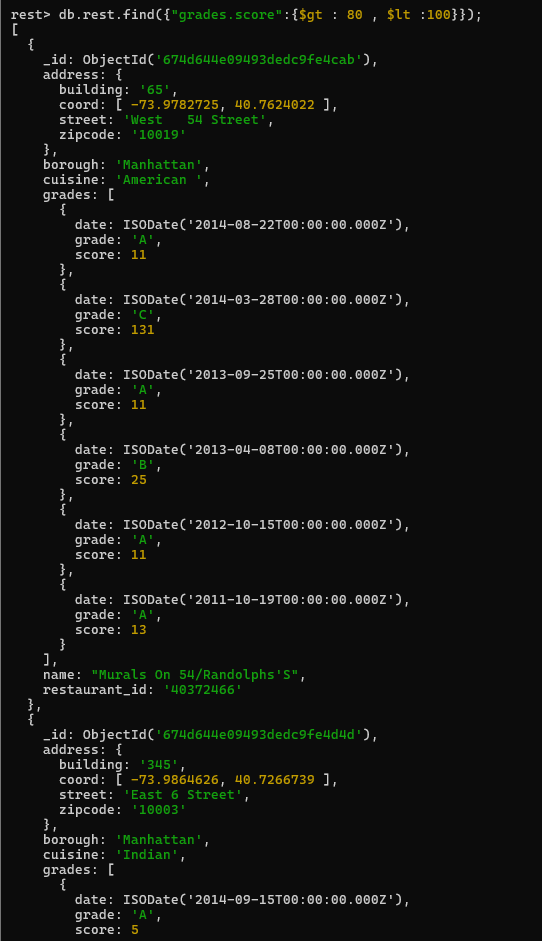
db.rest.find({grades : { $elemMatch:{"score":{$gt : 90}}}});

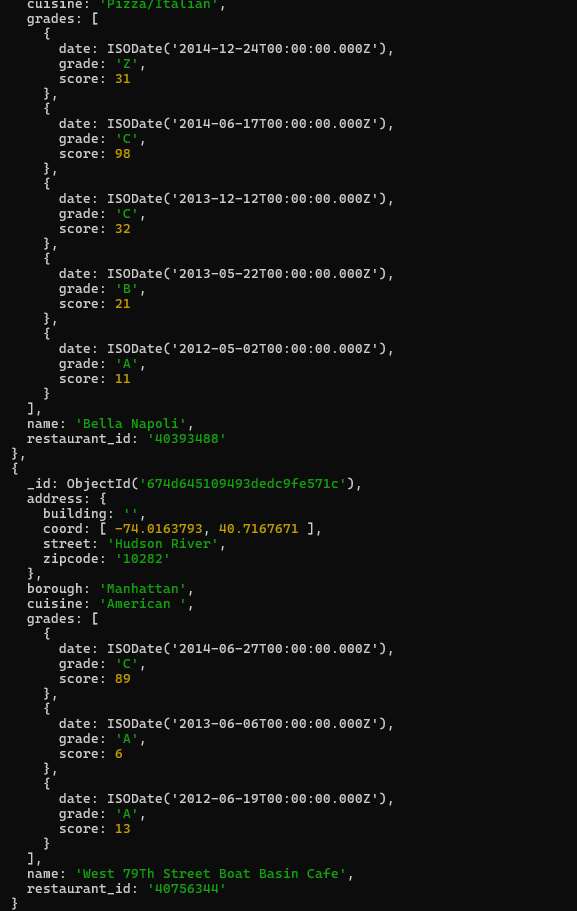




1. Write a MongoDB query to find the restaurants that achieved a score, more than 80 but less than 100.

db.rest.find({“grades.score":{$gt : 80 , $lt :100}});





1. Write a MongoDB query to find the restaurants which are located in latitude value less than - 95.754168.

db.rest.find({"address.coord" : {$lt : -95.754168}});

****

****

| **Practical No 6** | **Implementing Indexing in MongoDB** |
| --- | --- |
| A. | Create an index on a specific field in a MongoDB collection. |
| B. | Measure the impact of indexing on query performance. |

**Note: start server and shell.**

**Import given database file.**

Server (Mongod.exe)

1. Before starting the server make sure following things:
   1. Path of bin directory (located in C:\Program Files\MongoDB\Server\7.0\bin)
   2. Path of db directory (you must explicitly create this directory in C:\MSC-CS\sem-one\noSql-Practicals\data\db)
2. Open command prompt
3. Change directory :

C:>**cd C:\Program Files\MongoDB\Server\7.0\bin**

1. Start server:

C:\Program Files\….....\bin\>**mongod.exe --dbpath C:\MSC-CS\sem-one\noSql-Practicals\data\db**

Shell (Mongosh.exe)

1. Open new command prompt
2. Change directory : C:>**cd C:\Program Files\MongoDB\Server\7.0\bin**
3. Start shell: C:\Program Files\ \bin\>**mongosh.exe**

Import given files (mongoimport.exe)

1. Open new command prompt
2. Change directory : C:>**cd C:\Program Files\MongoDB\Server\7.0\bin**
3. C:\……..\bin>mongoimport --type csv --headerline --db  Msccs  --collection          movies  ---file C:\MSC-CS\sem-one\noSql-Practicals\movies.csv --uri mongodb://localhost:27017/
4. C:\……..\bin>mongoimport --type csv --headerline --db  Msccs  --collection             ratings ---file C:\MSC-CS\sem-one\noSql-Practicals\ratings.csv --uri mongodb://localhost:27017/

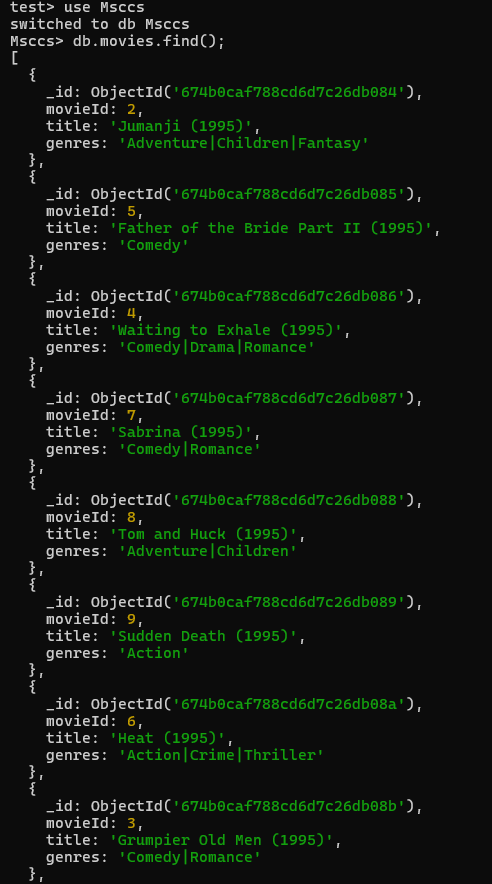
Once all the files are exported , Execute following commands on shell.

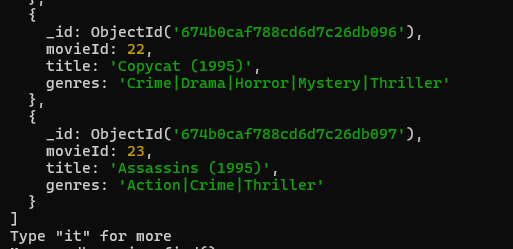
>use Msccs

switched to db Msccs

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

db.movies.find();





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

db.ratings.find();



Type it for more.

* 1. Write a MongoDB query to display the query performance.

db.ratings.find().explain(“allPlansExecution”);



* 1. Write a MongoDB query to display all the documents having movieId 2356.

db.ratings.find({movieId: 2356});



* 1. Write a MongoDB query to display query performance.

db.ratings.find({movie\_id: 2356}).explain(“allPlansExecution”);



* 1. Write a MongoDB query to create the index on movieId field.

db.ratings.ensureIndex({ movie\_id:1 });



* 1. Write a query to create an index with keys sorted in descending order.

db.ratings.ensureIndex({ movie\_id:-1 });

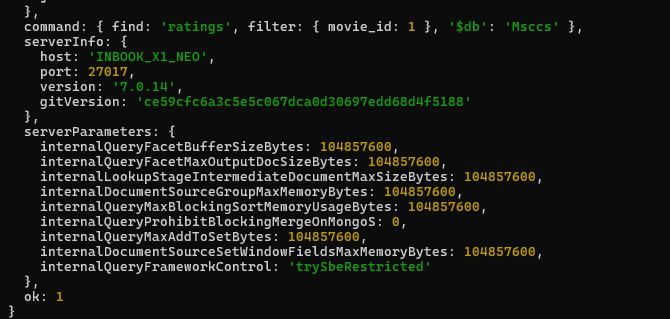


* 1. Write a MongoDB query to display query performance after index.

db.ratings.find({movie\_id: 1}).explain(“allPlansExecution”);

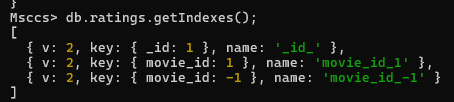






* 1. Write a MongoDB query to get a list of all indexes on ratings collection.

db.ratings.getIndexes();



* 1. Write a MongoDB query to drop an index on movieId.

db.ratings.dropIndex({ movie\_id:-1 });



* 1. Write a MongoDB query to drop all indexes on ratings collection.

db.ratings.dropIndexes();

