Register No.................................... Date:................................

Lab Manual

NOSQL DATABASES LAB

III Year – II Semester VR -20

Name of the Student:____

Regd.No.:________________ Section:________________________________

Department:____

Year: 20_____-20____

Jerripotula Priyanka, Asst Prof. CSE

VIGNAN'S INSTITUTE OF INFORMATION AND TECHNOLOGY, DUVVADA

Register No.................................... Date:..................................

VIGNAN's INSTITUTE OF INFORMATION TECHNOLOGY
(AUTONOMOUS)
(Approved by AICTE-New Delhi & Affiliated to JNTU-GV, Vizianagaram)
Beside VSEZ, Duvvada, Vadlapudi Post, Gajuwaka, Visakhapatnam - 530 049.

Department of
Computer Science and Engineering

CERTIFICATE

This is to certify that Mr./Ms...bearing the Regd. No. ....is a student of..B. Tech............... Semester has successfully completed........................ experiments in.. .... Lab during the Academic Year 2023-2024.

Faculty In-charge Head of the Department

External Examiner

Register No.................................. Date:...............................

INDEX

Sl. No	Date	Title of the Experiment	Skill	Page No.	Marks	Sign. With Date
1		**Exercise – 1** Introduction to MongoDB and its Installation on Windows & Linux	MongoDB Installation	1		
2		**Exercise – 2** Description of mongo Shell, Create database and show database.	Database Creation	11		
3		**Exercise – 3** Commands for MongoDB and to study operations in MongoDB – Insert, Query, Update, Delete and Projection	MongoDB Commands	13		
4		**Exercise – 4** Download a zip code dataset at http://media.mongodb.org/zips.json .Use mongo import to import the zip code dataset into MongoDB. After importing the data, answer the following questions by using aggregation pipelines: 1. Find all the states that have a city called “BOSTON”. Find all the states and cities whose names include the string “BOST”. 2. Each city has several zip codes. Find the city in each state with the greatest number of zip codes and rank those cities along with the states using the city populations. MongoDB can query on spatial information	MongoDB	17		
5		**Exercise – 5** Where Clause equivalent in MongoDB: Write a MongoDB query to find the restaurants that achieved a score, more than 80 but less than 100	MongoDB – where clause	21		
6		**Exercise – 6** Write a MongoDB query to find the restaurants which locate in latitude value less than -95.754168.	MongoDB – Query	23		
7		**Exercise – 7** To study operations in MongoDB – AND in MongoDB, OR in MongoDB, Limit Records and Sort Records. To study operations in MongoDB –	MongoDB Operations	25		

Register No.................................. Date:.................................

		Indexing, Advanced Indexing, Aggregation and Map Reduce				
8		**Exercise – 8** Column oriented databases study, queries, and practices	Column Oriented database	30		
9		**Exercise – 9** Create a database that stores road cars. Cars have a manufacturer, a type. Each car has a maximum performance and a maximum torque value. Do the following: Test Cassandras replication schema and consistency models.	Cassandra	33		
10		**Exercise – 10** Master Data Management using Neo4j Manage your master data more effectively the world of master data is changing. Data architects and application developers are swapping their relational databases with graph databases to store their master data. This switch enables them to use a data store optimized to discover new insights in existing data, provide a360-degree view of master data and answer questions about data relationships in real time	Data Management	45		
11		**Exercise – 11** Shopping Mall case study using Cassandra, where we have many customers ordering items from thermall and, we have suppliers who deliver them their ordered items	Cassandra	48		

Register No.	Date:

Register No.................................. Date:................................

Exercise-1

Introduction to Mongodb

MongoDB: Revolutionizing Data Management and Its Significance

In today's digital age, data is the lifeblood of businesses, fueling insights, innovation, and competitive advantage. However, traditional relational databases struggle to keep pace with the increasing volume, velocity, and variety of data generated daily. Enter MongoDB, a leading NoSQL database, heralded for its flexibility, scalability, and performance. This essay explores MongoDB's emergence, its key features, and its importance in modern data management.

MongoDB, founded in 2007 by Dwight Merriman, Eliot Horowitz, and Kevin Ryan, was born out of the need to address the limitations of relational databases in handling unstructured and semi-structured data. Unlike traditional SQL databases, MongoDB employs a document-oriented data model, storing data in flexible JSON-like documents. This schema-less approach allows for dynamic and agile data modeling, accommodating evolving application requirements without sacrificing performance.

One of MongoDB's core features is its scalability. With support for horizontal scaling through sharding, MongoDB can effortlessly distribute data across multiple servers, ensuring high availability and performance even as data volumes grow. This horizontal scaling capability is particularly crucial in today's era of big data, where organizations grapple with massive datasets generated by web applications, IoT devices, social media platforms, and more.

Another key aspect of MongoDB is its rich query language and powerful indexing capabilities. MongoDB's query language, which is based on JavaScript, provides developers with expressive and intuitive ways to retrieve and manipulate data. Furthermore, MongoDB's indexing capabilities enable efficient query execution, speeding up data retrieval and enhancing overall system performance. Combined, these features empower developers to build complex applications that deliver real-time insights and seamless user experiences.

Moreover, MongoDB's flexible data model makes it well-suited for agile development practices like DevOps and continuous integration/continuous deployment (CI/CD). Developers can quickly iterate on application features, adapting to changing business requirements without cumbersome schema migrations or downtime. This agility fosters innovation and accelerates time-to-market, giving organizations a competitive edge in today's fast-paced digital landscape.

MongoDB's importance extends beyond its technical capabilities; it has become a cornerstone of modern data architectures, underpinning a wide array of applications across various industries. From e-commerce platforms to social media networks, from financial services to healthcare, MongoDB powers mission-critical applications that drive business growth and transformation. Its versatility and scalability make it an ideal choice for a broad spectrum of use cases, from content management and real-time analytics to personalization and recommendation engines.

Furthermore, MongoDB's vibrant ecosystem of tools and integrations enhances its appeal to developers and enterprises alike. From robust drivers for popular programming languages like Python, Java, and Node.js to comprehensive monitoring and management tools like MongoDB Atlas, the MongoDB ecosystem provides everything developers need to build, deploy, and manage MongoDB applications with ease.

Additionally, MongoDB's commitment to open source and community collaboration has fostered a thriving ecosystem of developers, contributors, and enthusiasts. The MongoDB community actively contributes to the platform's evolution, sharing best practices, developing plugins and extensions, and providing support and guidance through forums, meetups, and online communities. This collaborative spirit has been instrumental in MongoDB's success and continued innovation.

In conclusion, MongoDB has emerged as a game-changer in the realm of data management, offering unparalleled flexibility, scalability, and performance for modern applications. Its document-oriented data model, horizontal scalability, expressive query language, and vibrant ecosystem make it a preferred choice for developers and enterprises worldwide. As organizations strive to harness the power of data to drive innovation and growth, MongoDB stands as a testament to the transformative potential of NoSQL databases in the digital age.

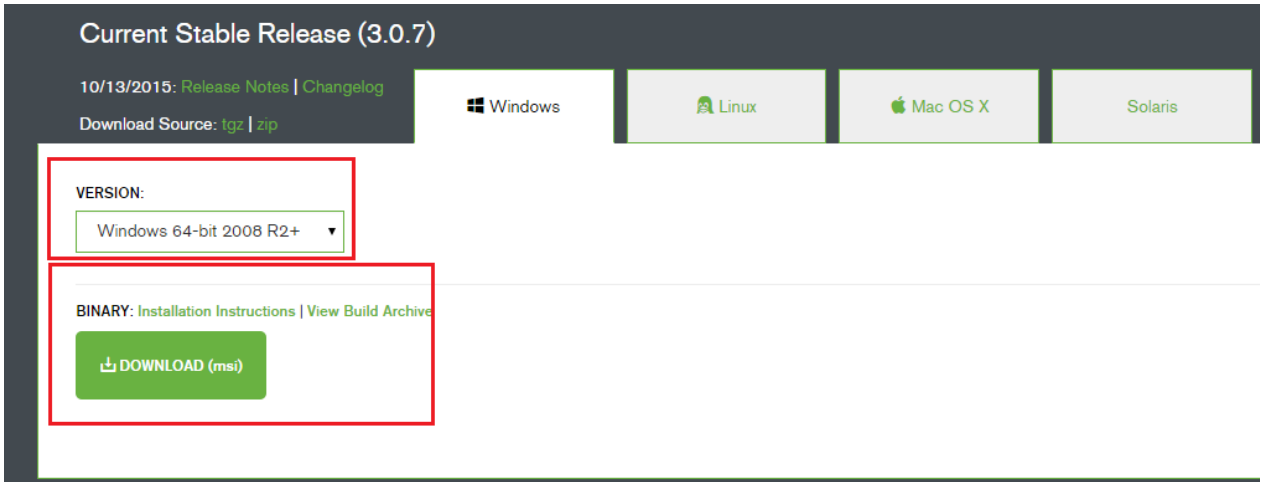
Installation of Mongodb

Mongo DB installation document for Windows OS: Below installation is done on Windows 10:

1. Download msi file from below link:

Register No.................................... Date:..................................

https://www.mongodb.org/downloads#production


Current Stable Release (3.0.7)

10/13/2015: Release Notes | Changelog

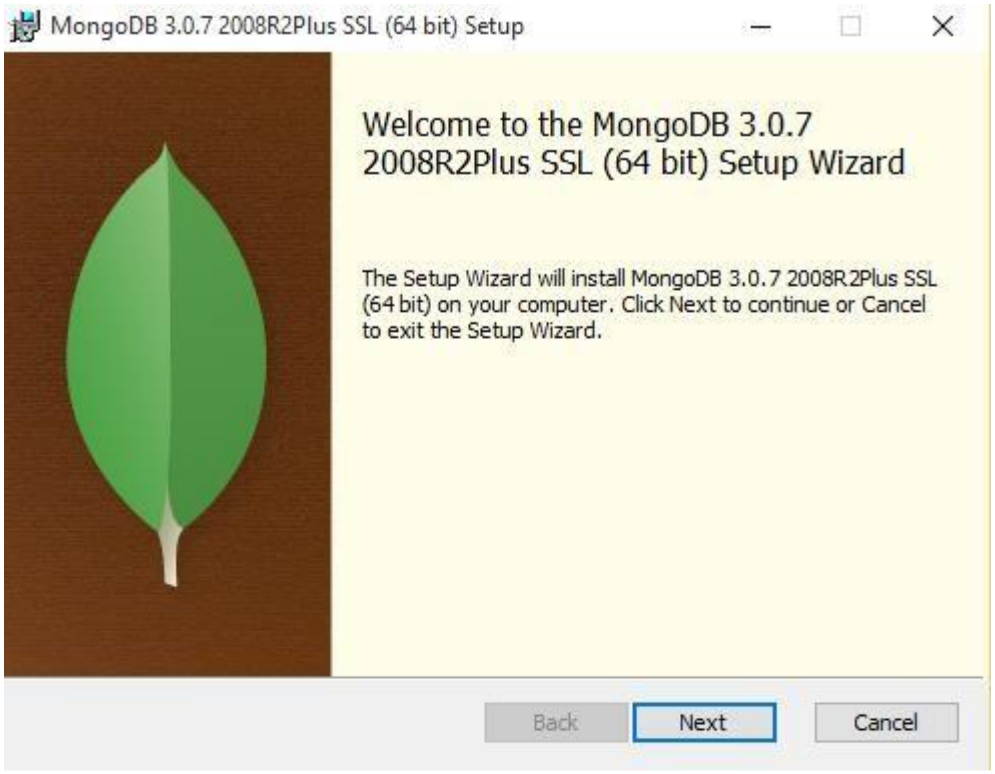
Download Source: tgz | zip

VERSION:

Windows 64-bit 2008 R2+

BINARY: Installation Instructions | View Build Archive

2. Run the msi file


MongoDB 3.0.7 2008R2Plus SSL (64 bit) Setup

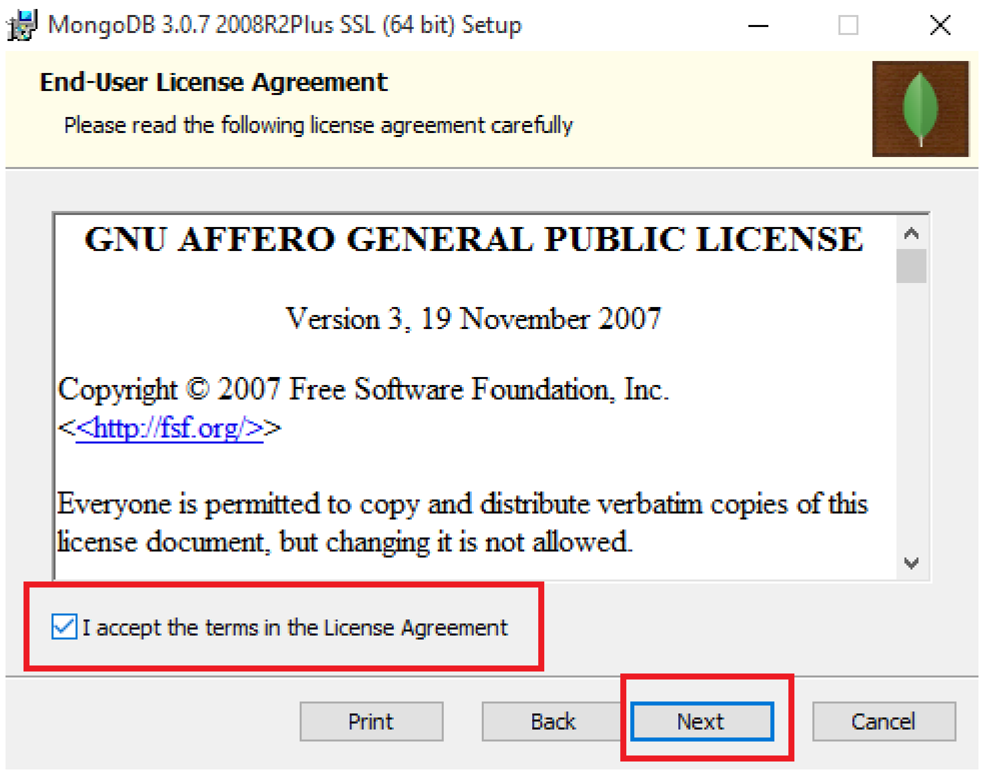
Welcome to the MongoDB 3.0.7 2008R2Plus SSL (64 bit) Setup Wizard

The Setup Wizard will install MongoDB 3.0.7 2008R2Plus SSL (64 bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Click Next

3. Accept the terms in the License Agreement and select Next

Register No.................................. Date:................................


MongoDB 3.0.7 2008R2Plus SSL (64 bit) Setup

End-User License Agreement

Please read the following license agreement carefully

GNU AFFERO GENERAL PUBLIC LICENSE

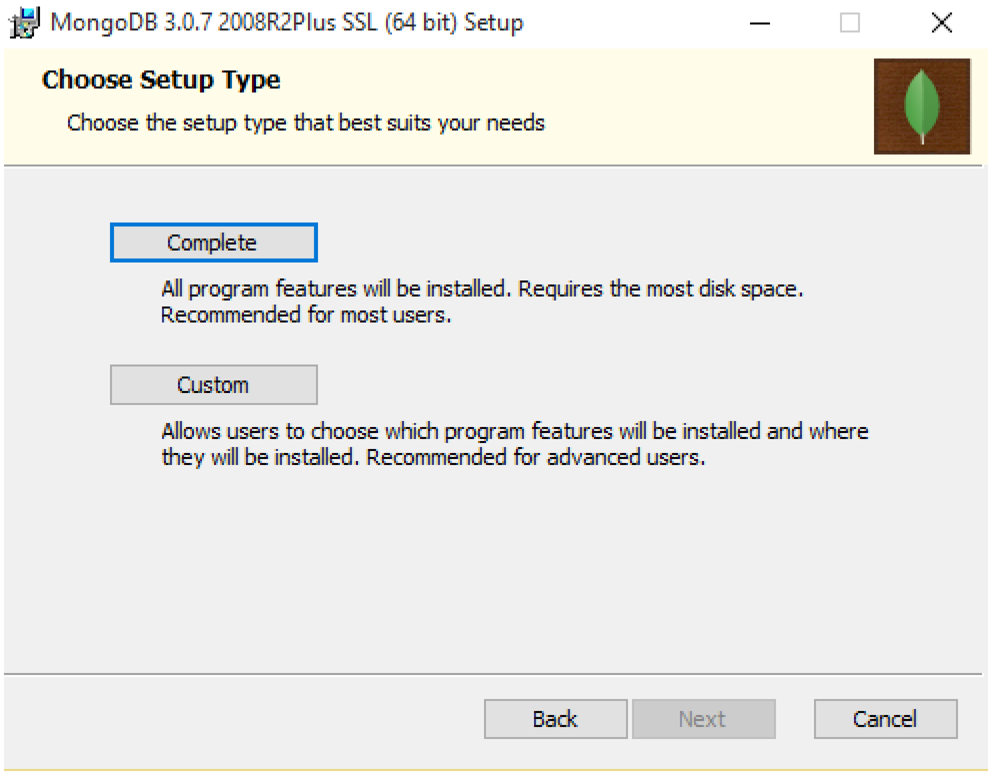
Version 3, 19 November 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

☑ I accept the terms in the License Agreement

4. Select Complete


MongoDB 3.0.7 2008R2Plus SSL (64 bit) Setup

Choose Setup Type

Choose the setup type that best suits your needs

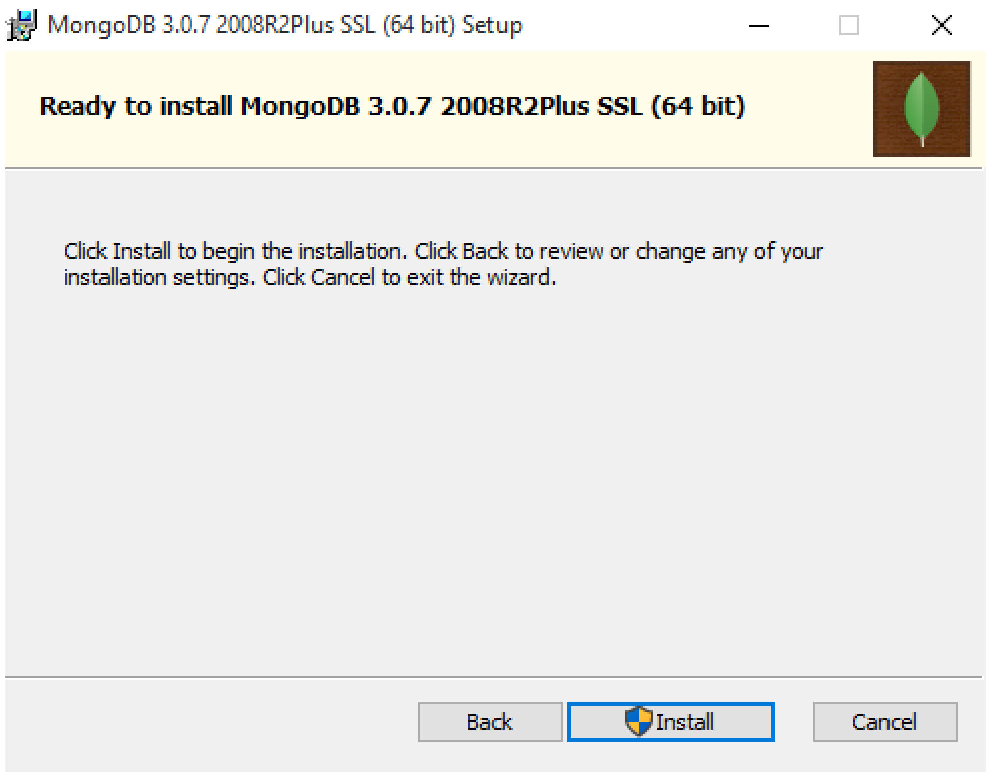
Complete

All program features will be installed. Requires the most disk space. Recommended for most users.

Custom

Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.

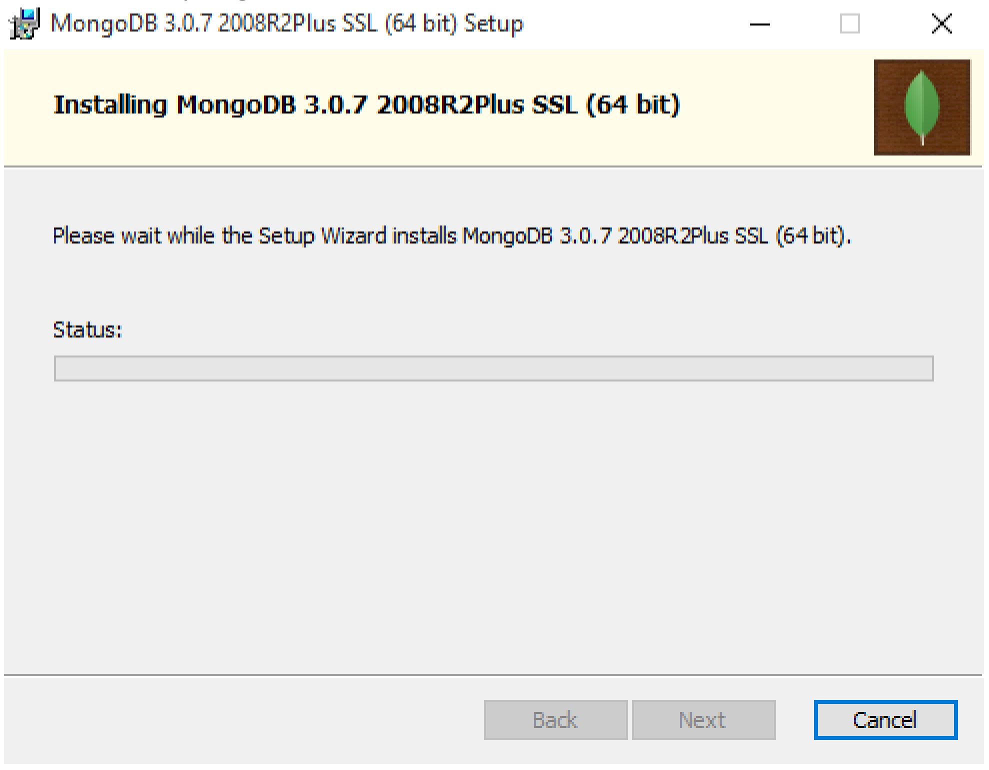
5. Select Install


MongoDB 3.0.7 2008R2Plus SSL (64 bit) Setup

Ready to install MongoDB 3.0.7 2008R2Plus SSL (64 bit)

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

6. Wait until you get the finish screen

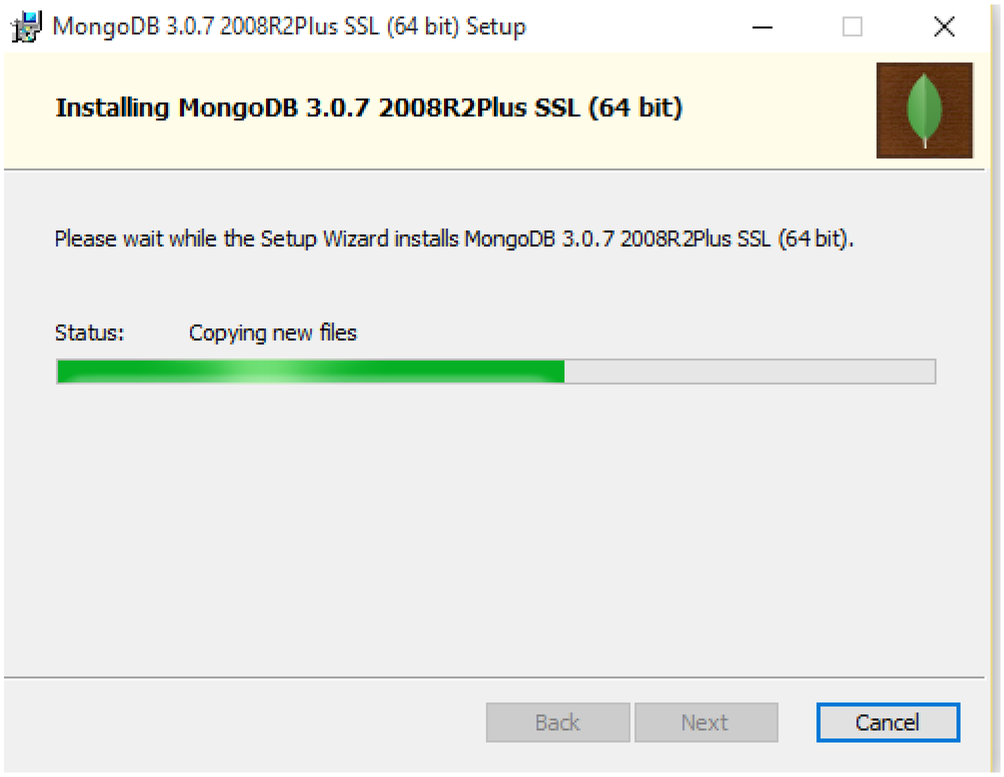

MongoDB 3.0.7 2008R2Plus SSL (64 bit) Setup

Installing MongoDB 3.0.7 2008R2Plus SSL (64 bit)

Please wait while the Setup Wizard installs MongoDB 3.0.7 2008R2Plus SSL (64 bit).

Status:

Register No................................. Date:...............................



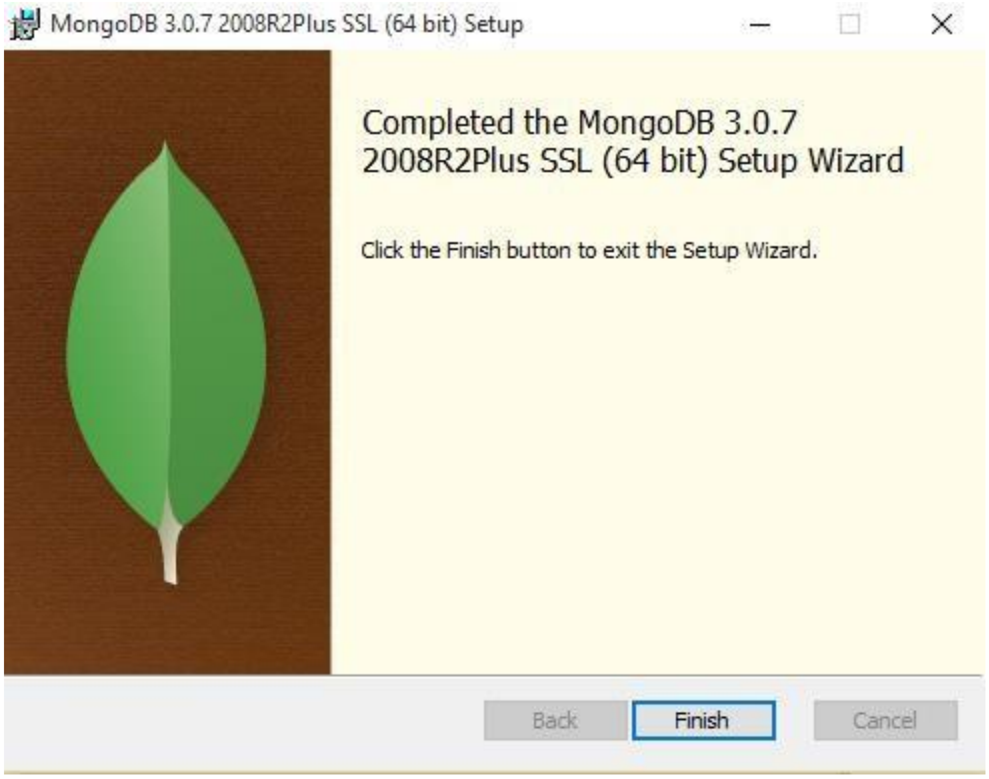
MongoDB 3.0.7 2008R2Plus SSL (64 bit) Setup

Installing MongoDB 3.0.7 2008R2Plus SSL (64 bit)

Please wait while the Setup Wizard installs MongoDB 3.0.7 2008R2Plus SSL (64 bit).

Status: Copying new files

7. Click Finish



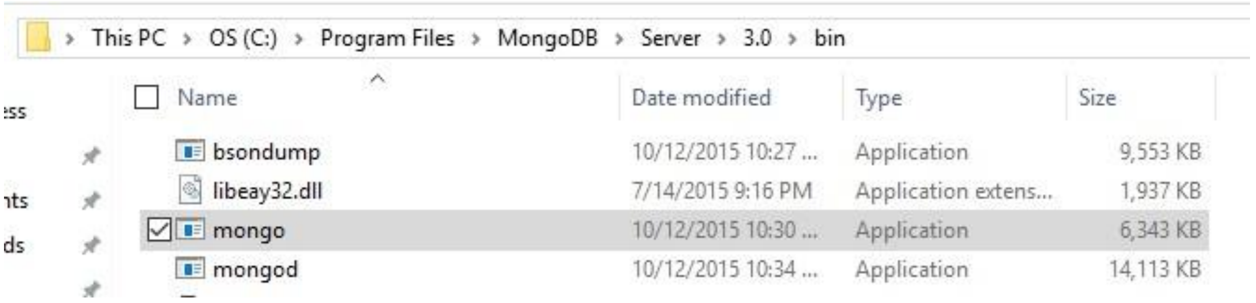
MongoDB 3.0.7 2008R2Plus SSL (64 bit) Setup

Completed the MongoDB 3.0.7 2008R2Plus SSL (64 bit) Setup Wizard

Click the Finish button to exit the Setup Wizard.

Register No................................ Date:..............................

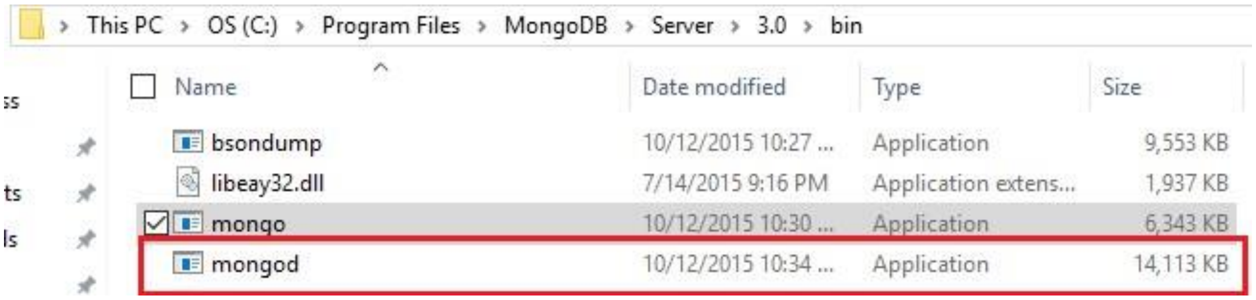
8. Create folders in C drive: data>db
9. Add below path in the environment variable
10. Navigate to C:> ProgramFiles>MongoDB>Server>3.0>bin



This PC > OS (C:) > Program Files > MongoDB > Server > 3.0 > bin

Name	Date modified	Type	Size
bsondump	10/12/2015 10:27 ...	Application	9,553 KB
libeay32.dll	7/14/2015 9:16 PM	Application extens...	1,937 KB
mongo	10/12/2015 10:30 ...	Application	6,343 KB
mongod	10/12/2015 10:34 ...	Application	14,113 KB

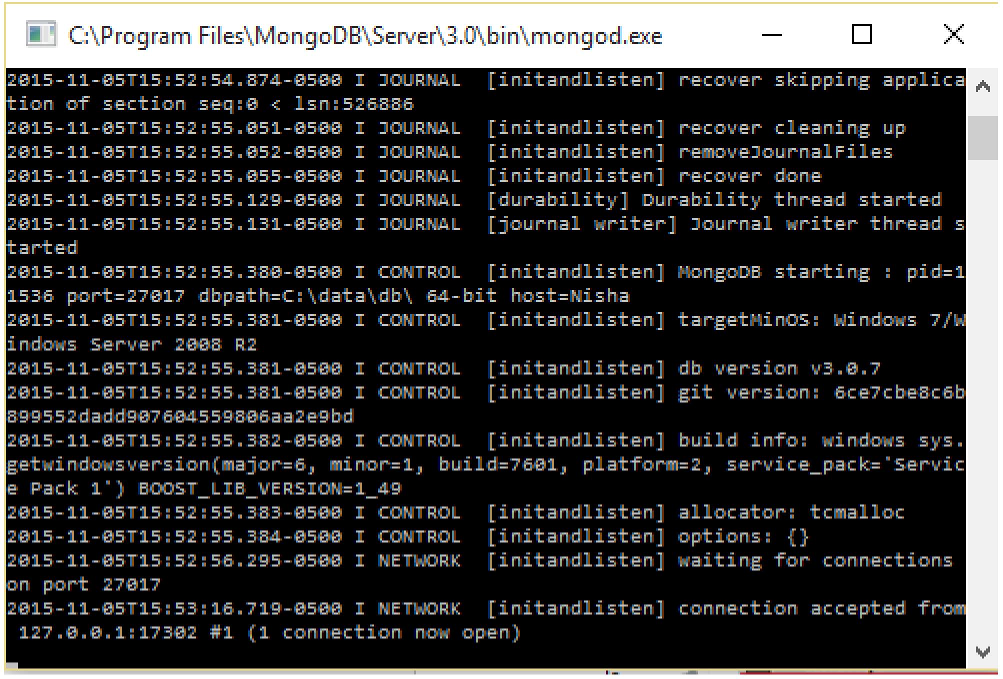
11. Right click and run as administrator: Start the mongo db server



This PC > OS (C:) > Program Files > MongoDB > Server > 3.0 > bin

Name	Date modified	Type	Size
bsondump	10/12/2015 10:27 ...	Application	9,553 KB
libeay32.dll	7/14/2015 9:16 PM	Application extens...	1,937 KB
mongo	10/12/2015 10:30 ...	Application	6,343 KB
mongod	10/12/2015 10:34 ...	Application	14,113 KB

Once you start the server you will see a command prompt saying (1 connection now open)

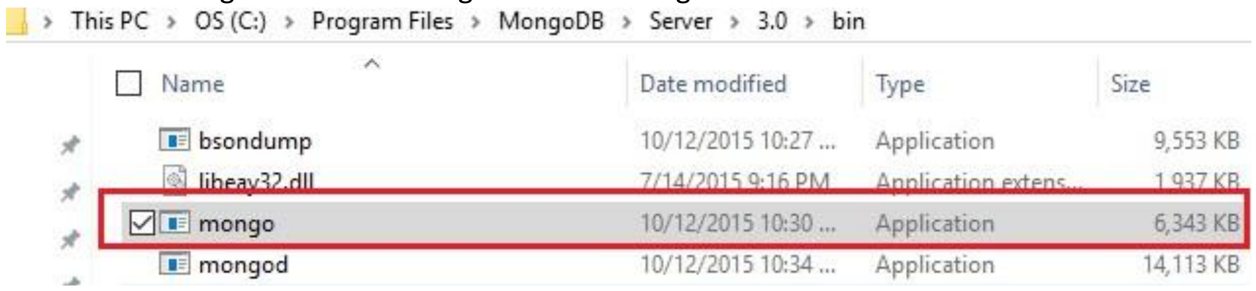


C:\Program Files\MongoDB\Server\3.0\bin\mongod.exe

```
2015-11-05T15:52:54.874-0500 I JOURNAL  [initandlisten] recover skipping applica
tion of section seq:0 < lsn:526886
2015-11-05T15:52:55.051-0500 I JOURNAL  [initandlisten] recover cleaning up
2015-11-05T15:52:55.052-0500 I JOURNAL  [initandlisten] removeJournalFiles
2015-11-05T15:52:55.055-0500 I JOURNAL  [initandlisten] recover done
2015-11-05T15:52:55.129-0500 I JOURNAL  [durability] Durability thread started
2015-11-05T15:52:55.131-0500 I JOURNAL  [journal writer] Journal writer thread s
tarted
2015-11-05T15:52:55.380-0500 I CONTROL  [initandlisten] MongoDB starting : pid=1
1536 port=27017 dbpath=C:\data\db\ 64-bit host=Nisha
2015-11-05T15:52:55.381-0500 I CONTROL  [initandlisten] targetMinOS: Windows 7/W
indows Server 2008 R2
2015-11-05T15:52:55.381-0500 I CONTROL  [initandlisten] db version v3.0.7
2015-11-05T15:52:55.381-0500 I CONTROL  [initandlisten] git version: 6ce7cbe8c6b
899552dadd907604559806aa2e9bd
2015-11-05T15:52:55.382-0500 I CONTROL  [initandlisten] build info: windows sys.
getwindowsversion(major=6, minor=1, build=7601, platform=2, service_pack='Servic
e Pack 1') BOOST_LIB_VERSION=1_49
2015-11-05T15:52:55.383-0500 I CONTROL  [initandlisten] allocator: tcmalloc
2015-11-05T15:52:55.384-0500 I CONTROL  [initandlisten] options: {}
2015-11-05T15:52:56.295-0500 I NETWORK  [initandlisten] waiting for connections
on port 27017
2015-11-05T15:53:16.719-0500 I NETWORK  [initandlisten] connection accepted from
 127.0.0.1:17302 #1 (1 connection now open)
```

Register No.................................. Date:.................................

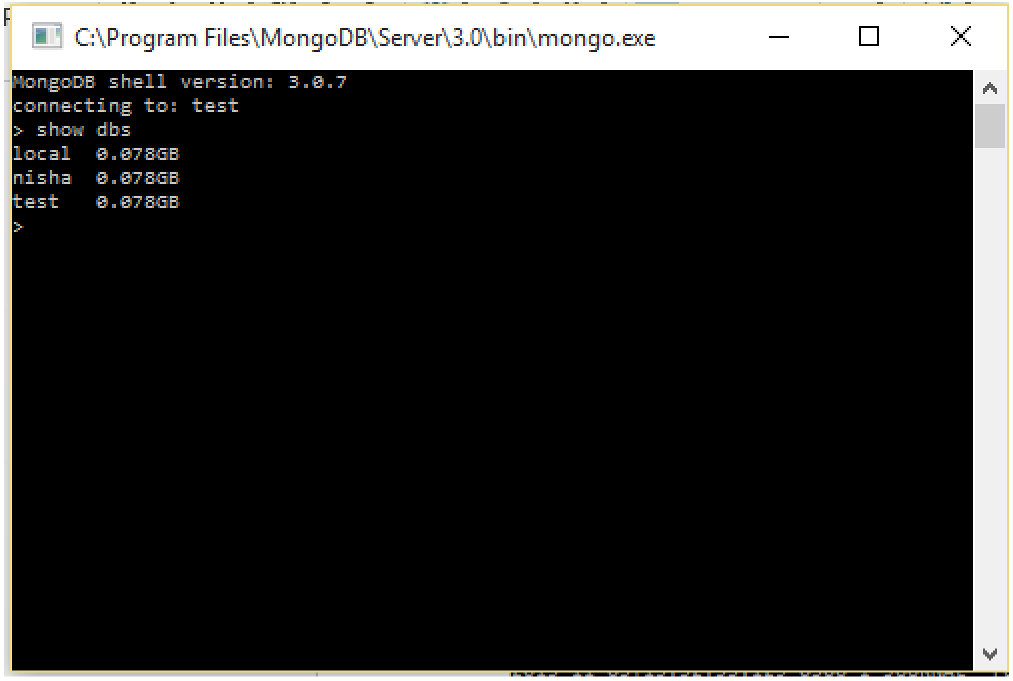
12. Start the mongo db database: Right Click on Mongo and run it as administration



This PC › OS (C:) › Program Files › MongoDB › Server › 3.0 › bin

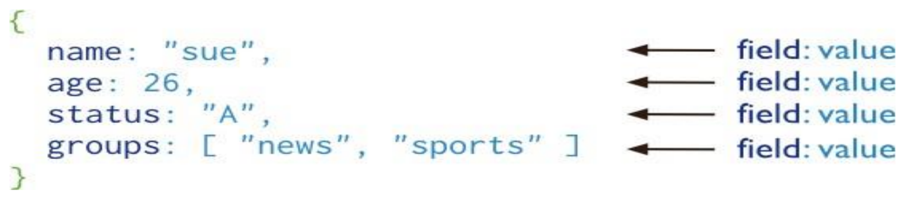
Name	Date modified	Type	Size
bsondump	10/12/2015 10:27 ...	Application	9,553 KB
libeay32.dll	7/14/2015 9:16 PM	Application extens...	1,937 KB
mongo	10/12/2015 10:30 ...	Application	6,343 KB
mongod	10/12/2015 10:34 ...	Application	14,113 KB

Once your database is started , it will be connected to the test database by default Use command: show dbs → it will show you all dbs



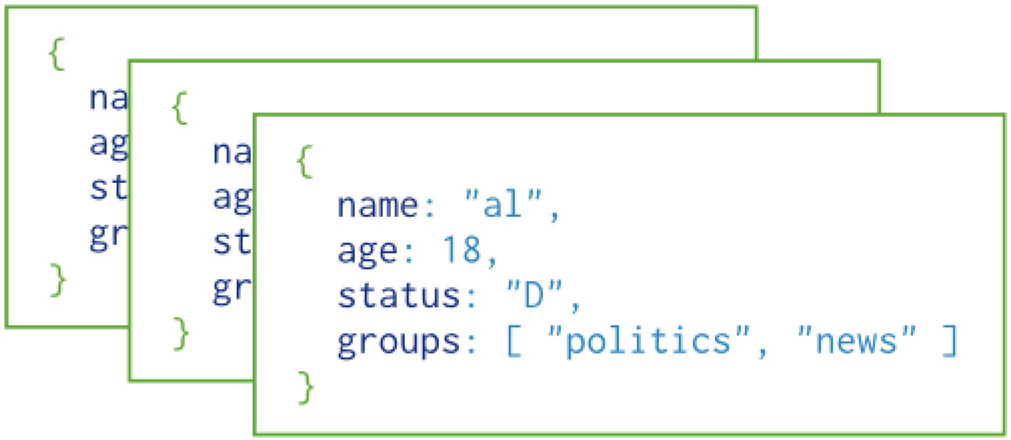
```
MongoDB shell version: 3.0.7
connecting to: test
> show dbs
local   0.078GB
nisha   0.078GB
test    0.078GB
>
```

MongoDB stores data in the form of Documents, which are JSON-like field –value pairs. Mongo DB Documents are called BSON documents: (Binary representation of JSON) with additional type information.



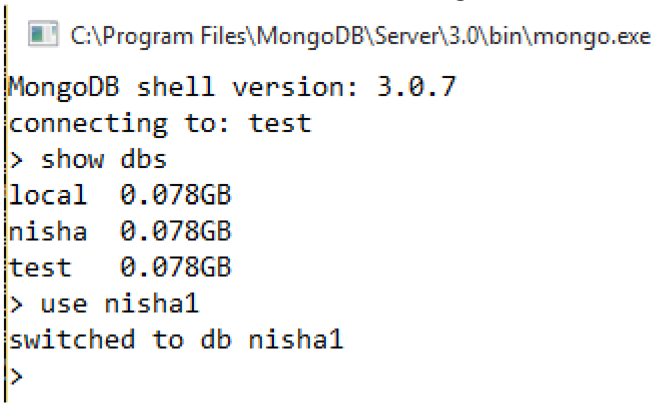
```
{
   name: "sue",                          ← field: value
   age: 26,                              ← field: value
   status: "A",                          ← field: value
   groups: [ "news", "sports" ]          ← field: value
}
```

MongoDB stores all documents in collections. A collection is a group of related documents that have a set of shared common indexes. Collections are analogous to a table in relational databases.



Collection

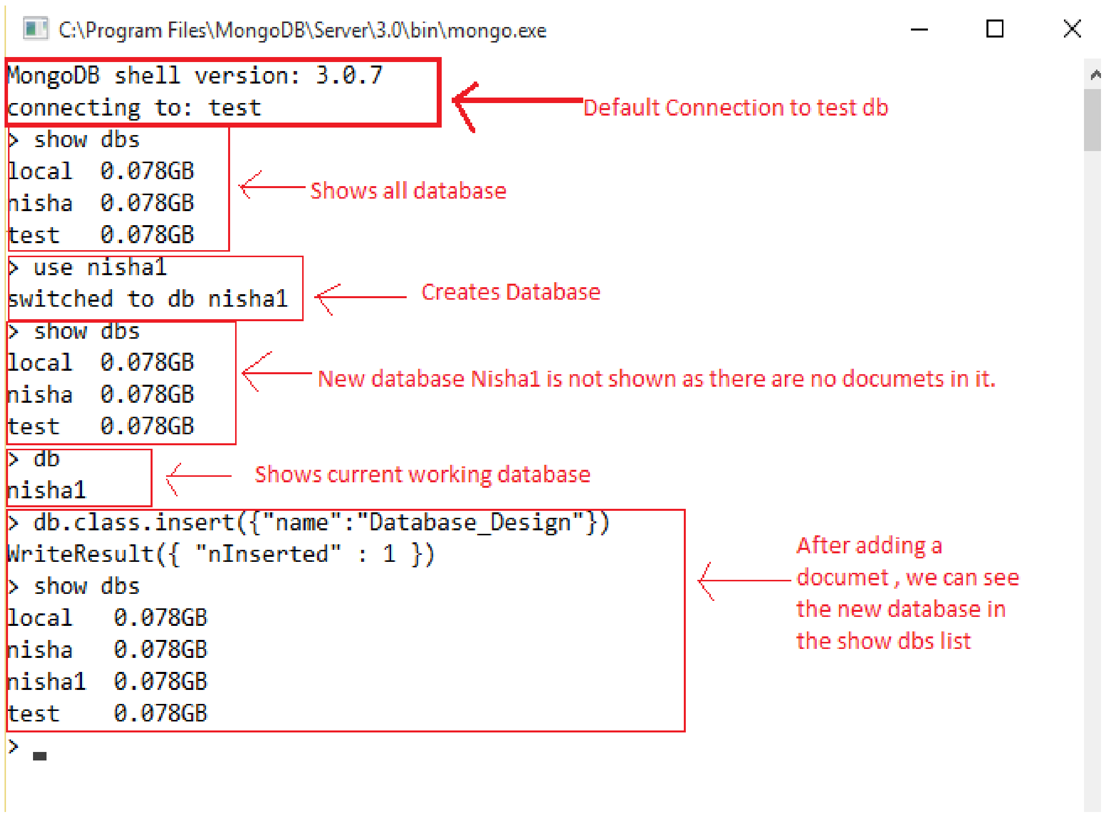
Use dbname ➔ created database e.g. use nisha1



```
C:\Program Files\MongoDB\Server\3.0\bin\mongo.exe
MongoDB shell version: 3.0.7
connecting to: test
> show dbs
local   0.078GB
nisha   0.078GB
test    0.078GB
> use nisha1
switched to db nisha1
>
```

Good resource for learning Mongo DB:
http://www.tutorialspoint.com/mongodb/mongodb_create_collection.htm

Some basic commands to try:



```
MongoDB shell version: 3.0.7
connecting to: test
> show dbs
local  0.078GB
nisha  0.078GB
test   0.078GB
> use nisha1
switched to db nisha1
> show dbs
local  0.078GB
nisha  0.078GB
test   0.078GB
> db
nisha1
> db.class.insert({"name":"Database_Design"})
WriteResult({ "nInserted" : 1 })
> show dbs
local   0.078GB
nisha   0.078GB
nisha1  0.078GB
test    0.078GB
>
```

Default Connection to test db

Shows all database

Creates Database

New database Nisha1 is not shown as there are no documets in it.

Shows current working database

After adding a documet , we can see the new database in the show dbs list

References: https://docs.mongodb.org/manual/

Register No................... Date:.................................

Exercise-2

Description of mongo Shell, Create database and show database

Navigating MongoDB: Unveiling the MongoDB Shell's Power in Creating and Managing Databases

In the realm of modern data management, MongoDB stands out as a frontrunner, offering unparalleled flexibility and scalability. At the heart of MongoDB's ecosystem lies the MongoDB Shell, a command-line interface (CLI) that empowers developers and administrators to interact with MongoDB instances seamlessly. This essay embarks on an exploration of the MongoDB Shell, elucidating its functionalities, and demonstrating the process of creating and managing databases with illustrative examples.

Understanding the MongoDB Shell:

The MongoDB Shell, often referred to as "mongo," serves as a robust tool for interfacing with MongoDB databases directly from the command line. Powered by JavaScript, the MongoDB Shell provides a familiar environment for executing database operations, administering collections, and querying data effortlessly. It is an indispensable component of MongoDB's toolkit, offering both simplicity and power to users.

Creating Databases with MongoDB Shell:

MongoDB's schema-less architecture allows for dynamic database creation, enabling developers to effortlessly set up databases on-the-fly. The MongoDB Shell simplifies this process, providing a straightforward mechanism for creating databases. Let's delve into a practical example:

```
// Start MongoDB Shell

mongo


// Create a new database named "mydatabase"

use mydatabase
```

In the above example, we initiate the MongoDB Shell by running the `mongo` command in the terminal. Once inside the shell, the `use` command is employed to switch to the desired database or create it if it does not already exist. Upon execution, MongoDB switches to the "mydatabase" database, creating it if it's not present. This lazy creation approach ensures database creation only upon necessity, conserving resources efficiently.

Exploring Databases:

After creating a database, users often need to navigate and explore existing databases. The MongoDB Shell provides commands to facilitate this exploration, allowing users to view available databases effortlessly. Let's illustrate this with an example:

```
// Display a list of all databases

show databases
```

Executing the `show databases` command within the MongoDB Shell yields a comprehensive list of databases available on the MongoDB server. The output includes system databases such as "admin" and "local," along with any user-defined databases like "mydatabase." This command provides users with a quick overview of the databases present, aiding in navigation and administration tasks.

Managing Collections:

Within MongoDB databases, data is organized into collections, analogous to tables in relational databases. The MongoDB Shell enables users to manage collections efficiently, facilitating tasks such as viewing existing collections and creating new ones. Let's illustrate this with an example:

```
// Display a list of collections in the current database

show collections
```

Executing the `show collections` command within a database reveals a list of all collections present within that database. Collections play a pivotal role in MongoDB's data model, serving as containers for JSON-like documents. This command empowers users to quickly ascertain the structure of their database, aiding in data exploration and analysis.

Conclusion:

In conclusion, the MongoDB Shell serves as a cornerstone of MongoDB's ecosystem, providing developers and administrators with a powerful interface for managing databases effortlessly. Through simple commands like `use`, `show databases`, and `show collections`, users can create databases, navigate existing ones, and administer collections seamlessly. With its intuitive JavaScript-based interface, the MongoDB Shell democratizes database management, empowering users of all levels to harness the full potential of MongoDB's flexible and scalable architecture.

By bridging the gap between developers and databases, the MongoDB Shell fosters collaboration, innovation, and efficiency, driving forward the evolution of modern data management practices. As organizations continue to embrace MongoDB as their preferred database solution, the MongoDB Shell will remain an indispensable tool in their arsenal, enabling them to navigate the complexities of data management with confidence and ease.

Register No.....................   Date:.................................

Exercise-3

Commands for MongoDB and to study operations in MongoDB – Insert, Query, Update, Delete and Projection

MongoDB Commands: Exploring Operations

MongoDB, a leading NoSQL database, offers a rich set of commands for performing various operations, from basic CRUD (Create, Read, Update, Delete) tasks to more advanced operations like Projection. In this essay, we delve into five fundamental MongoDB commands, namely Insert, Query, Update, Delete, and Projection, exploring their functionalities and providing illustrative examples with output.

1. Insert:

The `insert` command in MongoDB is used to insert documents into a collection. Documents in MongoDB are JSON-like objects that store data in key-value pairs. Let's explore the `insert` command with an example:

```
// Insert a document into the "users" collection

db.users.insert({

  name: "John Doe",

  age: 30,

  email: "john@example.com"

})
```

In the above example, we insert a document representing a user into the "users" collection. The document contains fields such as "name," "age," and "email," each with corresponding values. Upon execution, MongoDB adds this document to the "users" collection.

Output:

```
WriteResult({ "nInserted" : 1 })
```

The output indicates that one document was successfully inserted into the collection.

2. Query:

The `find` command in MongoDB is used to query documents from a collection based on specified criteria. It allows users to retrieve data that matches certain conditions. Let's explore the `find` command with an example:

```
// Query documents where age is greater than 25

db.users.find({ age: { $gt: 25 } })
```

In the above example, we query the "users" collection to retrieve documents where the "age" field is greater than 25. MongoDB's query language supports various operators like `$gt` (greater than), `$lt` (less than), `$eq` (equal to), etc., enabling users to craft complex queries.

Output:

```
{ "_id" : ObjectId("60b7d56da8e1d54b08a789cf"), "name" : "John Doe", "age" : 30, "email" :
"john@example.com" }
```

The output displays the document(s) that match the specified criteria.

3. Update:

The `update` command in MongoDB is used to modify existing documents within a collection. It allows users to update specific fields or replace entire documents. Let's explore the `update` command with an example:

```
// Update the email of a user with a specific name

db.users.update(

  { name: "John Doe" },

  { $set: { email: "john.doe@example.com" } }

)
```

In the above example, we update the email address of the user with the name "John Doe" to "john.doe@example.com." MongoDB's update command supports various modifiers like `$set`, `$unset`, `$inc`, etc., enabling precise updates to documents.

Output:

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

The output indicates that one document was matched and modified as a result of the update operation.

4. Delete:

The `delete` command in MongoDB is used to remove documents from a collection that match specified criteria. It allows users to delete one or multiple documents based on the provided query. Let's explore the `delete` command with an example:

```
// Delete documents where age is less than 25
db.users.deleteMany({ age: { $lt: 25 } })
```

In the above example, we delete documents from the "users" collection where the "age" field is less than 25. MongoDB's `deleteMany` command removes all documents that match the specified criteria.

Output:

```
{ "acknowledged" : true, "deletedCount" : 0 }
```

The output indicates that no documents were deleted as none matched the specified criteria.

5. Projection:

The `find` command in MongoDB also supports Projection, allowing users to retrieve only specific fields from documents rather than the entire document. It helps in optimizing query performance and reducing network overhead. Let's explore Projection with an example:

```
// Retrieve only the name and email fields of users
db.users.find({}, { name: 1, email: 1, _id: 0 })
```

In the above example, we query the "users" collection to retrieve only the "name" and "email" fields of documents while excluding the "_id" field. This reduces the amount of data transferred over the network and improves query performance.

Output:

```
{ "name" : "John Doe", "email" : "john.doe@example.com" }
```

The output displays the specified fields of the document(s) retrieved from the collection.

Conclusion:

In conclusion, MongoDB offers a versatile set of commands for performing various operations on databases and collections. From inserting and querying data to updating, deleting, and projecting fields, MongoDB commands provide users with the flexibility and power to manipulate data effectively. Through illustrative examples and output, this essay elucidates the functionalities of MongoDB commands, empowering users to leverage MongoDB's capabilities for efficient data management and manipulation.

Register No.................... Date:.................................

Exercise-4

Download a zip code dataset at http://media.mongodb.org/zips.json .Use mongo import to import the zip code dataset into MongoDB. After importing the data, answer the following questions by using aggregation pipelines:

1. Find all the states that have a city called "BOSTON". Find all the states and cities whose names include the string "BOST".
2. Each city has several zip codes. Find the city in each state with the greatest number of zip codes and rank those cities along with the states using the city populations. MongoDB can query on spatial information

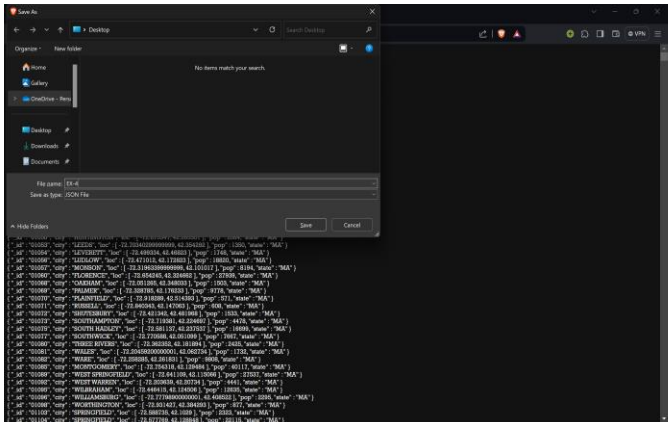
Aim: Download a zip code dataset at http://media.mongodb.org/zips.json. Use mongo import to import the zip code dataset into MongoDB. After importing the data, answer the following questions by using aggregation pipelines:

1. Find all the states that have a city called "BOSTON". Find all the states and cities whose names include the string "BOST".
2. *Each city has several zip codes. Find the city in each state with the greatest number of zip codes and rank those cities along with the states using the city populations. MongoDB can query on spatial information.*

PROCEDURE

STEPS

1. *Downloading the Dataset.*
   Click on the link below.
   [*http://media.mongodb.org/zips.json*](http://media.mongodb.org/zips.json)



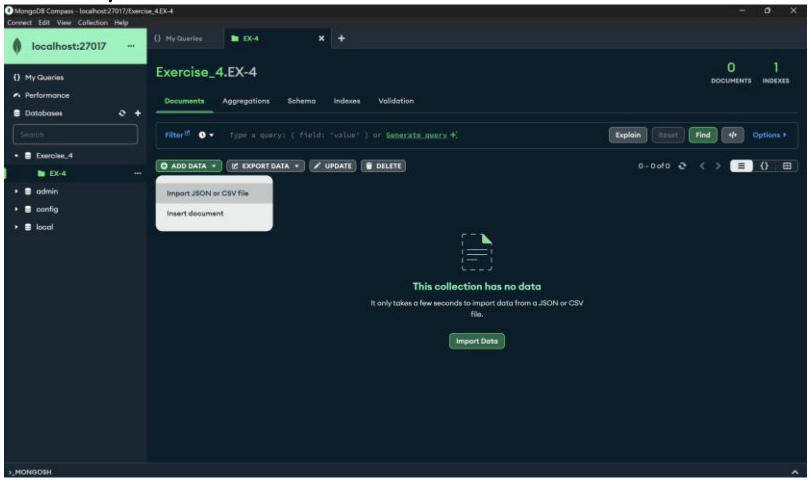
File name: EX-4
Save as type: JSON File

```
{ "_id" : "01053", "city" : "LEEDS", "loc" : [ -72.70340299999999, 42.354282 ], "pop" : 1350, "state" : "MA" }
{ "_id" : "01054", "city" : "LEVERETT", "loc" : [ -72.499334, 42.46823 ], "pop" : 1748, "state" : "MA" }
{ "_id" : "01056", "city" : "LUDLOW", "loc" : [ -72.471012, 42.172823 ], "pop" : 18820, "state" : "MA" }
{ "_id" : "01057", "city" : "MONSON", "loc" : [ -72.31963399999999, 42.101017 ], "pop" : 8194, "state" : "MA" }
{ "_id" : "01060", "city" : "FLORENCE", "loc" : [ -72.654245, 42.324662 ], "pop" : 27939, "state" : "MA" }
{ "_id" : "01068", "city" : "OAKHAM", "loc" : [ -72.051265, 42.348033 ], "pop" : 1503, "state" : "MA" }
{ "_id" : "01069", "city" : "PALMER", "loc" : [ -72.328785, 42.176233 ], "pop" : 9778, "state" : "MA" }
{ "_id" : "01070", "city" : "PLAINFIELD", "loc" : [ -72.918289, 42.514393 ], "pop" : 571, "state" : "MA" }
{ "_id" : "01071", "city" : "RUSSELL", "loc" : [ -72.840343, 42.147063 ], "pop" : 608, "state" : "MA" }
{ "_id" : "01072", "city" : "SHUTESBURY", "loc" : [ -72.421342, 42.481968 ], "pop" : 1533, "state" : "MA" }
{ "_id" : "01073", "city" : "SOUTHAMPTON", "loc" : [ -72.719381, 42.224697 ], "pop" : 4478, "state" : "MA" }
{ "_id" : "01075", "city" : "SOUTH HADLEY", "loc" : [ -72.581137, 42.237537 ], "pop" : 16699, "state" : "MA" }
{ "_id" : "01077", "city" : "SOUTHWICK", "loc" : [ -72.770588, 42.051099 ], "pop" : 7667, "state" : "MA" }
{ "_id" : "01080", "city" : "THREE RIVERS", "loc" : [ -72.362352, 42.181894 ], "pop" : 2425, "state" : "MA" }
{ "_id" : "01081", "city" : "WALES", "loc" : [ -72.20458200000001, 42.062734 ], "pop" : 1732, "state" : "MA" }
{ "_id" : "01082", "city" : "WARE", "loc" : [ -72.258285, 42.261831 ], "pop" : 9808, "state" : "MA" }
{ "_id" : "01085", "city" : "MONTGOMERY", "loc" : [ -72.754318, 42.129484 ], "pop" : 40117, "state" : "MA" }
{ "_id" : "01089", "city" : "WEST SPRINGFIELD", "loc" : [ -72.641109, 42.115066 ], "pop" : 27537, "state" : "MA" }
{ "_id" : "01092", "city" : "WEST WARREN", "loc" : [ -72.203639, 42.20734 ], "pop" : 4441, "state" : "MA" }
{ "_id" : "01095", "city" : "WILBRAHAM", "loc" : [ -72.446415, 42.124506 ], "pop" : 12635, "state" : "MA" }
{ "_id" : "01096", "city" : "WILLIAMSBURG", "loc" : [ -72.77798900000001, 42.408522 ], "pop" : 2295, "state" : "MA" }
{ "_id" : "01098", "city" : "WORTHINGTON", "loc" : [ -72.931427, 42.384293 ], "pop" : 877, "state" : "MA" }
{ "_id" : "01103", "city" : "SPRINGFIELD", "loc" : [ -72.588735, 42.1029 ], "pop" : 2323, "state" : "MA" }
{ "_id" : "01104", "city" : "SPRINGFIELD", "loc" : [ -72.577769, 42.128848 ], "pop" : 22115, "state" : "MA" }
```

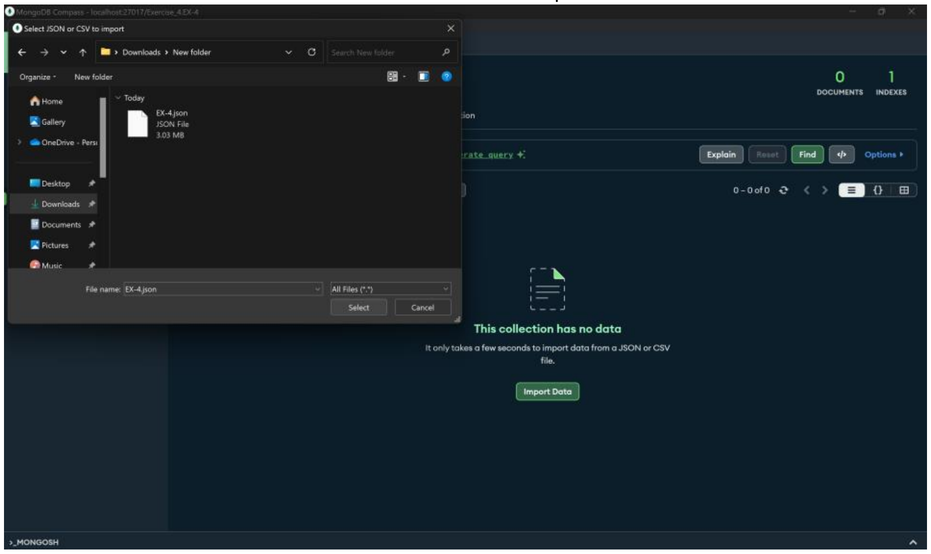
Press '*Ctrl + S*' to save the file.
File Name: EX-4
File Type: JSON File

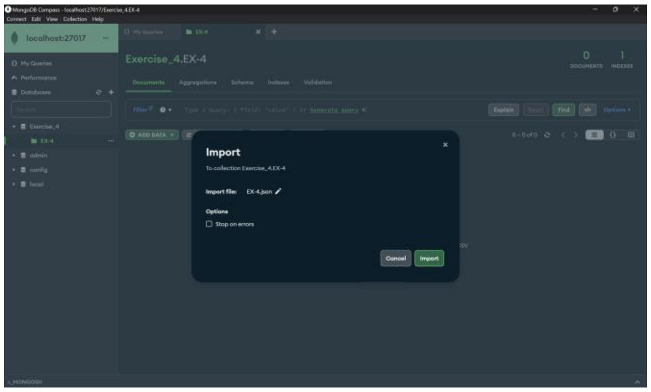
2. Import the Dataset on to MongoDB Compass

i. Connect to MongoDB Compass
ii. Create database. For example, LAB.
iii. Create a collection. For example, EX4.
iv. Click on ***ADD DATA.*** Select ***Import JSON or SCV File.***



v. Chose the download dataset. And click **Select.** And then Import.





vi. Open MongoDB shell or ***_MONGOSH.***

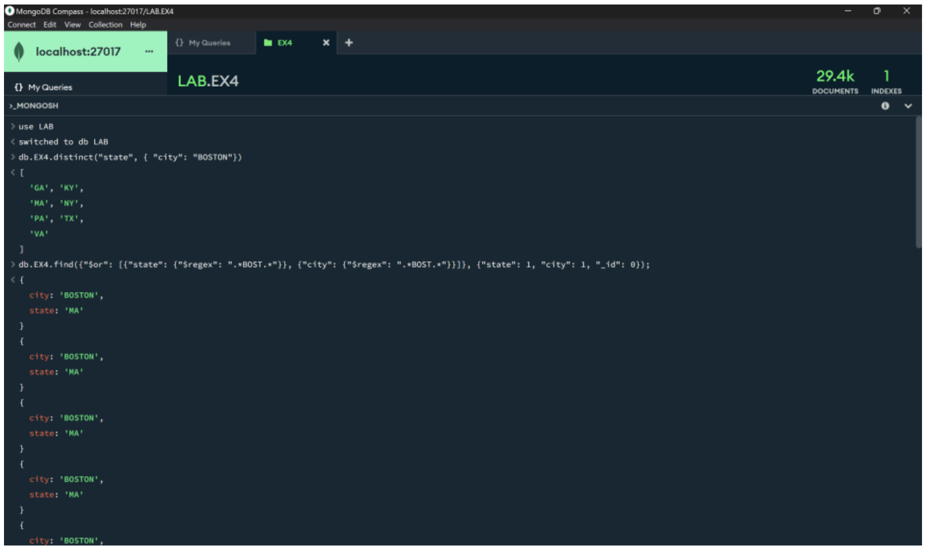
3. Queries

a) Commands

test> use LAB

LAB> db.EX4.distinct("state", { "city": "BOSTON"})

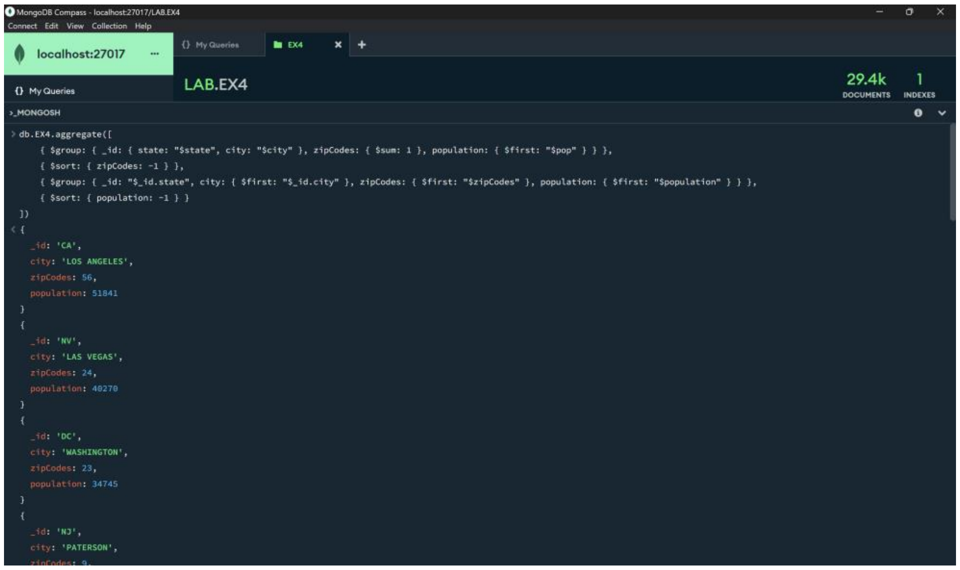
LAB> db.EX4.find({"$or": [{"state": {"$regex": ".*BOST.*"}}, {"city": {"$regex": ".*BOST.*"}}]}, {"state": 1, "city": 1, "_id": 0});


MongoDB Compass - localhost:27017/LAB.EX4
Connect Edit View Collection Help
localhost:27017
My Queries
EX4
LAB.EX4
29.4k DOCUMENTS
1 INDEXES
>_MONGOSH

```
> use LAB
< switched to db LAB
> db.EX4.distinct("state", { "city": "BOSTON"})
< [
    'GA', 'KY',
    'MA', 'NY',
    'PA', 'TX',
    'VA'
  ]
> db.EX4.find({"$or": [{"state": {"$regex": ".*BOST.*"}}, {"city": {"$regex": ".*BOST.*"}}]}, {"state": 1, "city": 1, "_id": 0});
< {
    city: 'BOSTON',
    state: 'MA'
  }
  {
    city: 'BOSTON',
    state: 'MA'
  }
  {
    city: 'BOSTON',
    state: 'MA'
  }
  {
    city: 'BOSTON',
    state: 'MA'
  }
  {
    city: 'BOSTON',
```

b) Commands

```
LAB> cls
LAB> db.EX4.aggregate([
   { $group: { _id: { state: "$state", city: "$city" }, zipCodes: { $sum: 1 }, pop: { $first: "$pop" } } },
   { $sort: { zipCodes:-1 } },
   { $group: { _id: "$_id.state", city: { $first: "$_id.city" }, zipCodes: { $first: "$zipCodes" }, pop: { $first:
"$pop" } } },
   { $sort: { pop:-1 } }
])
```


MongoDB Compass - localhost:27017/LAB.EX4
Connect Edit View Collection Help
localhost:27017
{} My Queries
EX4
LAB.EX4
29.4k DOCUMENTS
1 INDEXES
>_MONGOSH

```
> db.EX4.aggregate([
      { $group: { _id: { state: "$state", city: "$city" }, zipCodes: { $sum: 1 }, population: { $first: "$pop" } } },
      { $sort: { zipCodes: -1 } },
      { $group: { _id: "$_id.state", city: { $first: "$_id.city" }, zipCodes: { $first: "$zipCodes" }, population: { $first: "$population" } } },
      { $sort: { population: -1 } }
  ])
< {
    _id: 'CA',
    city: 'LOS ANGELES',
    zipCodes: 56,
    population: 51841
  }
  {
    _id: 'NV',
    city: 'LAS VEGAS',
    zipCodes: 24,
    population: 40270
  }
  {
    _id: 'DC',
    city: 'WASHINGTON',
    zipCodes: 23,
    population: 34745
  }
  {
    _id: 'NJ',
    city: 'PATERSON',
    zipCodes: 9,
```

Register No.....................   Date:...................................

Exercise-5

Where Clause equivalent in MongoDB: Write a MongoDB query to find the restaurants that achieved a score, more than 80 but less than 100

AIM:

Where Clause equivalent in MongoDB: Write a MongoDB query to find the restaurants that achieved a score, more than 80 but less than 100.

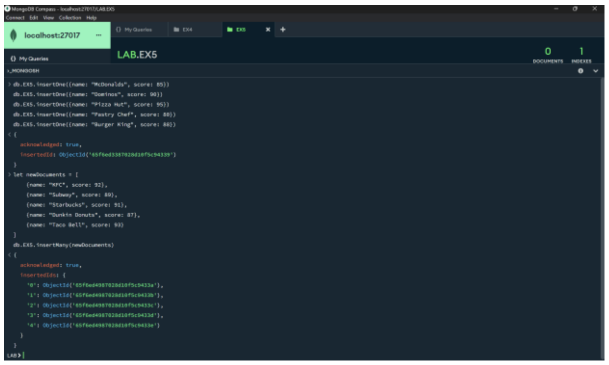
PROCEDURE

STEPS

1. MongoDB Compass
   i. Connect to MongoDB Compass
   ii. Create a collection. For example, EX5.

2. Commands

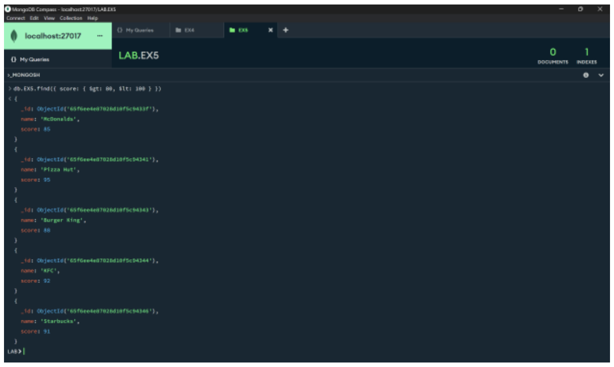
```
LAB>db.EX5.insertOne({name: "McDonalds", score: 85})
db.EX5.insertOne({name: "Dominos", score: 70})
db.EX5.insertOne({name: "Pizza Hut", score: 95})
db.EX5.insertOne({name: "Pastry Chef", score: 70})
db.EX5.insertOne({name: "Burger King", score: 88})
LAB> let newDocuments = [
   {name: "KFC", score: 92},
   {name: "Subway", score: 79},
   {name: "Starbucks", score: 91},
   {name: "Dunkin Donuts", score: 77},
   {name: "Taco Bell", score: 93}
]
db.EX5.insertMany(newDocuments)
```


MongoDB Compass - localhost:27017/LAB.EX5
Connect Edit View Collection Help
localhost:27017
My Queries
EX4
EX5
LAB.EX5
>_MONGOSH

```
> db.EX5.insertOne({name: "McDonalds", score: 85})
  db.EX5.insertOne({name: "Dominos", score: 90})
  db.EX5.insertOne({name: "Pizza Hut", score: 95})
  db.EX5.insertOne({name: "Pastry Chef", score: 80})
  db.EX5.insertOne({name: "Burger King", score: 88})
< {
    acknowledged: true,
    insertedId: ObjectId('65f6ed3387028d10f5c94339')
  }
> let newDocuments = [
      {name: "KFC", score: 92},
      {name: "Subway", score: 89},
      {name: "Starbucks", score: 91},
      {name: "Dunkin Donuts", score: 87},
      {name: "Taco Bell", score: 93}
  ]
  db.EX5.insertMany(newDocuments)
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('65f6ed4987028d10f5c9433a'),
      '1': ObjectId('65f6ed4987028d10f5c9433b'),
      '2': ObjectId('65f6ed4987028d10f5c9433c'),
      '3': ObjectId('65f6ed4987028d10f5c9433d'),
      '4': ObjectId('65f6ed4987028d10f5c9433e')
    }
  }
LAB>
```

```
LAB> db.EX5.find({ score: { $gt: 80, $lt: 100 } })
```

Register No.………………… Date:………………………………



```
> db.EX5.find({ score: { $gt: 80, $lt: 100 } })
< {
    _id: ObjectId('65f6ee4e87028d10f5c9433f'),
    name: 'McDonalds',
    score: 85
  }
  {
    _id: ObjectId('65f6ee4e87028d10f5c94341'),
    name: 'Pizza Hut',
    score: 95
  }
  {
    _id: ObjectId('65f6ee4e87028d10f5c94343'),
    name: 'Burger King',
    score: 88
  }
  {
    _id: ObjectId('65f6ee4e87028d10f5c94344'),
    name: 'KFC',
    score: 92
  }
  {
    _id: ObjectId('65f6ee4e87028d10f5c94346'),
    name: 'Starbucks',
    score: 91
  }
LAB>
```

Register No.................... Date:................................

Exercise-6

Write a MongoDB query to find the restaurants which locate in longitude value less than -95.754168.

AIM

Write a MongoDB query to find the restaurants which locate in longitude value less than -95.754168.

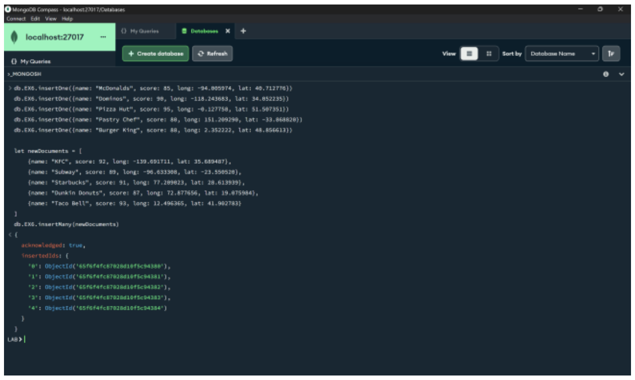
PROCEDURE

STEPS

1. **MongoDB Compass**
   i. Connect to MongoDB Compass
   ii. Create a collection. For example, EX6.

2. **Commands**

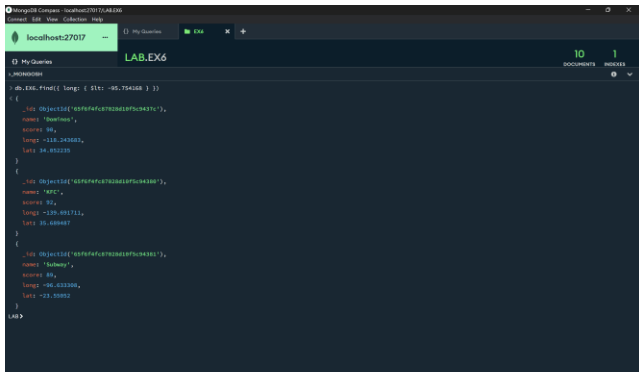
```
LAB>db.EX6.insertOne({name: "McDonalds", score: 85, long: -94.005974, lat: 40.712776})
db.EX6.insertOne({name: "Dominos", score: 90, long: -118.243683, lat: 34.052235})
db.EX6.insertOne({name: "Pizza Hut", score: 95, long: -0.127758, lat: 51.507351})
db.EX6.insertOne({name: "Pastry Chef", score: 80, long: 151.209290, lat: -33.868820})
db.EX6.insertOne({name: "Burger King", score: 88, long: 2.352222, lat: 48.856613})

let newDocuments = [
   {name: "KFC", score: 92, long: -139.691711, lat: 35.689487},
   {name: "Subway", score: 89, long: -96.633308, lat: -23.550520},
   {name: "Starbucks", score: 91, long: 77.209023, lat: 28.613939},
   {name: "Dunkin Donuts", score: 87, long: 72.877656, lat: 19.075984},
   {name: "Taco Bell", score: 93, long: 12.496365, lat: 41.902783}
]
db.EX6.insertMany(newDocuments)
```



```
> db.EX6.insertOne({name: "McDonalds", score: 85, long: -94.005974, lat: 40.712776})
  db.EX6.insertOne({name: "Dominos", score: 90, long: -118.243683, lat: 34.052235})
  db.EX6.insertOne({name: "Pizza Hut", score: 95, long: -0.127758, lat: 51.507351})
  db.EX6.insertOne({name: "Pastry Chef", score: 80, long: 151.209290, lat: -33.868820})
  db.EX6.insertOne({name: "Burger King", score: 88, long: 2.352222, lat: 48.856613})

  let newDocuments = [
     {name: "KFC", score: 92, long: -139.691711, lat: 35.689487},
     {name: "Subway", score: 89, long: -96.633308, lat: -23.550520},
     {name: "Starbucks", score: 91, long: 77.209023, lat: 28.613939},
     {name: "Dunkin Donuts", score: 87, long: 72.877656, lat: 19.075984},
     {name: "Taco Bell", score: 93, long: 12.496365, lat: 41.902783}
  ]
  db.EX6.insertMany(newDocuments)
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('65f6f4fc87028d10f5c94380'),
      '1': ObjectId('65f6f4fc87028d10f5c94381'),
      '2': ObjectId('65f6f4fc87028d10f5c94382'),
      '3': ObjectId('65f6f4fc87028d10f5c94383'),
      '4': ObjectId('65f6f4fc87028d10f5c94384')
    }
  }
LAB>
```

```
LAB> db.EX6.find({ long: { $lt: -95.754168 } })
```

Register No.………………… Date:……………………………


MongoDB Compass - localhost:27017/LAB.EX6
Connect Edit View Collection Help
localhost:27017
My Queries
EX6
LAB.EX6
10 DOCUMENTS
1 INDEXES
>_MONGOSH

```
> db.EX6.find({ long: { $lt: -95.754168 } })
< {
    _id: ObjectId('65f6f4fc87028d10f5c9437c'),
    name: 'Dominos',
    score: 90,
    long: -118.243683,
    lat: 34.052235
  }
  {
    _id: ObjectId('65f6f4fc87028d10f5c94380'),
    name: 'KFC',
    score: 92,
    long: -139.691711,
    lat: 35.689487
  }
  {
    _id: ObjectId('65f6f4fc87028d10f5c94381'),
    name: 'Subway',
    score: 89,
    long: -96.633308,
    lat: -23.55052
  }
LAB>
```

Register No.………………… Date:………………………………

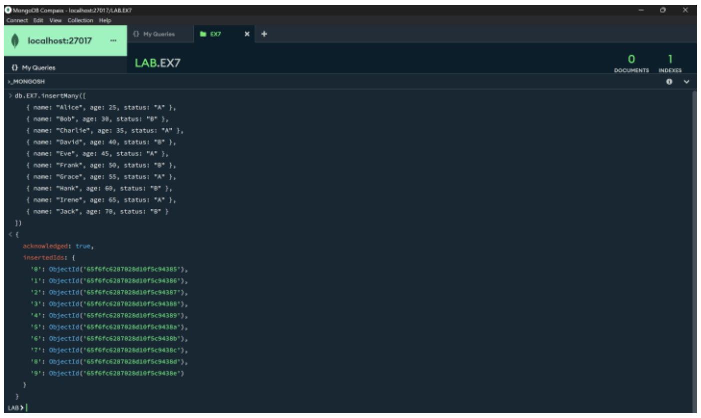
Exercise-7

AIM

To study operations in MongoDB – AND in MongoDB, OR in MongoDB, Limit Records and Sort Records. To study operations in MongoDB – Indexing, Advanced Indexing, Aggregation and Map Reduce.

DATABASE

```
LAB>db.EX7.insertMany([
  { name: "Alice", age: 25, status: "A" },
  { name: "Bob", age: 30, status: "B" },
  { name: "Charlie", age: 35, status: "A" },
  { name: "David", age: 40, status: "B" },
  { name: "Eve", age: 45, status: "A" },
  { name: "Frank", age: 50, status: "B" },
  { name: "Grace", age: 55, status: "A" },
  { name: "Hank", age: 60, status: "B" },
  { name: "Irene", age: 65, status: "A" },
  { name: "Jack", age: 70, status: "B" }
])
```


MongoDB Compass - localhost:27017/LAB.EX7
Connect Edit View Collection Help
localhost:27017
My Queries
EX7
LAB.EX7
0 DOCUMENTS
1 INDEXES
>_MONGOSH

```
> db.EX7.insertMany([
    { name: "Alice", age: 25, status: "A" },
    { name: "Bob", age: 30, status: "B" },
    { name: "Charlie", age: 35, status: "A" },
    { name: "David", age: 40, status: "B" },
    { name: "Eve", age: 45, status: "A" },
    { name: "Frank", age: 50, status: "B" },
    { name: "Grace", age: 55, status: "A" },
    { name: "Hank", age: 60, status: "B" },
    { name: "Irene", age: 65, status: "A" },
    { name: "Jack", age: 70, status: "B" }
  ])
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('65f6fc6287028d10f5c94385'),
      '1': ObjectId('65f6fc6287028d10f5c94386'),
      '2': ObjectId('65f6fc6287028d10f5c94387'),
      '3': ObjectId('65f6fc6287028d10f5c94388'),
      '4': ObjectId('65f6fc6287028d10f5c94389'),
      '5': ObjectId('65f6fc6287028d10f5c9438a'),
      '6': ObjectId('65f6fc6287028d10f5c9438b'),
      '7': ObjectId('65f6fc6287028d10f5c9438c'),
      '8': ObjectId('65f6fc6287028d10f5c9438d'),
      '9': ObjectId('65f6fc6287028d10f5c9438e')
    }
  }
LAB>
```

OPERATIONS

1. ***AND***

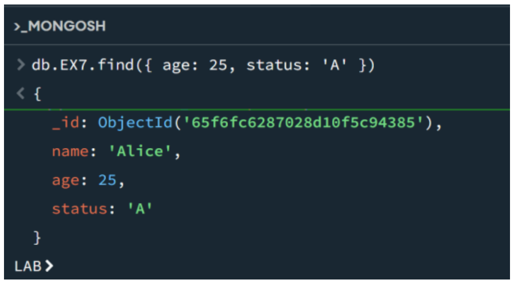
In MongoDB, you can use the comma `,` operator to perform an AND operation.

Syntax:

```
db.collection.find({ field1: value1, field2: value2 })
```

Example:

```
db.EX7.find({ age: 25, status: 'A' })
```



```
>_MONGOSH
> db.EX7.find({ age: 25, status: 'A' })
< {
    _id: ObjectId('65f6fc6287028d10f5c94385'),
    name: 'Alice',
    age: 25,
    status: 'A'
  }
LAB >
```

2. *OR*

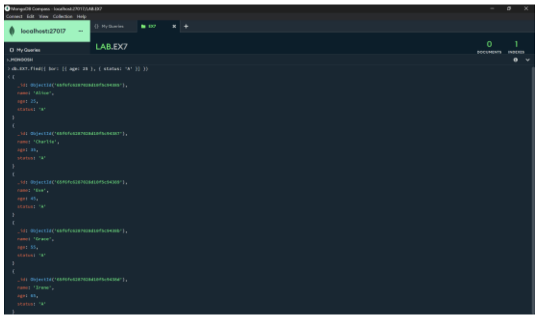
You can use the `$or` operator to perform an OR operation.

Syntax:

```
db.collection.find({ $or: [{ field1: value1 }, { field2: value2 }] })
```

Example:

db.EX7.find({ $or: [{ age: 25 }, { status: 'A' }] })



```
MongoDB Compass - localhost:27017/LAB.EX7
localhost:27017    My Queries    EX7
LAB.EX7                          0 DOCUMENTS   1 INDEXES
>_MONGOSH
> db.EX7.find({ $or: [{ age: 25 }, { status: 'A' }] })
< {
    _id: ObjectId('[illegible]'),
    name: 'Alice',
    age: 25,
    status: 'A'
  }
  {
    _id: ObjectId('[illegible]'),
    name: 'Charlie',
    age: 35,
    status: 'A'
  }
  {
    _id: ObjectId('[illegible]'),
    name: 'Eve',
    age: 45,
    status: 'A'
  }
  {
    _id: ObjectId('[illegible]'),
    name: 'Grace',
    age: 55,
    status: 'A'
  }
  {
    _id: ObjectId('[illegible]'),
    name: 'Irene',
    age: 65,
    status: 'A'
  }
```

3. *LIMIT RECORDS*

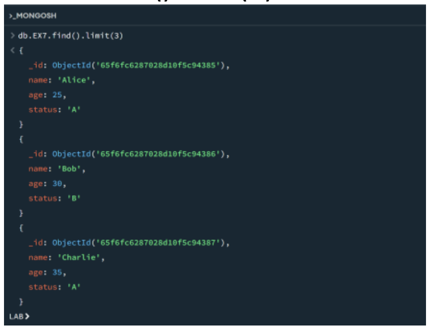
You can use the `limit()` method to limit the number of records.

Syntax:

```
db.collection.find().limit(number)
```

Example:

db.EX7.find().limit(3)



```
>_MONGOSH
> db.EX7.find().limit(3)
< {
    _id: ObjectId('65f6fc6287028d10f5c94385'),
    name: 'Alice',
    age: 25,
    status: 'A'
  }
  {
    _id: ObjectId('65f6fc6287028d10f5c94386'),
    name: 'Bob',
    age: 30,
    status: 'B'
  }
  {
    _id: ObjectId('65f6fc6287028d10f5c94387'),
    name: 'Charlie',
    age: 35,
    status: 'A'
  }
LAB >
```

4. ***SORT RECORDS***:

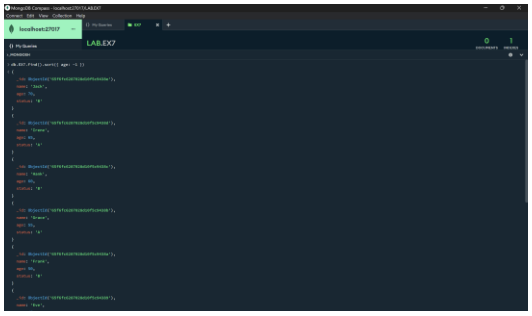
You can use the `sort()` method to sort the records.

Syntax:

db.collection.find().sort({ field1: 1 })

1 for ascending order, -1 for descending.

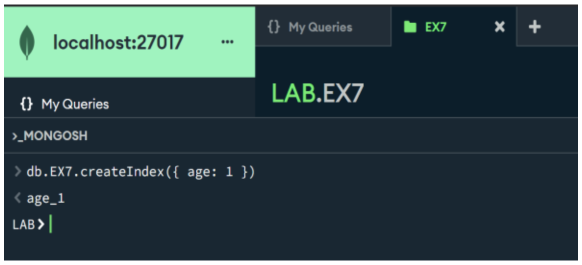
Example:
db.EX7.find().sort({ age: -1 })



```
> db.EX7.find().sort({ age: -1 })
< {
    _id: ObjectId('[illegible]'),
    name: 'Jack',
    age: 70,
    status: 'B'
  }
  {
    _id: ObjectId('[illegible]'),
    name: 'Irene',
    age: 65,
    status: 'A'
  }
  {
    _id: ObjectId('[illegible]'),
    name: 'Hank',
    age: 60,
    status: 'B'
  }
  {
    _id: ObjectId('[illegible]'),
    name: 'Grace',
    age: 55,
    status: 'A'
  }
  {
    _id: ObjectId('[illegible]'),
    name: 'Frank',
    age: 50,
    status: 'B'
  }
  {
    _id: ObjectId('[illegible]'),
    name: 'Eve',
```

5. ***INDEXING:***
Indexing in MongoDB works by storing a small portion of the collection's data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field.
Syntax:
db.collection.createIndex({ field1: 1 })

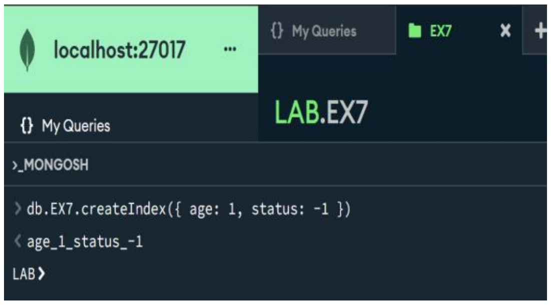
Example:
db.EX7.createIndex({ age: 1 })



```
> db.EX7.createIndex({ age: 1 })
< age_1
LAB>
```

6. ***ADVANCED INDEXING:***
MongoDB also supports advanced indexing techniques like compound indexes, multikey indexes, text indexes, etc.
Syntax of creating a compound index:
db.collection.createIndex({ field1: 1, field2: -1 })

Example:
db.EX7.createIndex({ age: 1, status: -1 })



```
> db.EX7.createIndex({ age: 1, status: -1 })
< age_1_status_-1
LAB>
```

7. *AGGREGATION:*

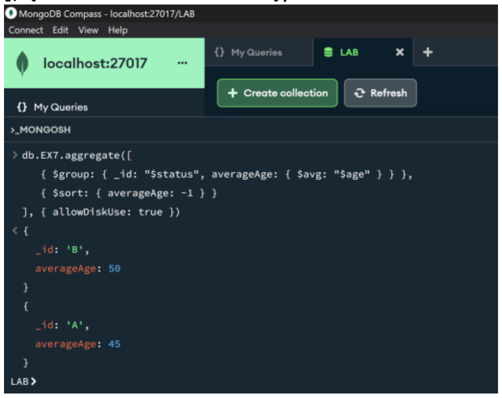
Aggregation operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. MongoDB provides three ways to perform aggregation: the aggregation pipeline, the map-reduce function, and single purpose aggregation methods.

Syntax of an aggregation pipeline:

```
db.collection.aggregate([
  { $stage1: { ... } },
  { $stage2: { ... } },
  { $stage3: { ... } },
  { $stage4: { ... } },
  ...
], { allowDiskUse: true })
```

Example:

```
db.EX7.aggregate([
  { $group: { _id: "$status", averageAge: { $avg: "$age" } } },
  { $sort: { averageAge: -1 } }
], { allowDiskUse: true })
```



```
> db.EX7.aggregate([
    { $group: { _id: "$status", averageAge: { $avg: "$age" } } },
    { $sort: { averageAge: -1 } }
  ], { allowDiskUse: true })
< {
    _id: 'B',
    averageAge: 50
  }
  {
    _id: 'A',
    averageAge: 45
  }
LAB>
```

8. *MAP REDUCE: (DEPRICATED/REMOVED FROM MongoDB)*

Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results. MongoDB uses mapReduce command for map-reduce operations.

MongoDB has deprecated the mapReduce function in favor of using the aggregate function, which can perform the same operations more efficiently.

Syntax:

```
db.collection.mapReduce(
  function() { emit(key, value); },
  function(key, values) { return reduceFunction(values) },
  {
    out: outputCollection,
    query: query, // Optional: query to select input documents
    sort: sort, // Optional: sort the input documents
    limit: limit // Optional: limit the number of input documents
  }
)
```

- ✓ **$match**: Filters the documents to pass only documents that match the specified condition(s) to the next pipeline stage.
- ✓ **$group**: Groups input documents by a specified identifier expression and applies the accumulator expression(s), if specified, to each group.
- ✓ **$sort**: Reorders the document stream by a specified sort key. Only the order changes; the documents remain unmodified.
- ✓ **$limit**: Passes the first "n" documents unmodified to the pipeline where "n" is the specified limit. For each input document, outputs either one document (unchanged) or zero documents.
- ✓ **$out**: Writes the resulting documents of the aggregation pipeline to a collection

Register No.................... Date:...............................

Exercise- 8

Column oriented databases study, queries, and practices

Column-oriented databases

Column-oriented databases, such as MongoDB, store data in columns rather than rows, which provides several advantages for analytical processing and reporting. MongoDB is a popular NoSQL document-oriented database that supports both row-oriented and column-oriented storage. It offers a flexible schema and supports various querying techniques, including document-based queries, aggregation pipelines, and indexing. Key aspects of column-oriented databases and MongoDB include:

1. **Comparison with row-oriented databases**: Column-oriented databases are optimized for analytical processing and reporting, while row-oriented databases are better suited for transactional processing. MongoDB supports both row-oriented and column-oriented storage, allowing it to handle a wide range of workloads.

2. **Performance advantages**: Column-oriented databases can achieve faster query performance by keeping data closer together and reducing seek time. MongoDB uses column-wise compression techniques to improve storage efficiency and query performance.

3. **Query examples**: MongoDB supports various querying techniques, including document-based queries, aggregation pipelines, and indexing. These features allow for efficient querying of data stored in columns.

4. **Best practices**: When working with column-oriented databases like MongoDB, it's essential to consider factors such as data modeling, indexing, and query optimization to ensure optimal performance. MongoDB provides tools and resources to help users optimize their databases for specific workloads.

In summary, column-oriented databases, such as MongoDB, offer several advantages for analytical processing and reporting, including faster query performance, efficient storage, and support for various querying techniques. By understanding the unique characteristics of column-oriented databases and best practices for working with them, users can effectively leverage these systems for their data management needs.

QUERIES

Column-oriented databases, such as MongoDB, offer several advantages for querying and processing data stored in columns. Here are some key aspects of querying in column-oriented databases:

1. **Document-based queries**: MongoDB supports document-based queries, which allow users to retrieve specific documents based on various criteria. These queries can be performed using MongoDB's query language, which supports a wide range of operators and filtering options.

2. **Aggregation pipelines**: MongoDB provides aggregation pipelines, which are a set of stages that can be applied to a collection of documents to perform various transformations and computations. These

Register No.................... Date:................................

pipelines can be used to perform complex queries, such as grouping, sorting, and filtering, on large volumes of data.

3. **Indexing**: Column-oriented databases, like MongoDB, support indexing to improve query performance. Indexes can be created on specific columns or fields, allowing for faster retrieval of data based on those columns.

4. **Query optimization**: MongoDB provides tools and resources to help users optimize their queries for specific workloads. This includes features like query profiling and explain plans, which allow users to analyze the performance of their queries and identify potential bottlenecks.

5. **Real-time analytics**: Column-oriented databases, such as MongoDB, are well-suited for real-time analytics and event-driven workloads. They can handle high volumes of data and provide fast query performance, making them suitable for applications that require real-time data processing and analysis.

In summary, column-oriented databases like MongoDB offer several advantages for querying and processing data stored in columns, including document-based queries, aggregation pipelines, indexing, query optimization, and support for real-time analytics. By understanding the unique features and capabilities of column-oriented databases, users can effectively leverage these systems for their data management and analysis needs.

Practises

For column-oriented databases like MongoDB, there are several practices and techniques that can be employed to optimize performance and improve query efficiency. Some of these practices include:

1. **Data modeling**: Choosing an appropriate data model is crucial for efficient data access and querying. MongoDB supports various data modeling approaches, such as document-oriented modeling and schema-less modeling. It is essential to consider factors like data size, query patterns, and performance requirements when designing the data model.

2. **Indexing**: Indexing is a critical aspect of MongoDB performance tuning. Indexes help speed up read operations by allowing the database to locate documents more efficiently. By creating appropriate indexes for frequently queried fields, you can significantly improve the performance of your queries. MongoDB supports various index types, including single-field indexes, compound indexes, and partial indexes.

3. **Query optimization**: Analyzing query performance and optimizing queries can help improve the efficiency of your database. MongoDB provides tools like the $explain operator, which allows you to analyze query performance and identify potential bottlenecks. Additionally, optimizing query patterns, such as using projection to limit the returned fields and utilizing query operators and aggregation for efficient filtering and data manipulation, can help improve query performance.

4. **Performance tuning**: Tuning hardware and OS configuration, managing resources, and optimizing schema design can all contribute to better MongoDB performance. For example, selecting the right hardware, managing connection pools, and configuring read and write concerns can help balance performance and availability.

5. **Query execution**: Understanding the physical layout of column-oriented databases, such as how data is stored and accessed, can help optimize query performance. Techniques like tuple construction and join processing can be optimized to improve I/O and CPU utilization.

   By following these practices and techniques, you can effectively optimize the performance of your column-oriented database, such as MongoDB, and achieve better query efficiency and faster data retrieval.

Register No.………………… Date:……………………………

Exercise-. 9

Create a database that stores road cars. Cars have a manufacturer, a type. Each car has a maximum performance and a maximum torque value. Do the following: Test Cassandras replication schema and consistency models..

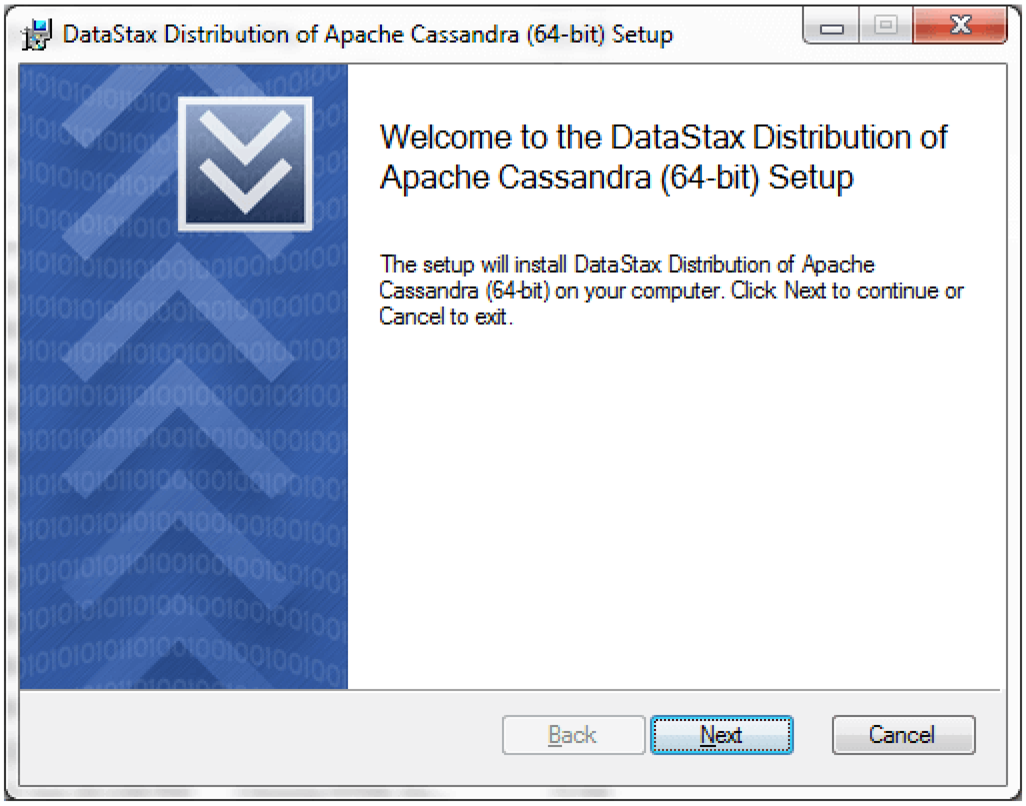
Installation : Download the Cassandra software from the link below

https://datastax-community-edition.software.informer.com/download/

Download and Install Cassandra

Follow the steps given below:

- Run the datastax community edition setup. After running the setup, you will see the following page will be displayed. It is a screenshot of 64 bit version.



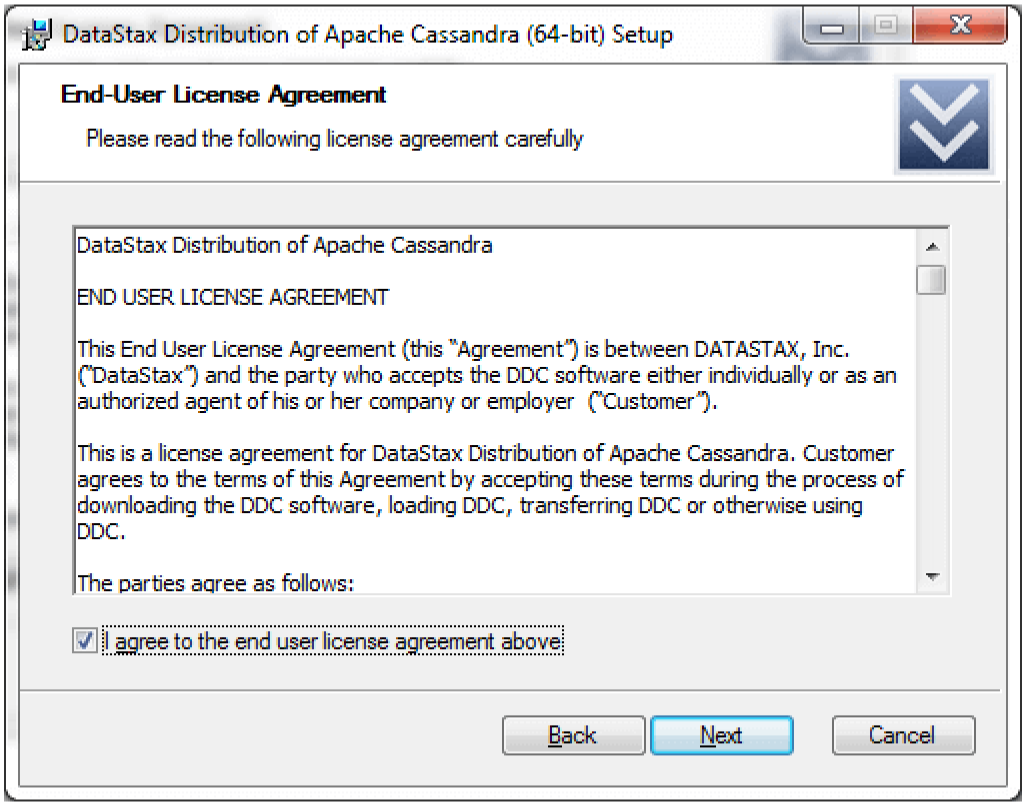
DataStax Distribution of Apache Cassandra (64-bit) Setup

Welcome to the DataStax Distribution of Apache Cassandra (64-bit) Setup

The setup will install DataStax Distribution of Apache Cassandra (64-bit) on your computer. Click Next to continue or Cancel to exit.

- Click on the next button and you will get the following page:

Register No................... Date:..............................


DataStax Distribution of Apache Cassandra (64-bit) Setup

End-User License Agreement

Please read the following license agreement carefully

DataStax Distribution of Apache Cassandra

END USER LICENSE AGREEMENT

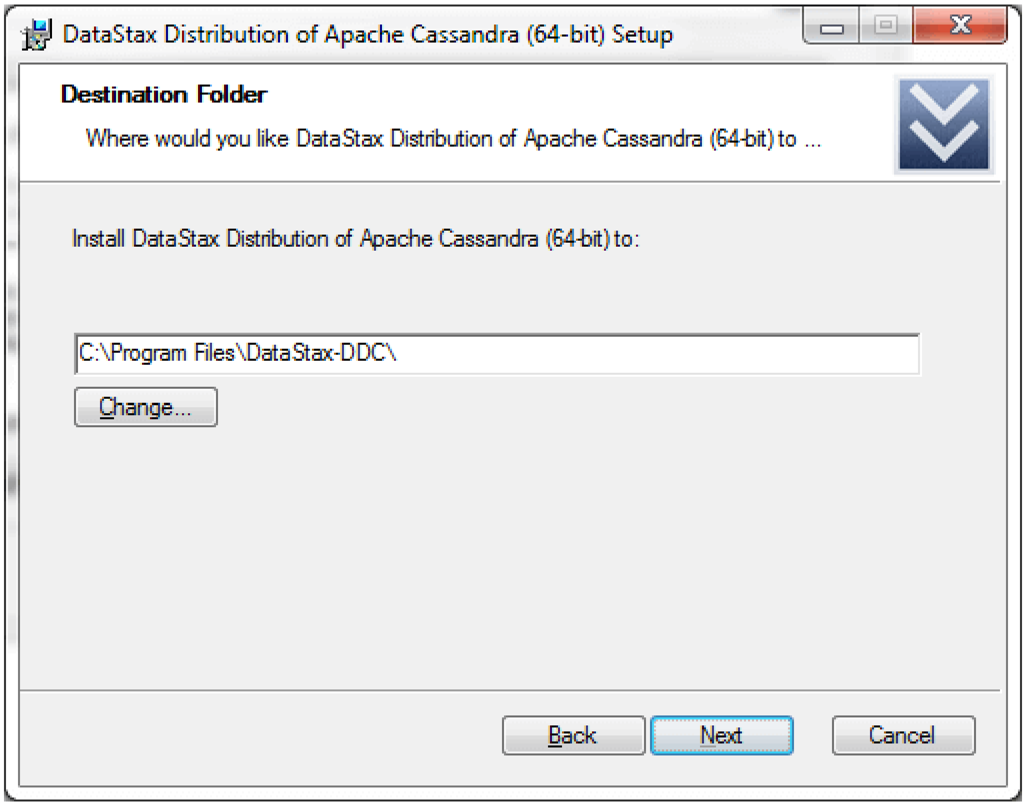
This End User License Agreement (this "Agreement") is between DATASTAX, Inc. ("DataStax") and the party who accepts the DDC software either individually or as an authorized agent of his or her company or employer ("Customer").

This is a license agreement for DataStax Distribution of Apache Cassandra. Customer agrees to the terms of this Agreement by accepting these terms during the process of downloading the DDC software, loading DDC, transferring DDC or otherwise using DDC.

The parties agree as follows:

[✓] I agree to the end user license agreement above

- Press the next button and you will get the following page specifying the location of the installation.


DataStax Distribution of Apache Cassandra (64-bit) Setup

Destination Folder

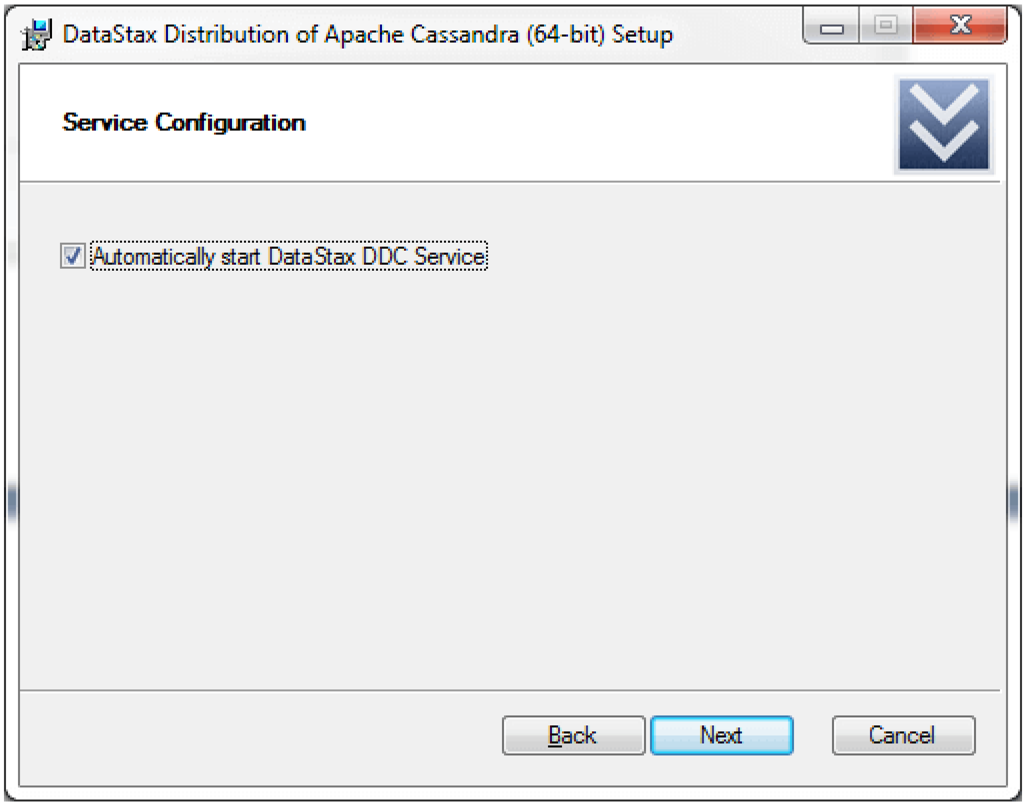
Where would you like DataStax Distribution of Apache Cassandra (64-bit) to ...

Install DataStax Distribution of Apache Cassandra (64-bit) to:

C:\Program Files\DataStax-DDC\

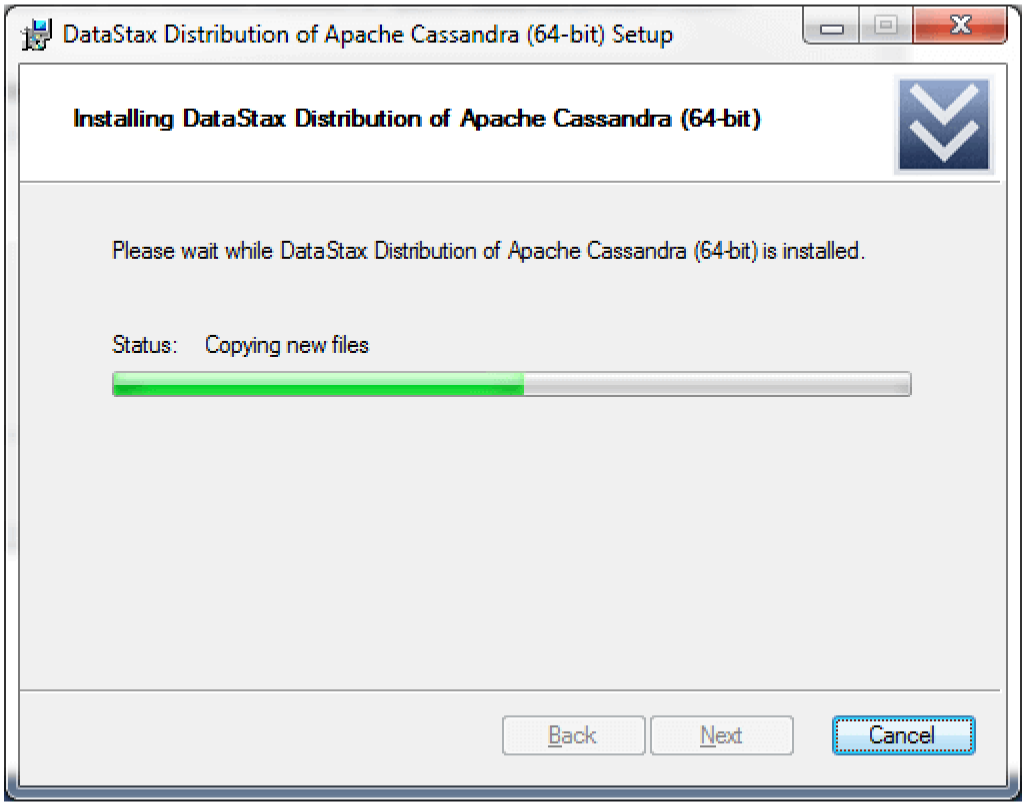
Register No.................... Date:...............................

- Press the next button and a page will appear asking about whether you automatically start Data Stax DDC service. Click on the radio button and proceed next.


DataStax Distribution of Apache Cassandra (64-bit) Setup

Service Configuration

[✓] Automatically start DataStax DDC Service


DataStax Distribution of Apache Cassandra (64-bit) Setup

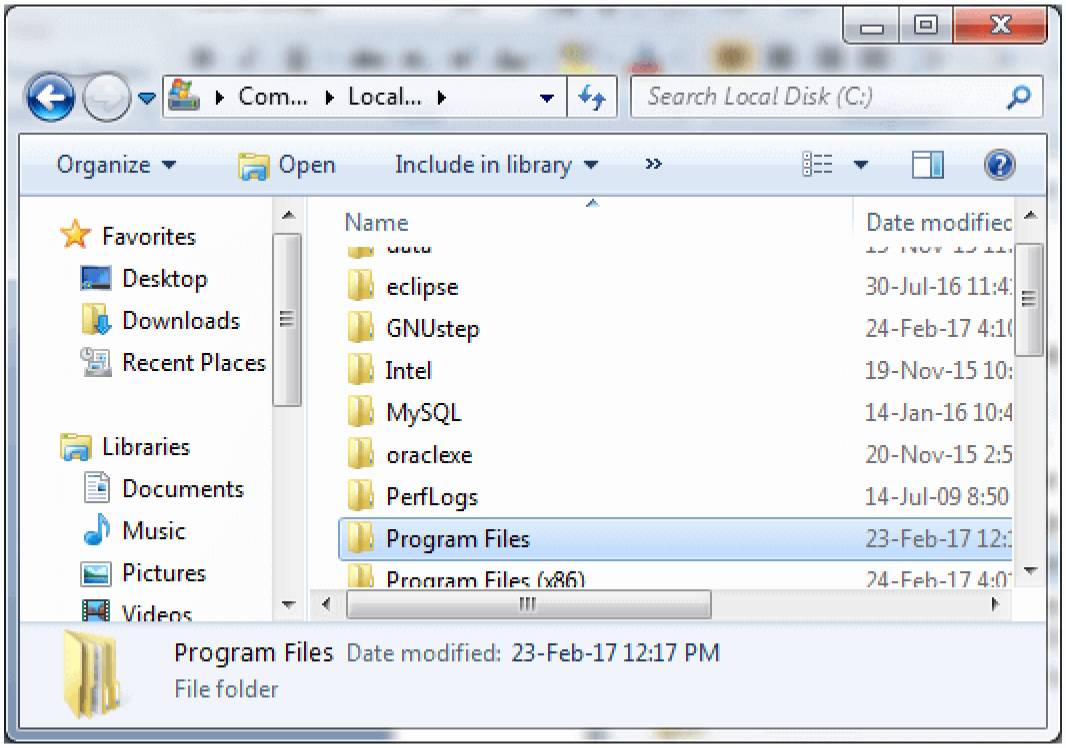
Installing DataStax Distribution of Apache Cassandra (64-bit)

Please wait while DataStax Distribution of Apache Cassandra (64-bit) is installed.

Status: Copying new files

- Installation is started now. After completing the installation, go to program files where Data Stax is installed.

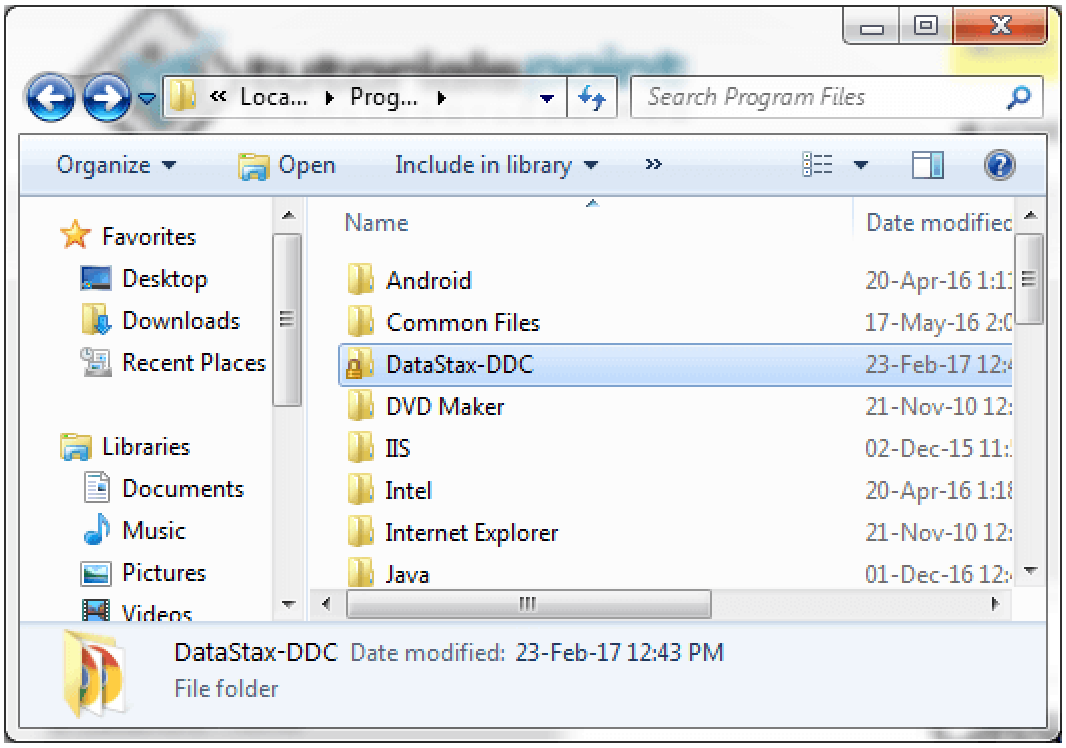
Register No.................... Date:..............................



Name	Date modified
eclipse	30-Jul-16 11:4
GNUstep	24-Feb-17 4:1
Intel	19-Nov-15 10:
MySQL	14-Jan-16 10:4
oraclexe	20-Nov-15 2:5
PerfLogs	14-Jul-09 8:50
Program Files	23-Feb-17 12:
Program Files (x86)	24-Feb-17 4:0

Program Files Date modified: 23-Feb-17 12:17 PM
File folder

- Open Program Files then you see the following page:

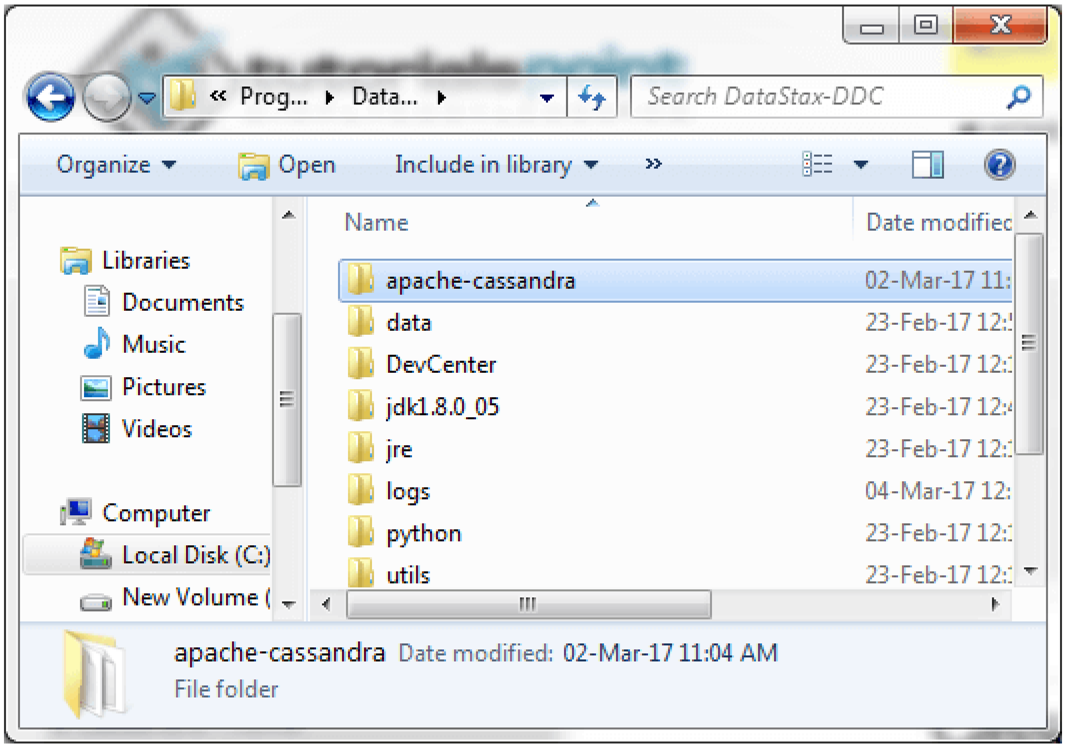


Name	Date modified
Android	20-Apr-16 1:1
Common Files	17-May-16 2:0
DataStax-DDC	23-Feb-17 12:
DVD Maker	21-Nov-10 12:
IIS	02-Dec-15 11:
Intel	20-Apr-16 1:1
Internet Explorer	21-Nov-10 12:
Java	01-Dec-16 12:

DataStax-DDC Date modified: 23-Feb-17 12:43 PM
File folder

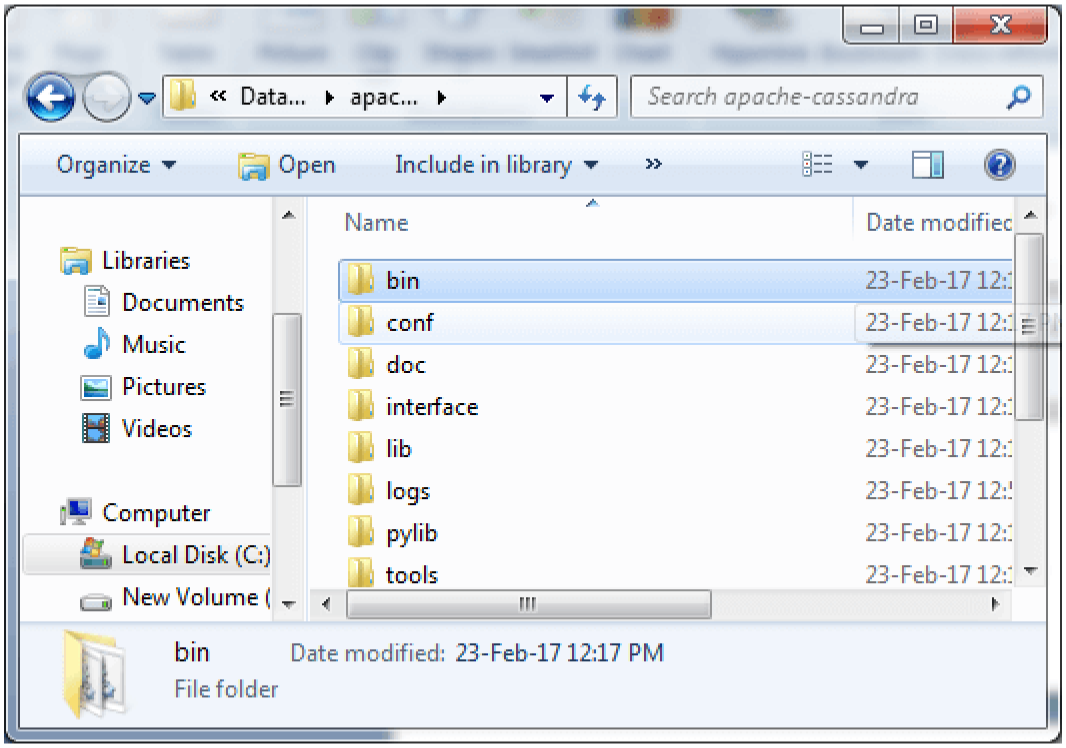
-

- Open DataStax-DDC then you see Apache Cassandra:



```
Name                 Date modified
apache-cassandra     02-Mar-17 11:
data                 23-Feb-17 12:
DevCenter            23-Feb-17 12:
jdk1.8.0_05          23-Feb-17 12:
jre                  23-Feb-17 12:
logs                 04-Mar-17 12:
python               23-Feb-17 12:
utils                23-Feb-17 12:

apache-cassandra  Date modified: 02-Mar-17 11:04 AM
File folder
```

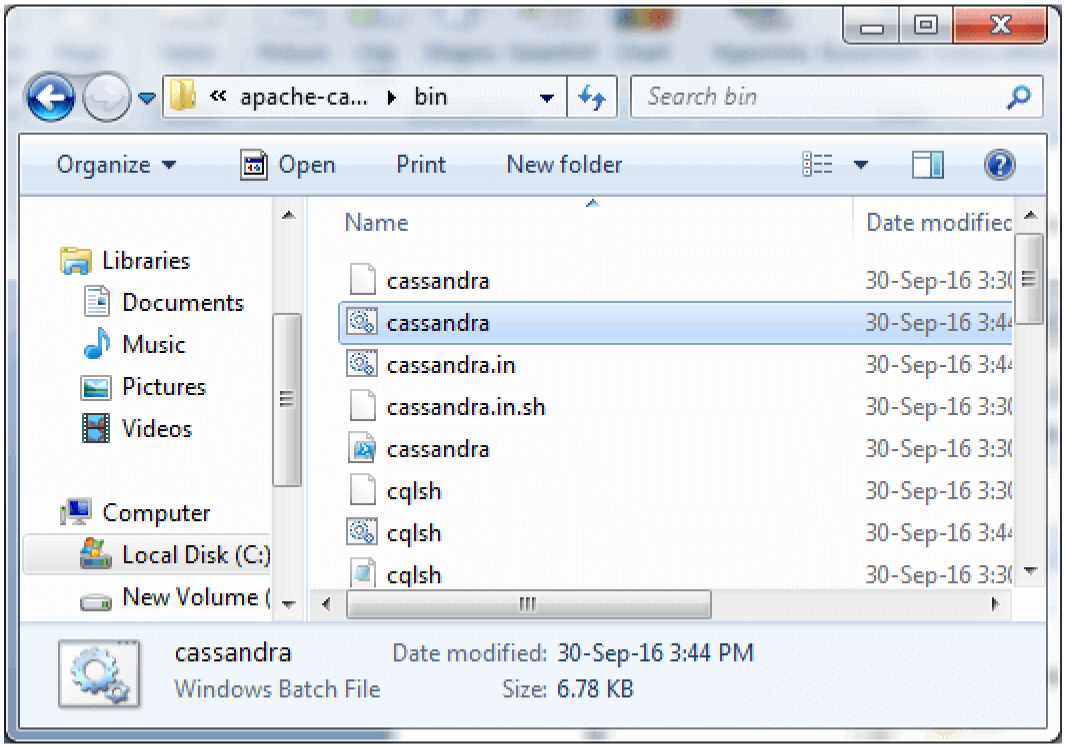
- Open Apache Cassandra and you see bin:



```
Name                 Date modified
bin                  23-Feb-17 12:
conf                 23-Feb-17 12:
doc                  23-Feb-17 12:
interface            23-Feb-17 12:
lib                  23-Feb-17 12:
logs                 23-Feb-17 12:
pylib                23-Feb-17 12:
tools                23-Feb-17 12:

bin         Date modified: 23-Feb-17 12:17 PM
File folder
```

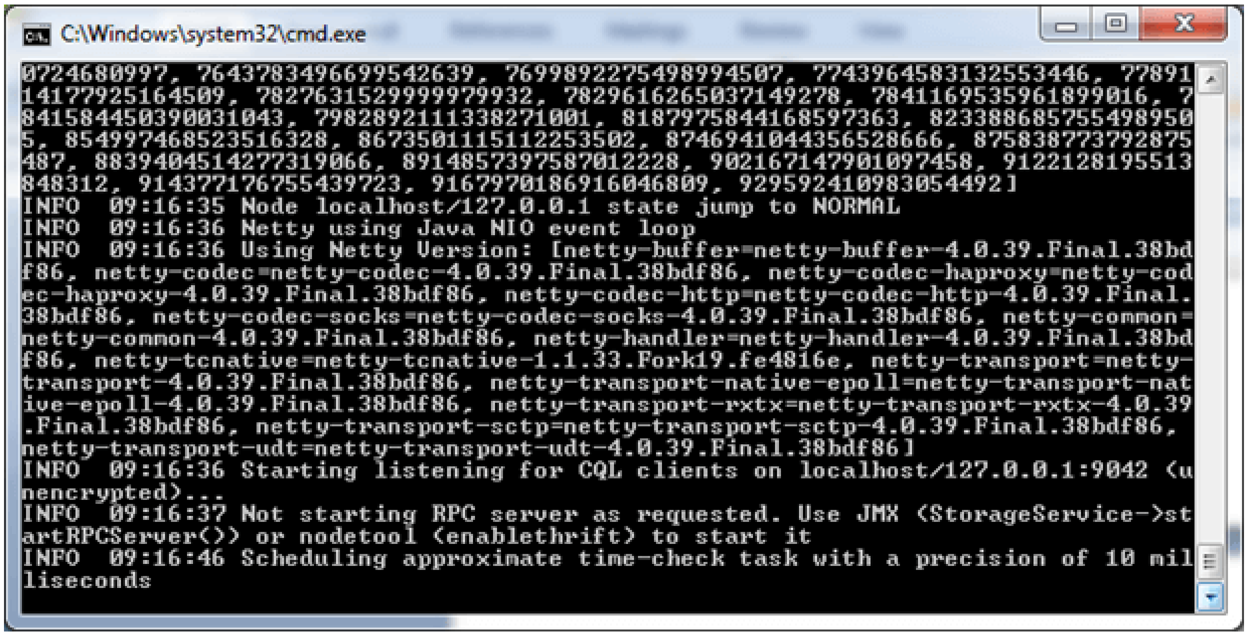
- Open bin and you will see Cassandra Windows batch File:

Register No.................... Date:.................................



```
Name                Date modified
cassandra           30-Sep-16 3:3
cassandra           30-Sep-16 3:4
cassandra.in        30-Sep-16 3:4
cassandra.in.sh     30-Sep-16 3:3
cassandra           30-Sep-16 3:3
cqlsh               30-Sep-16 3:3
cqlsh               30-Sep-16 3:4
cqlsh               30-Sep-16 3:3

cassandra           Date modified: 30-Sep-16 3:44 PM
Windows Batch File  Size: 6.78 KB
```

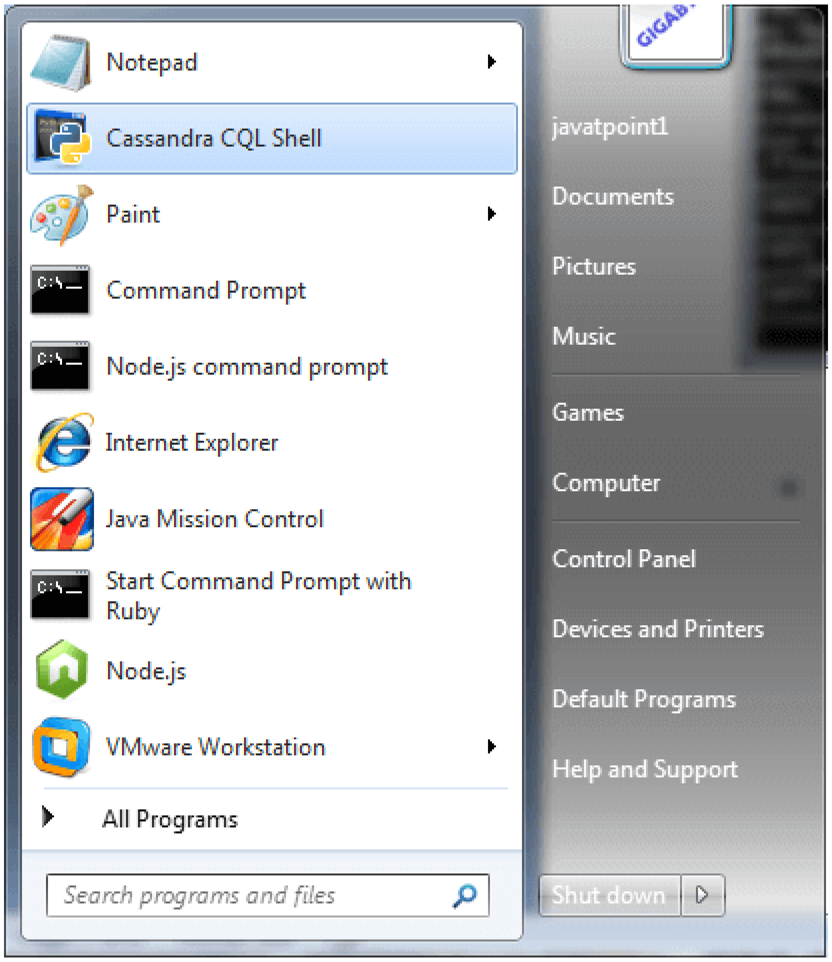
- Run this file. It will start the Cassandra server and you will see the following page:



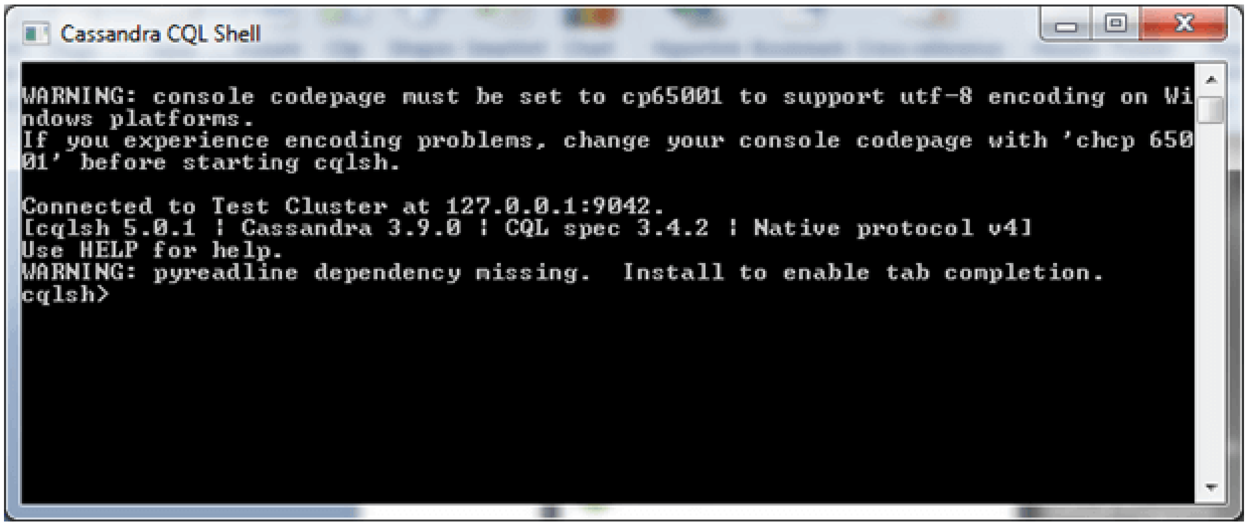
```
C:\Windows\system32\cmd.exe
0724680997, 7643783496699542639, 7699892275498994507, 7743964583132553446, 77891
14177925164509, 7827631529999979932, 7829616265037149278, 7841169535961899016, 7
8415844503900310 43, 7982892111338271001, 8187975844168597363, 823388685755498950
5, 8549974685235163 28, 8673501115112253502, 8746941044356528666, 875838773792875
487, 8839404514277319066, 8914857397587012228, 9021671479010974 58, 912212819551 3
848312, 9143771767554397 23, 9167970186916046809, 9295924109830544921
INFO  09:16:35 Node localhost/127.0.0.1 state jump to NORMAL
INFO  09:16:36 Netty using Java NIO event loop
INFO  09:16:36 Using Netty Version: [netty-buffer=netty-buffer-4.0.39.Final.38bd
f86, netty-codec=netty-codec-4.0.39.Final.38bdf86, netty-codec-haproxy=netty-cod
ec-haproxy-4.0.39.Final.38bdf86, netty-codec-http=netty-codec-http-4.0.39.Final.
38bdf86, netty-codec-socks=netty-codec-socks-4.0.39.Final.38bdf86, netty-common=
netty-common-4.0.39.Final.38bdf86, netty-handler=netty-handler-4.0.39.Final.38bd
f86, netty-tcnative=netty-tcnative-1.1.33.Fork19.fe4816e, netty-transport=netty-
transport-4.0.39.Final.38bdf86, netty-transport-native-epoll=netty-transport-nat
ive-epoll-4.0.39.Final.38bdf86, netty-transport-rxtx=netty-transport-rxtx-4.0.39
.Final.38bdf86, netty-transport-sctp=netty-transport-sctp-4.0.39.Final.38bdf86,
netty-transport-udt=netty-transport-udt-4.0.39.Final.38bdf86]
INFO  09:16:36 Starting listening for CQL clients on localhost/127.0.0.1:9042 (u
nencrypted)...
INFO  09:16:37 Not starting RPC server as requested. Use JMX (StorageService->st
artRPCServer()) or nodetool (enablethrift) to start it
INFO  09:16:46 Scheduling approximate time-check task with a precision of 10 mil
liseconds
```

- Server is started now go to windows start programs, search Cassandra Shell

Register No..................   Date:.............................



- Run the Cassandra Shell. After running Cassandra shell, you will see the following command line:



Cassandra CQL Shell

```
WARNING: console codepage must be set to cp65001 to support utf-8 encoding on Wi
ndows platforms.
If you experience encoding problems, change your console codepage with 'chcp 650
01' before starting cqlsh.

Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.9.0 | CQL spec 3.4.2 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing.  Install to enable tab completion.
cqlsh>
```

Description

Register No.………………… Date:……………………………

Step 1: Define Keyspace

In Cassandra, a keyspace is like a schema in relational databases. It defines the replication strategy and other options for data distribution.

```
CREATE KEYSPACE IF NOT EXISTS car_keyspace
WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 3};
```

In this example, we are using SimpleStrategy with a replication factor of 3 for simplicity. In a production environment, you would adjust the replication strategy based on your needs.

Step 2: Create Table

Now, let's create a table to store information about road cars.

```
CREATE TABLE IF NOT EXISTS car_keyspace.cars (
   car_id UUID PRIMARY KEY,
   manufacturer TEXT,
   type TEXT,
   max_performance DOUBLE,
   max_torque DOUBLE
);
```

This table has columns for car_id (primary key), manufacturer, type, max_performance, and max_torque.

Step 3: Insert Data

You can insert data into the table using INSERT statements.

```
INSERT INTO car_keyspace.cars (car_id, manufacturer, type, max_performance, max_torque)
VALUES (uuid(), 'Toyota', 'SUV', 300, 400);

INSERT INTO car_keyspace.cars (car_id, manufacturer, type, max_performance, max_torque)
VALUES (uuid(), 'BMW', 'Sedan', 350, 450);

INSERT INTO car_keyspace.cars (car_id, manufacturer, type, max_performance, max_torque)
VALUES (uuid(), 'Ford', 'Truck', 250, 350);
```

Step 4: Query Data

You can retrieve data from the table using SELECT statements.

```
SELECT * FROM car_keyspace.cars
```

Step 5: Replication and Consistency

Cassandra provides various replication strategies and consistency levels. By default, Cassandra uses eventual consistency, but you can configure consistency levels on a per-query basis or globally.

Replication can be tested by setting up a Cassandra cluster with multiple nodes and observing how data is replicated across nodes.

Consistency can be tested by performing read and write operations with different consistency levels and observing the results.

For example:

```
INSERT INTO car_keyspace.cars (car_id, manufacturer, type, max_performance, max_torque)
VALUES (uuid(), 'Mercedes', 'Coupe', 400, 500) USING CONSISTENCY QUORUM;
SELECT * FROM car_keyspace.cars USING CONSISTENCY QUORUM;
```

In this example, the INSERT statement specifies a consistency level of QUORUM, ensuring that the write operation is acknowledged by a majority of replicas. The subsequent SELECT statement also uses QUORUM consistency level, ensuring that the data retrieved is consistent across replicas.

By experimenting with different consistency levels and observing the results, you can test Cassandra's consistency models effectively.

Program

Step 1

CREATE KEYSPACE IF NOT EXISTS car_keyspace

... WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 3};

sh> CREATE TABLE IF NOT EXISTS car_keyspace.cars (

...   car_id UUID PRIMARY KEY,

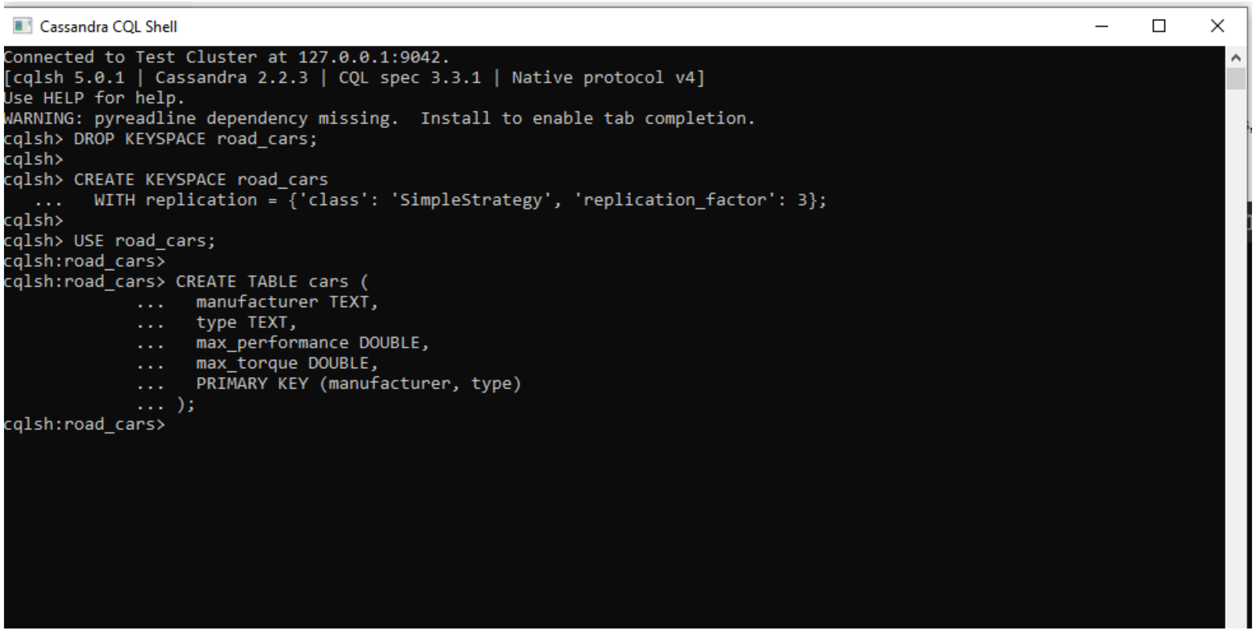
...   manufacturer TEXT,

...   type TEXT,

...   max_performance DOUBLE,

...   max_torque DOUBLE

...);



```
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 2.2.3 | CQL spec 3.3.1 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing.  Install to enable tab completion.
cqlsh> DROP KEYSPACE road_cars;
cqlsh>
cqlsh> CREATE KEYSPACE road_cars
   ...   WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 3};
cqlsh>
cqlsh> USE road_cars;
cqlsh:road_cars>
cqlsh:road_cars> CREATE TABLE cars (
             ...   manufacturer TEXT,
             ...   type TEXT,
             ...   max_performance DOUBLE,
             ...   max_torque DOUBLE,
             ...   PRIMARY KEY (manufacturer, type)
             ... );
cqlsh:road_cars>
```

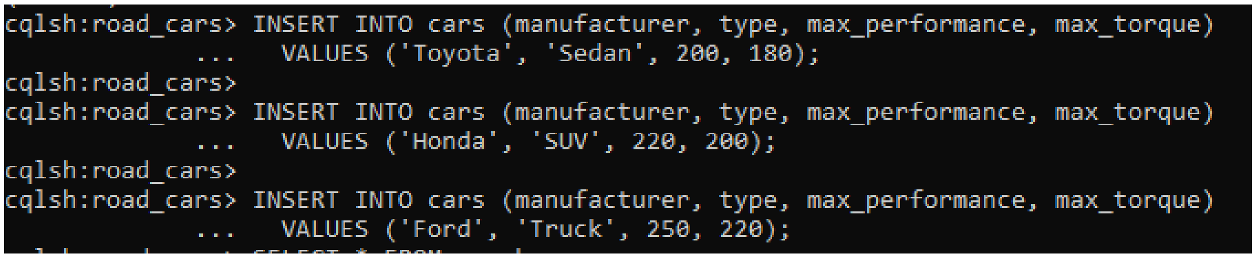
sh> INSERT INTO car_keyspace.cars (car_id, manufacturer, type, max_performance, max_torque)

... VALUES (uuid(), 'Toyota', 'SUV', 300, 400);

sh>

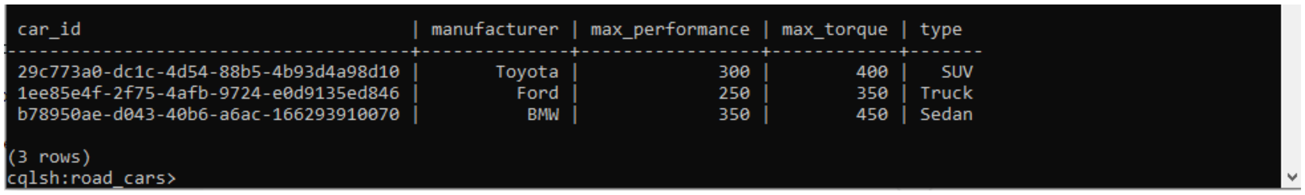
sh> INSERT INTO car_keyspace.cars (car_id, manufacturer, type, max_performance, max_torque)

```
  ... VALUES (uuid(), 'BMW', 'Sedan', 350, 450);

sh>

sh> INSERT INTO car_keyspace.cars (car_id, manufacturer, type, max_performance, max_torque)

  ... VALUES (uuid(), 'Ford', 'Truck', 250, 350);
```



```
cqlsh:road_cars> INSERT INTO cars (manufacturer, type, max_performance, max_torque)
             ...   VALUES ('Toyota', 'Sedan', 200, 180);
cqlsh:road_cars>
cqlsh:road_cars> INSERT INTO cars (manufacturer, type, max_performance, max_torque)
             ...   VALUES ('Honda', 'SUV', 220, 200);
cqlsh:road_cars>
cqlsh:road_cars> INSERT INTO cars (manufacturer, type, max_performance, max_torque)
             ...   VALUES ('Ford', 'Truck', 250, 220);
```

```
sh> SELECT * FROM car_keyspace.cars;

 car_id                               | manufacturer | max_performance | max_torque | type
--------------------------------------+--------------+-----------------+------------+-------
 29c773a0-dc1c-4d54-88b5-4b93d4a98d10 |       Toyota |             300 |        400 |   SUV
 1ee85e4f-2f75-4afb-9724-e0d9135ed846 |         Ford |             250 |        350 | Truck
 b78950ae-d043-40b6-a6ac-166293910070 |          BMW |             350 |        450 | Sedan
```



```
 car_id                               | manufacturer | max_performance | max_torque | type
--------------------------------------+--------------+-----------------+------------+-------
 29c773a0-dc1c-4d54-88b5-4b93d4a98d10 |       Toyota |             300 |        400 |   SUV
 1ee85e4f-2f75-4afb-9724-e0d9135ed846 |         Ford |             250 |        350 | Truck
 b78950ae-d043-40b6-a6ac-166293910070 |          BMW |             350 |        450 | Sedan

(3 rows)
cqlsh:road_cars>
```

(3 rows)

Register No.....................   Date:................................

Exercise– 10

Master Data Management using Neo4j Manage your master data more effectively the world of master data is changing. Data architects and application developers are swapping their relational databases with graph databases to store their master data. This switch enables them to use a data store optimized to discover new insights in existing data, provide a360-degree view of master data and answer questions about data relationships in real time write an essay with examples and explanation with output

Title: Unleashing the Power of Neo4j for Master Data Management

In the dynamic landscape of data management, the traditional paradigm of using relational databases for master data management is undergoing a paradigm shift. Data architects and application developers are increasingly turning to graph databases, notably Neo4j, to store and manage their master data. This transformation heralds a new era of master data management, one characterized by enhanced agility, deeper insights, and real-time relationship exploration. This essay explores the evolution of master data management with Neo4j, elucidating its benefits, and providing examples to illustrate its transformative capabilities.

Understanding Neo4j for Master Data Management:

Neo4j is a leading graph database that excels in representing and querying highly connected data. Unlike relational databases, which rely on tabular structures, Neo4j employs a graph-based model consisting of nodes, relationships, and properties. This model is particularly well-suited for master data management, as it enables the representation of complex relationships between entities in a natural and intuitive manner.

Benefits of Neo4j for Master Data Management:

1. Discovering New Insights: By leveraging the graph-based nature of Neo4j, organizations can uncover hidden relationships and patterns within their master data. For example, in a customer master data scenario, Neo4j can reveal previously unrecognized connections between customers based on shared attributes, interactions, or transactions.

2. 360-Degree View of Master Data: Neo4j facilitates the creation of a holistic view of master data by capturing not only the entities themselves but also the relationships between them. This enables organizations to gain a comprehensive understanding of their data landscape, facilitating informed decision-making and strategic planning.

3. Real-Time Relationship Exploration: With Neo4j, organizations can explore data relationships in real-time, enabling dynamic analysis and visualization of interconnected data. For instance, in a product master data scenario, Neo4j can provide immediate insights into product hierarchies, dependencies, and associations, empowering organizations to adapt quickly to changing market dynamics.

Example: Customer Master Data Management with Neo4j:

Consider a scenario where a retail company seeks to manage its customer master data using Neo4j. The company stores information about customers, their purchases, and interactions across various touchpoints. Let's explore how Neo4j can facilitate this process:

```cypher
// Create nodes for customers and products
CREATE (:Customer {id: 1, name: 'Alice'}),
    (:Customer {id: 2, name: 'Bob'}),
    (:Product {id: 'P1', name: 'Laptop'}),
    (:Product {id: 'P2', name: 'Smartphone'})

// Create relationships between customers and products
MATCH (c:Customer {name: 'Alice'}), (p:Product {id: 'P1'})
CREATE (c)-[:PURCHASED]->(p)

MATCH (c:Customer {name: 'Bob'}), (p:Product {id: 'P2'})
CREATE (c)-[:PURCHASED]->(p)
```

In this example, we create nodes representing customers and products, and establish relationships between customers and the products they have purchased. This graph-based representation enables the company to easily query and analyze customer purchase patterns, identify cross-selling opportunities, and personalize marketing campaigns based on customer preferences and behavior.

Conclusion:

In conclusion, Neo4j represents a paradigm shift in master data management, offering organizations a powerful platform to store, manage, and analyze their master data. By leveraging the inherent capabilities of graph databases, Neo4j enables organizations to gain deeper insights, create holistic views of their data, and explore data relationships in real-time. Through examples like customer master data management, it becomes evident that Neo4j's graph-based approach revolutionizes traditional master data management practices, empowering organizations to unlock new possibilities and drive innovation in the digital age.

Register No.................... Date:.................................

Exercise– 11

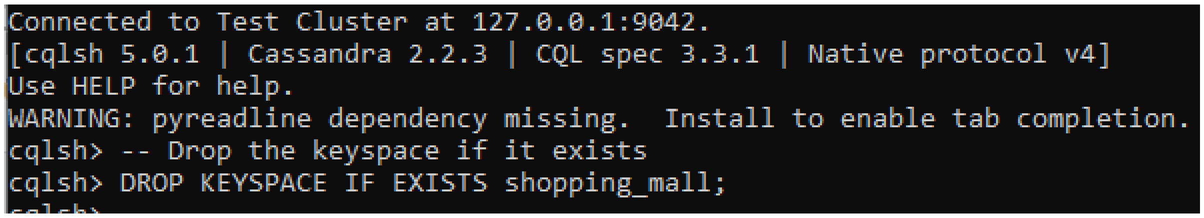
Shopping Mall case study using Cassandra, where we have many customers ordering items from thermall and, we have suppliers who deliver them their ordered items

AIM: Shopping Mall case study using Cassandra, where we have many customers ordering items from thermall and, we have suppliers who deliver them their ordered items.

Description: In this lab experiment, we explore database management with Apache Cassandra. They begin by creating a keyspace to organize data, then establish tables for customers, suppliers, products, orders, and order details. Through CQL commands, they insert sample data representing customers, suppliers, and products. The experiment delves into querying techniques, enabling students to retrieve specific customer orders and analyze relationships between entities. Through hands-on practice, we gain proficiency in database schema design, data insertion, and retrieval, while comprehending the fundamentals of distributed database systems exemplified by Cassandra's schema flexibility and **eventual consistency model.**

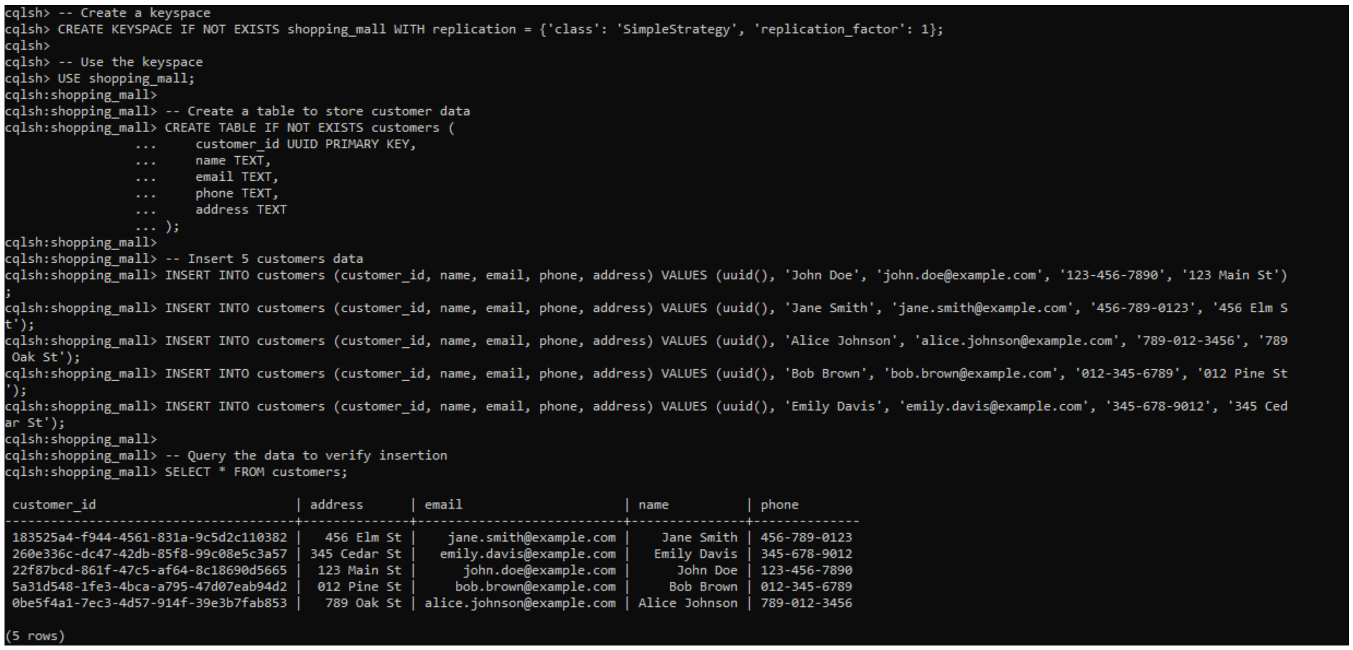
Procedure:

Step 1: Once you run this command, it will remove the keyspace shopping_mall along with all the tables and data inside it. Make sure you're certain about dropping the keyspace as it will delete all data associated with it.



```
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 2.2.3 | CQL spec 3.3.1 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing.  Install to enable tab completion.
cqlsh> -- Drop the keyspace if it exists
cqlsh> DROP KEYSPACE IF EXISTS shopping_mall;
```

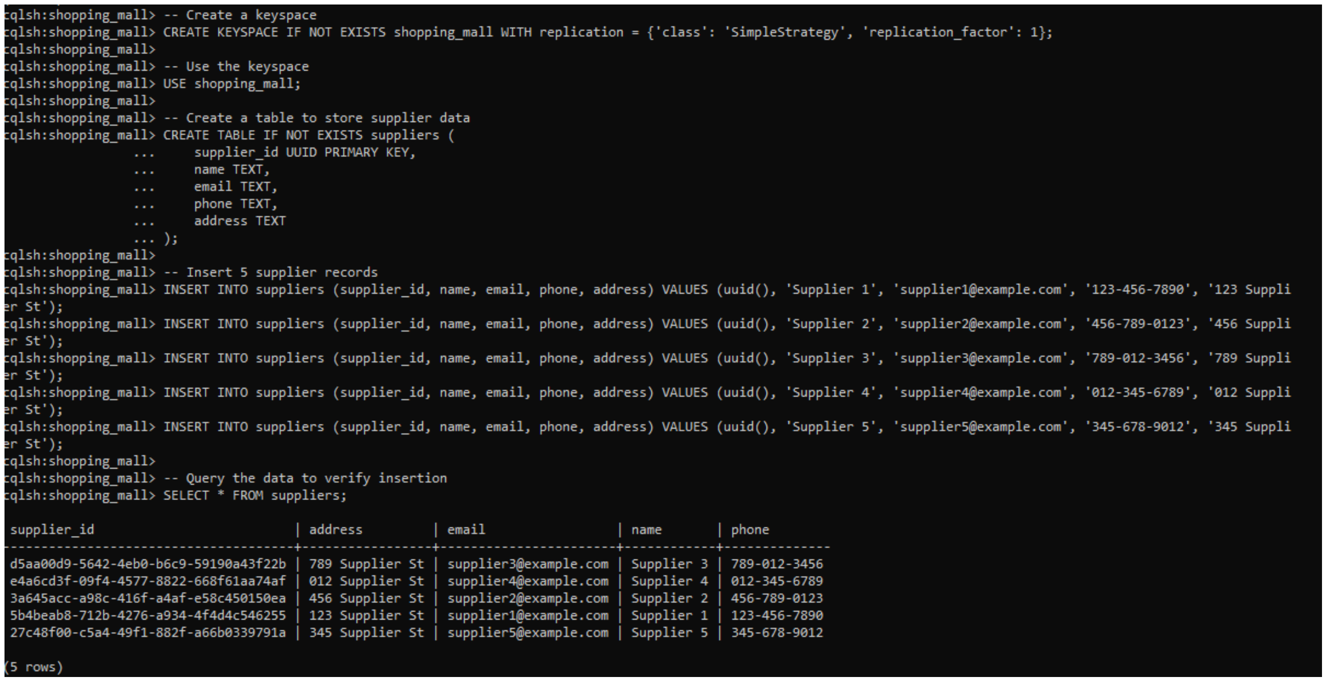
Step 2: Create a keyspace, define a table for customer data, and insert 5 customer records into a Cassandra database using CQL (Cassandra Query Language) in the Cassandra CQL shell:



```
cqlsh> -- Create a keyspace
cqlsh> CREATE KEYSPACE IF NOT EXISTS shopping_mall WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
cqlsh>
cqlsh> -- Use the keyspace
cqlsh> USE shopping_mall;
cqlsh:shopping_mall>
cqlsh:shopping_mall> -- Create a table to store customer data
cqlsh:shopping_mall> CREATE TABLE IF NOT EXISTS customers (
                 ...     customer_id UUID PRIMARY KEY,
                 ...     name TEXT,
                 ...     email TEXT,
                 ...     phone TEXT,
                 ...     address TEXT
                 ... );
cqlsh:shopping_mall>
cqlsh:shopping_mall> -- Insert 5 customers data
cqlsh:shopping_mall> INSERT INTO customers (customer_id, name, email, phone, address) VALUES (uuid(), 'John Doe', 'john.doe@example.com', '123-456-7890', '123 Main St')
;
cqlsh:shopping_mall> INSERT INTO customers (customer_id, name, email, phone, address) VALUES (uuid(), 'Jane Smith', 'jane.smith@example.com', '456-789-0123', '456 Elm S
t');
cqlsh:shopping_mall> INSERT INTO customers (customer_id, name, email, phone, address) VALUES (uuid(), 'Alice Johnson', 'alice.johnson@example.com', '789-012-3456', '789
 Oak St');
cqlsh:shopping_mall> INSERT INTO customers (customer_id, name, email, phone, address) VALUES (uuid(), 'Bob Brown', 'bob.brown@example.com', '012-345-6789', '012 Pine St
');
cqlsh:shopping_mall> INSERT INTO customers (customer_id, name, email, phone, address) VALUES (uuid(), 'Emily Davis', 'emily.davis@example.com', '345-678-9012', '345 Ced
ar St');
cqlsh:shopping_mall>
cqlsh:shopping_mall> -- Query the data to verify insertion
cqlsh:shopping_mall> SELECT * FROM customers;

 customer_id                          | address      | email                     | name          | phone
--------------------------------------+--------------+---------------------------+---------------+--------------
 183525a4-f944-4561-831a-9c5d2c110382 |   456 Elm St |    jane.smith@example.com |    Jane Smith | 456-789-0123
 260e336c-dc47-42db-85f8-99c08e5c3a57 | 345 Cedar St |   emily.davis@example.com |   Emily Davis | 345-678-9012
 22f87bcd-861f-47c5-af64-8c18690d5665 |  123 Main St |      john.doe@example.com |      John Doe | 123-456-7890
 5a31d548-1fe3-4bca-a795-47d07eab94d2 |  012 Pine St |     bob.brown@example.com |     Bob Brown | 012-345-6789
 0be5f4a1-7ec3-4d57-914f-39e3b7fab853 |   789 Oak St | alice.johnson@example.com | Alice Johnson | 789-012-3456

(5 rows)
```

This code will create a keyspace named shopping_mall, a table named customers to store customer data, and insert 5 sample customer records into the table. Finally, it selects and displays all the customer data to verify that the insertion was successful.

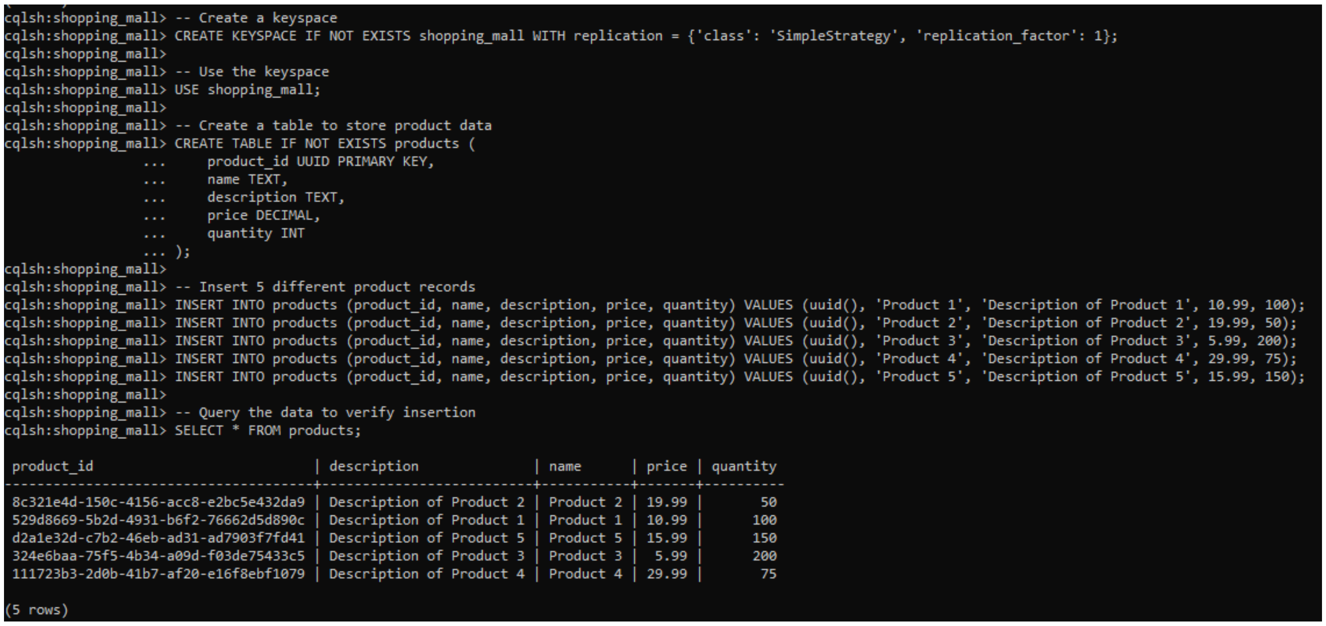
Step 3: This code will create a table named suppliers to store supplier data, and insert 5 sample supplier records into the table. Finally, it selects and displays all the supplier data to verify that the insertion was successful.



```
cqlsh:shopping_mall> -- Create a keyspace
cqlsh:shopping_mall> CREATE KEYSPACE IF NOT EXISTS shopping_mall WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
cqlsh:shopping_mall>
cqlsh:shopping_mall> -- Use the keyspace
cqlsh:shopping_mall> USE shopping_mall;
cqlsh:shopping_mall>
cqlsh:shopping_mall> -- Create a table to store supplier data
cqlsh:shopping_mall> CREATE TABLE IF NOT EXISTS suppliers (
                 ...     supplier_id UUID PRIMARY KEY,
                 ...     name TEXT,
                 ...     email TEXT,
                 ...     phone TEXT,
                 ...     address TEXT
                 ... );
cqlsh:shopping_mall>
cqlsh:shopping_mall> -- Insert 5 supplier records
cqlsh:shopping_mall> INSERT INTO suppliers (supplier_id, name, email, phone, address) VALUES (uuid(), 'Supplier 1', 'supplier1@example.com', '123-456-7890', '123 Suppli
er St');
cqlsh:shopping_mall> INSERT INTO suppliers (supplier_id, name, email, phone, address) VALUES (uuid(), 'Supplier 2', 'supplier2@example.com', '456-789-0123', '456 Suppli
er St');
cqlsh:shopping_mall> INSERT INTO suppliers (supplier_id, name, email, phone, address) VALUES (uuid(), 'Supplier 3', 'supplier3@example.com', '789-012-3456', '789 Suppli
er St');
cqlsh:shopping_mall> INSERT INTO suppliers (supplier_id, name, email, phone, address) VALUES (uuid(), 'Supplier 4', 'supplier4@example.com', '012-345-6789', '012 Suppli
er St');
cqlsh:shopping_mall> INSERT INTO suppliers (supplier_id, name, email, phone, address) VALUES (uuid(), 'Supplier 5', 'supplier5@example.com', '345-678-9012', '345 Suppli
er St');
cqlsh:shopping_mall>
cqlsh:shopping_mall> -- Query the data to verify insertion
cqlsh:shopping_mall> SELECT * FROM suppliers;

 supplier_id                          | address         | email                 | name       | phone
--------------------------------------+-----------------+-----------------------+------------+--------------
 d5aa00d9-5642-4eb0-b6c9-59190a43f22b | 789 Supplier St | supplier3@example.com | Supplier 3 | 789-012-3456
 e4a6cd3f-09f4-4577-8822-668f61aa74af | 012 Supplier St | supplier4@example.com | Supplier 4 | 012-345-6789
 3a645acc-a98c-416f-a4af-e58c450150ea | 456 Supplier St | supplier2@example.com | Supplier 2 | 456-789-0123
 5b4beab8-712b-4276-a934-4f4d4c546255 | 123 Supplier St | supplier1@example.com | Supplier 1 | 123-456-7890
 27c48f00-c5a4-49f1-882f-a66b0339791a | 345 Supplier St | supplier5@example.com | Supplier 5 | 345-678-9012

(5 rows)
```

Step 4:

Below is the CQL code to create a table for products and insert 5 different product records into the Cassandra database. This code will create a table named products to store product data, and insert 5 different product records into the table. Finally, it selects and displays all the product data to verify that the insertion was successful.

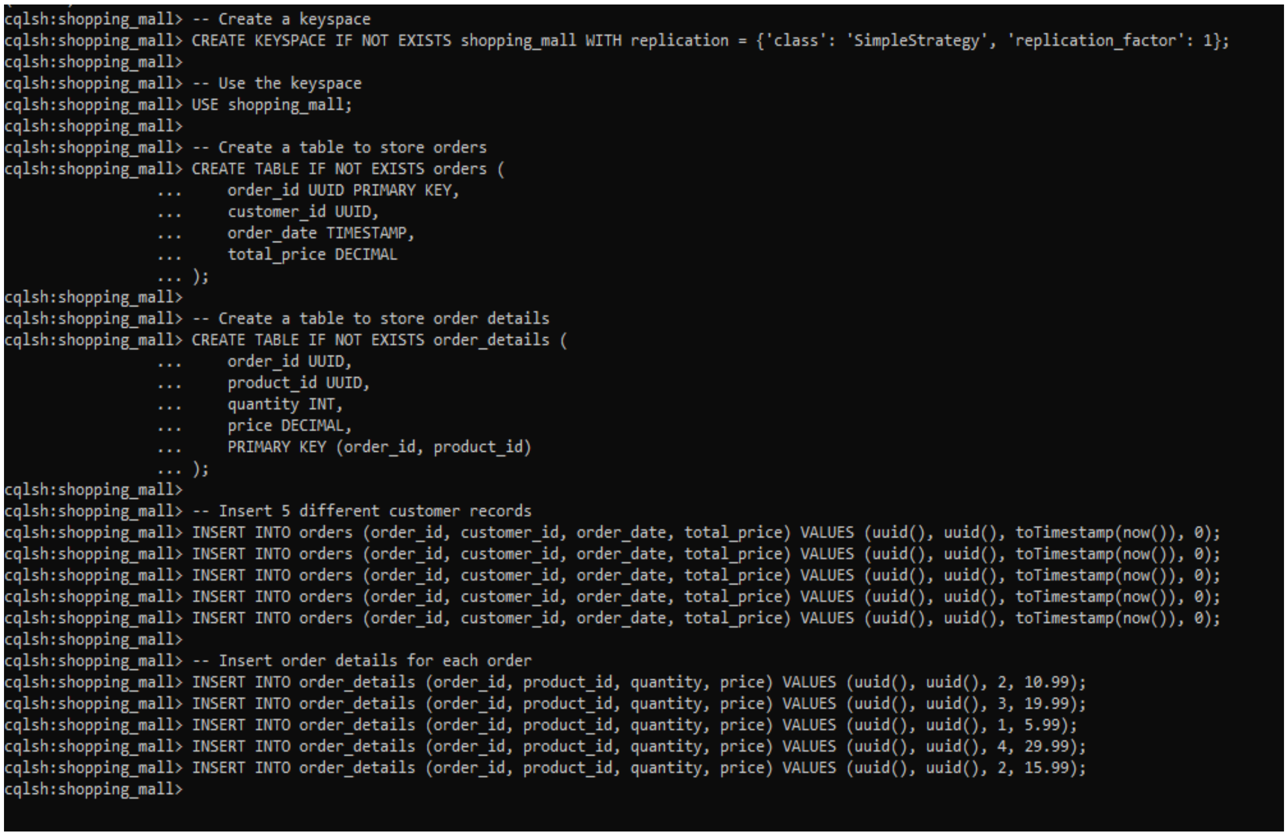


```
cqlsh:shopping_mall> -- Create a keyspace
cqlsh:shopping_mall> CREATE KEYSPACE IF NOT EXISTS shopping_mall WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
cqlsh:shopping_mall>
cqlsh:shopping_mall> -- Use the keyspace
cqlsh:shopping_mall> USE shopping_mall;
cqlsh:shopping_mall>
cqlsh:shopping_mall> -- Create a table to store product data
cqlsh:shopping_mall> CREATE TABLE IF NOT EXISTS products (
                 ...     product_id UUID PRIMARY KEY,
                 ...     name TEXT,
                 ...     description TEXT,
                 ...     price DECIMAL,
                 ...     quantity INT
                 ... );
cqlsh:shopping_mall>
cqlsh:shopping_mall> -- Insert 5 different product records
cqlsh:shopping_mall> INSERT INTO products (product_id, name, description, price, quantity) VALUES (uuid(), 'Product 1', 'Description of Product 1', 10.99, 100);
cqlsh:shopping_mall> INSERT INTO products (product_id, name, description, price, quantity) VALUES (uuid(), 'Product 2', 'Description of Product 2', 19.99, 50);
cqlsh:shopping_mall> INSERT INTO products (product_id, name, description, price, quantity) VALUES (uuid(), 'Product 3', 'Description of Product 3', 5.99, 200);
cqlsh:shopping_mall> INSERT INTO products (product_id, name, description, price, quantity) VALUES (uuid(), 'Product 4', 'Description of Product 4', 29.99, 75);
cqlsh:shopping_mall> INSERT INTO products (product_id, name, description, price, quantity) VALUES (uuid(), 'Product 5', 'Description of Product 5', 15.99, 150);
cqlsh:shopping_mall>
cqlsh:shopping_mall> -- Query the data to verify insertion
cqlsh:shopping_mall> SELECT * FROM products;

 product_id                           | description              | name      | price | quantity
--------------------------------------+--------------------------+-----------+-------+----------
 8c321e4d-150c-4156-acc8-e2bc5e432da9 | Description of Product 2 | Product 2 | 19.99 |       50
 529d8669-5b2d-4931-b6f2-76662d5d890c | Description of Product 1 | Product 1 | 10.99 |      100
 d2a1e32d-c7b2-46eb-ad31-ad7903f7fd41 | Description of Product 5 | Product 5 | 15.99 |      150
 324e6baa-75f5-4b34-a09d-f03de75433c5 | Description of Product 3 | Product 3 |  5.99 |      200
 111723b3-2d0b-41b7-af20-e16f8ebf1079 | Description of Product 4 | Product 4 | 29.99 |       75

(5 rows)
```

Step 5:

A scenario where 5 customers are ordering products from suppliers, we'll need to first define tables for orders and order details. Below is the CQL code to set up the scenario In this scenario, we create two tables: orders to store order information and order details to store details of each product ordered in an order. We then insert 5 orders into the orders table, each representing a different customer's order. Additionally, we insert order details for each order into the order details table.

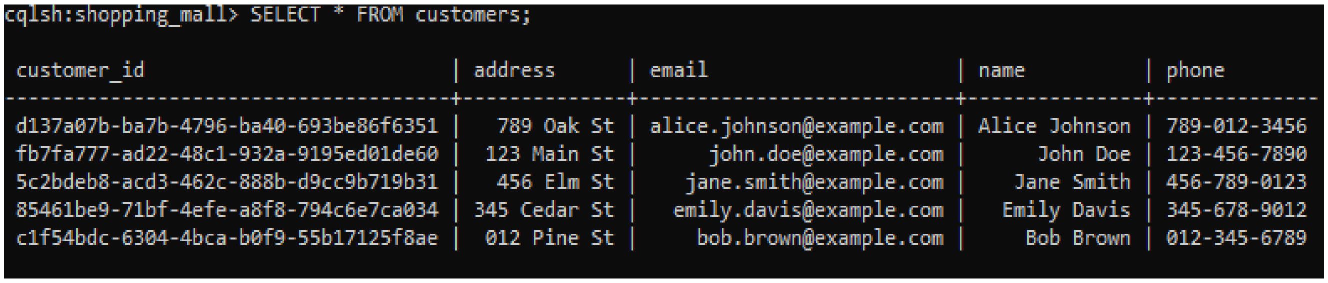
Register No.................... Date:................................



```
cqlsh:shopping_mall> -- Create a keyspace
cqlsh:shopping_mall> CREATE KEYSPACE IF NOT EXISTS shopping_mall WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
cqlsh:shopping_mall>
cqlsh:shopping_mall> -- Use the keyspace
cqlsh:shopping_mall> USE shopping_mall;
cqlsh:shopping_mall>
cqlsh:shopping_mall> -- Create a table to store orders
cqlsh:shopping_mall> CREATE TABLE IF NOT EXISTS orders (
                 ...     order_id UUID PRIMARY KEY,
                 ...     customer_id UUID,
                 ...     order_date TIMESTAMP,
                 ...     total_price DECIMAL
                 ... );
cqlsh:shopping_mall>
cqlsh:shopping_mall> -- Create a table to store order details
cqlsh:shopping_mall> CREATE TABLE IF NOT EXISTS order_details (
                 ...     order_id UUID,
                 ...     product_id UUID,
                 ...     quantity INT,
                 ...     price DECIMAL,
                 ...     PRIMARY KEY (order_id, product_id)
                 ... );
cqlsh:shopping_mall>
cqlsh:shopping_mall> -- Insert 5 different customer records
cqlsh:shopping_mall> INSERT INTO orders (order_id, customer_id, order_date, total_price) VALUES (uuid(), uuid(), toTimestamp(now()), 0);
cqlsh:shopping_mall> INSERT INTO orders (order_id, customer_id, order_date, total_price) VALUES (uuid(), uuid(), toTimestamp(now()), 0);
cqlsh:shopping_mall> INSERT INTO orders (order_id, customer_id, order_date, total_price) VALUES (uuid(), uuid(), toTimestamp(now()), 0);
cqlsh:shopping_mall> INSERT INTO orders (order_id, customer_id, order_date, total_price) VALUES (uuid(), uuid(), toTimestamp(now()), 0);
cqlsh:shopping_mall> INSERT INTO orders (order_id, customer_id, order_date, total_price) VALUES (uuid(), uuid(), toTimestamp(now()), 0);
cqlsh:shopping_mall>
cqlsh:shopping_mall> -- Insert order details for each order
cqlsh:shopping_mall> INSERT INTO order_details (order_id, product_id, quantity, price) VALUES (uuid(), uuid(), 2, 10.99);
cqlsh:shopping_mall> INSERT INTO order_details (order_id, product_id, quantity, price) VALUES (uuid(), uuid(), 3, 19.99);
cqlsh:shopping_mall> INSERT INTO order_details (order_id, product_id, quantity, price) VALUES (uuid(), uuid(), 1, 5.99);
cqlsh:shopping_mall> INSERT INTO order_details (order_id, product_id, quantity, price) VALUES (uuid(), uuid(), 4, 29.99);
cqlsh:shopping_mall> INSERT INTO order_details (order_id, product_id, quantity, price) VALUES (uuid(), uuid(), 2, 15.99);
cqlsh:shopping_mall>
```

Sample Queries

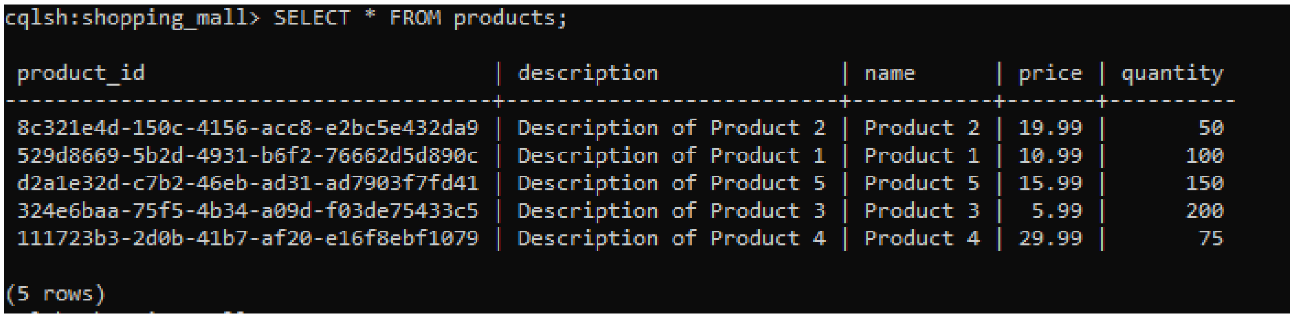
1. Query to Retrieve All Customers and its output



```
cqlsh:shopping_mall> SELECT * FROM customers;

 customer_id                          | address      | email                     | name          | phone
--------------------------------------+--------------+---------------------------+---------------+--------------
 d137a07b-ba7b-4796-ba40-693be86f6351 |   789 Oak St | alice.johnson@example.com | Alice Johnson | 789-012-3456
 fb7fa777-ad22-48c1-932a-9195ed01de60 |  123 Main St |      john.doe@example.com |      John Doe | 123-456-7890
 5c2bdeb8-acd3-462c-888b-d9cc9b719b31 |   456 Elm St |    jane.smith@example.com |    Jane Smith | 456-789-0123
 85461be9-71bf-4efe-a8f8-794c6e7ca034 | 345 Cedar St |   emily.davis@example.com |   Emily Davis | 345-678-9012
 c1f54bdc-6304-4bca-b0f9-55b17125f8ae |  012 Pine St |     bob.brown@example.com |     Bob Brown | 012-345-6789
```

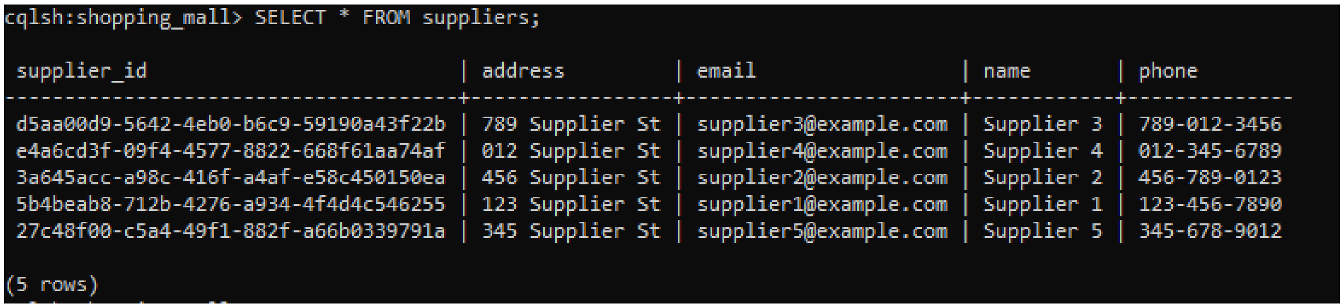
2. Query to Retrieve All Products and its output

Register No.....................   Date:................................



```
cqlsh:shopping_mall> SELECT * FROM products;

 product_id                           | description              | name      | price | quantity
--------------------------------------+--------------------------+-----------+-------+----------
 8c321e4d-150c-4156-acc8-e2bc5e432da9 | Description of Product 2 | Product 2 | 19.99 |       50
 529d8669-5b2d-4931-b6f2-76662d5d890c | Description of Product 1 | Product 1 | 10.99 |      100
 d2a1e32d-c7b2-46eb-ad31-ad7903f7fd41 | Description of Product 5 | Product 5 | 15.99 |      150
 324e6baa-75f5-4b34-a09d-f03de75433c5 | Description of Product 3 | Product 3 |  5.99 |      200
 111723b3-2d0b-41b7-af20-e16f8ebf1079 | Description of Product 4 | Product 4 | 29.99 |       75

(5 rows)
```

3. Query to Retrieve All Suppliers and its output:



```
cqlsh:shopping_mall> SELECT * FROM suppliers;

 supplier_id                          | address        | email                 | name       | phone
--------------------------------------+----------------+-----------------------+------------+--------------
 d5aa00d9-5642-4eb0-b6c9-59190a43f22b | 789 Supplier St | supplier3@example.com | Supplier 3 | 789-012-3456
 e4a6cd3f-09f4-4577-8822-668f61aa74af | 012 Supplier St | supplier4@example.com | Supplier 4 | 012-345-6789
 3a645acc-a98c-416f-a4af-e58c450150ea | 456 Supplier St | supplier2@example.com | Supplier 2 | 456-789-0123
 5b4beab8-712b-4276-a934-4f4d4c546255 | 123 Supplier St | supplier1@example.com | Supplier 1 | 123-456-7890
 27c48f00-c5a4-49f1-882f-a66b0339791a | 345 Supplier St | supplier5@example.com | Supplier 5 | 345-678-9012

(5 rows)
```