VISHNU INSTITUTE OF TECHNOLOGY

(AUTONOMOUS)

VISHNUPUR: BHIMAVARAM

WEST.GODAVARI– District, ANDHRA PRADESH

CERTIF ICATE

Certified that this is a bonafide record of practical work done by

Mr. / Ms. ..... Roll No .....of

III B.Tech - II SEM in the NoSQL Databases Laboratory of Artificial Intelligence and

Data Science Department during the year 2022 – 2023

Date Head of the Department In-charge Faculty

Submitted for the Practical Examination held on .....

INTERNAL EXAMINER EXTERNAL EXAMINER

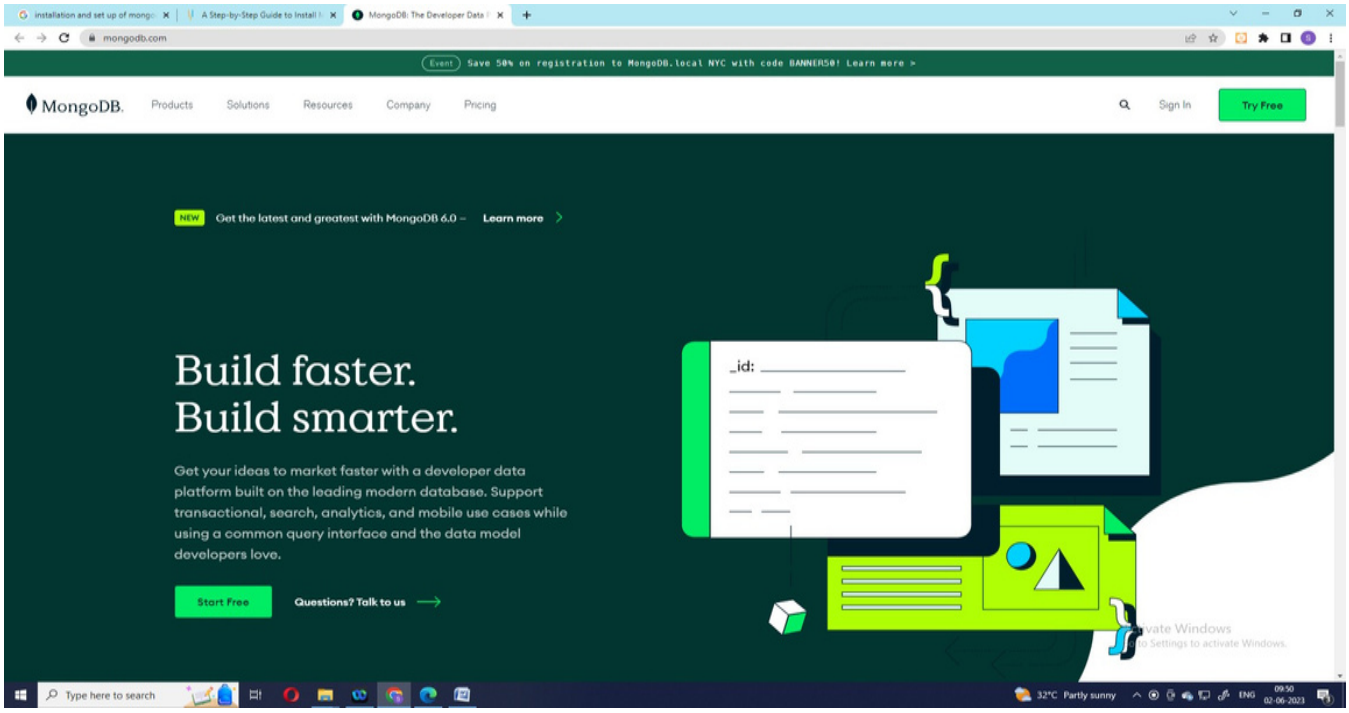
INDEX

Ex. No: 1 Date:	Installation and set up of MongoDB client and server

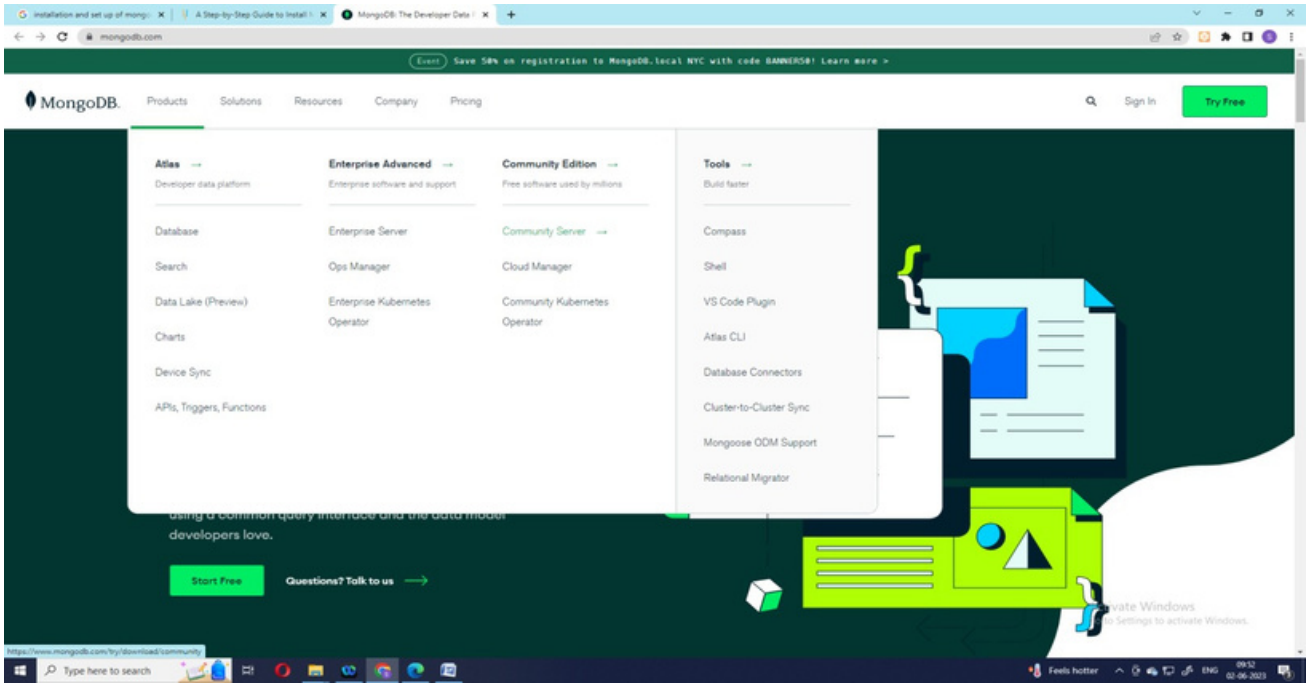
AIM: Installation and set up of MongoDB client and server

STEPS:

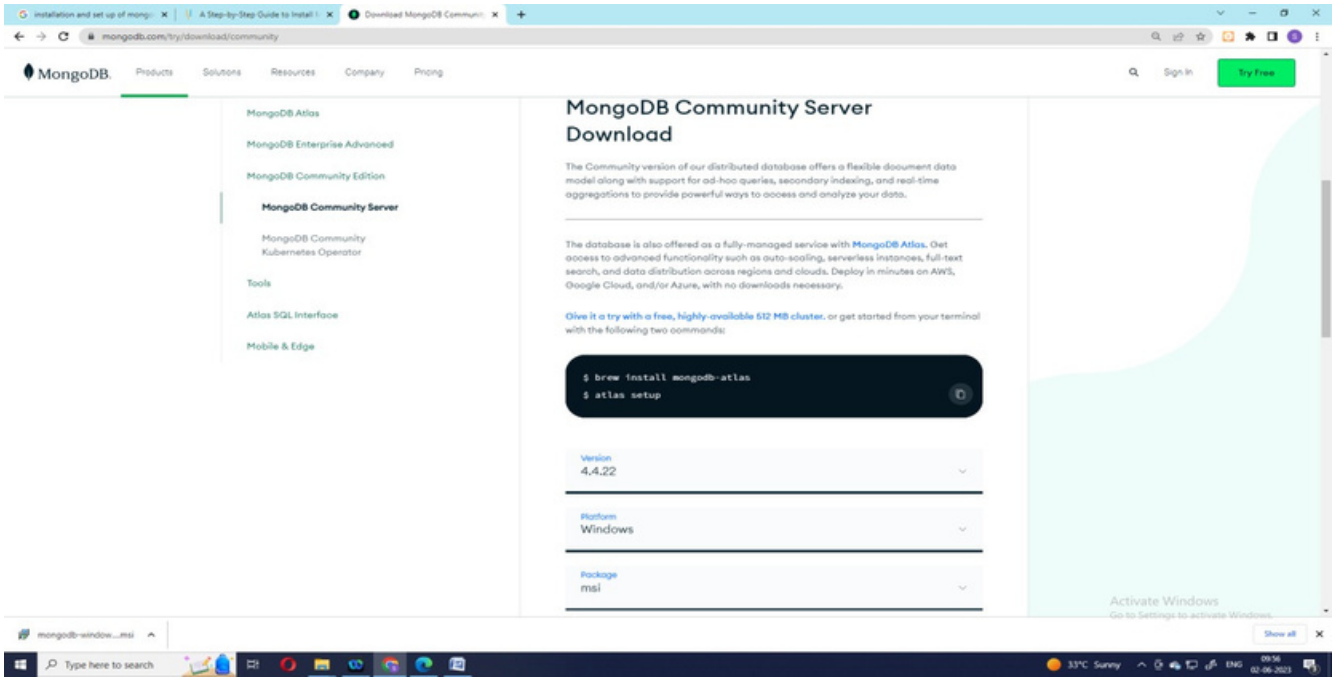
1. Navigate to the official MongoDB website - https://www.mongodb.com



2. Under the Products section, click on the Community server version.

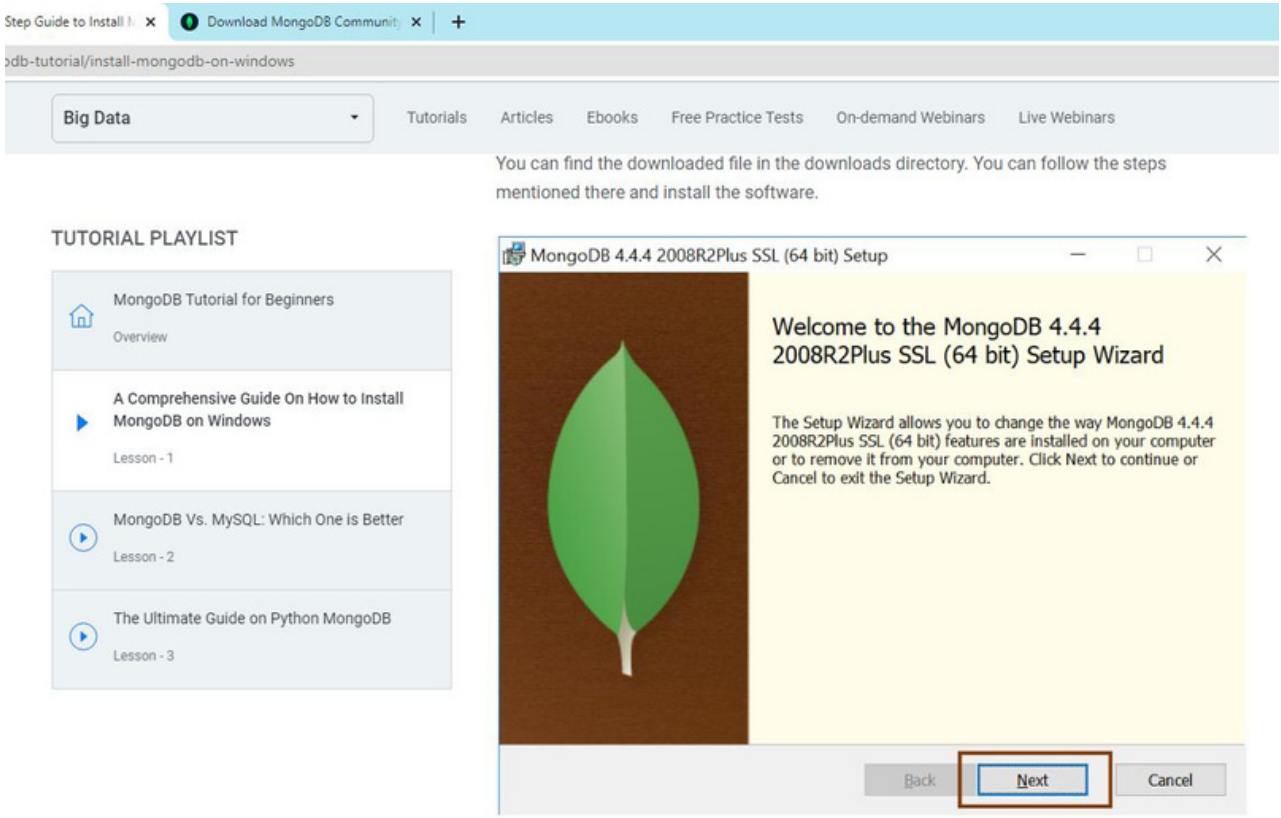


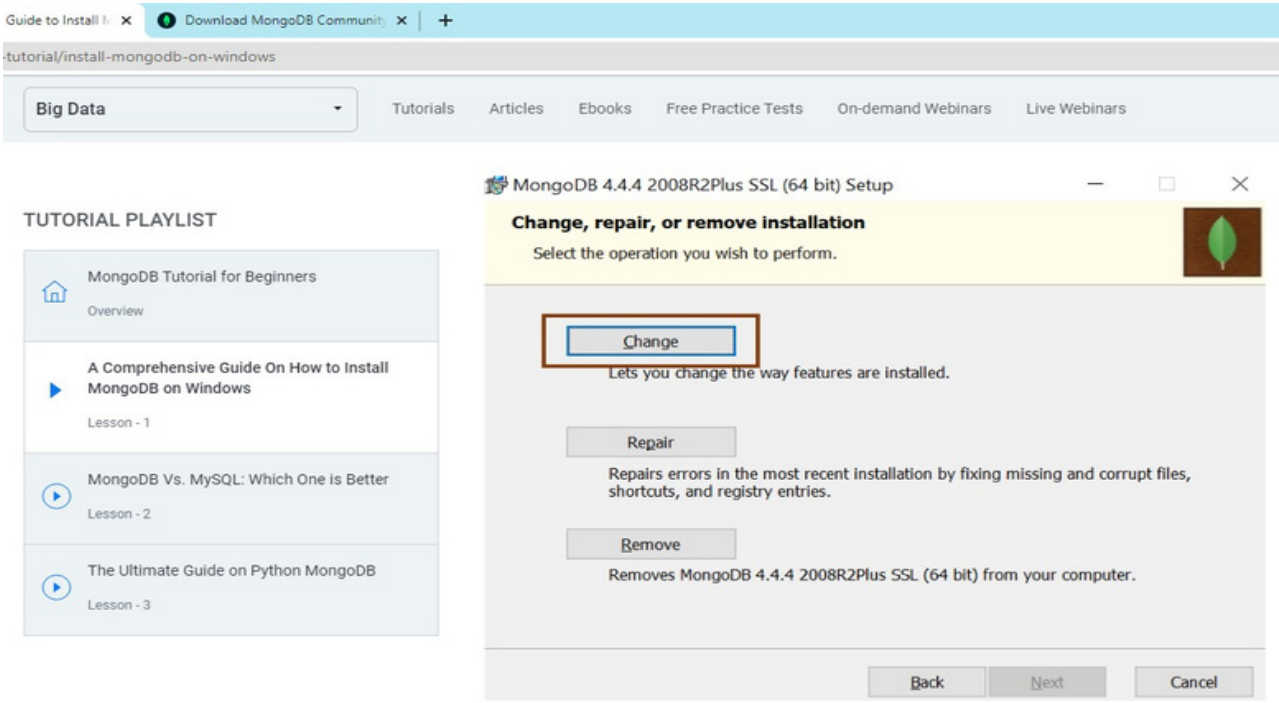
3. Make sure that the specifications to the right of the screen are correct. At the time of writing, the latest version is 4.4.22. Ensure that the platform is Windows, and the package is MSI. Go ahead and click on download.

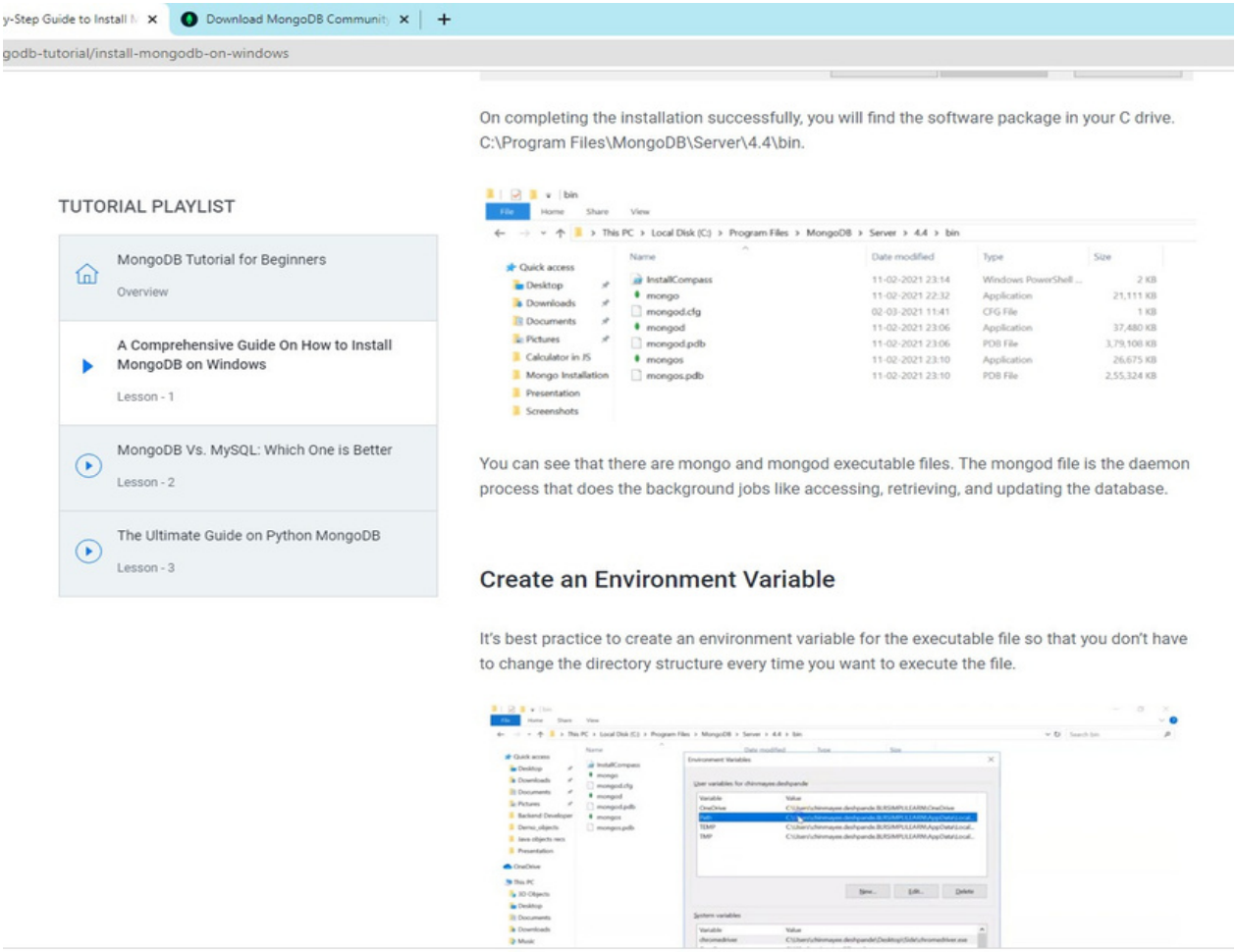


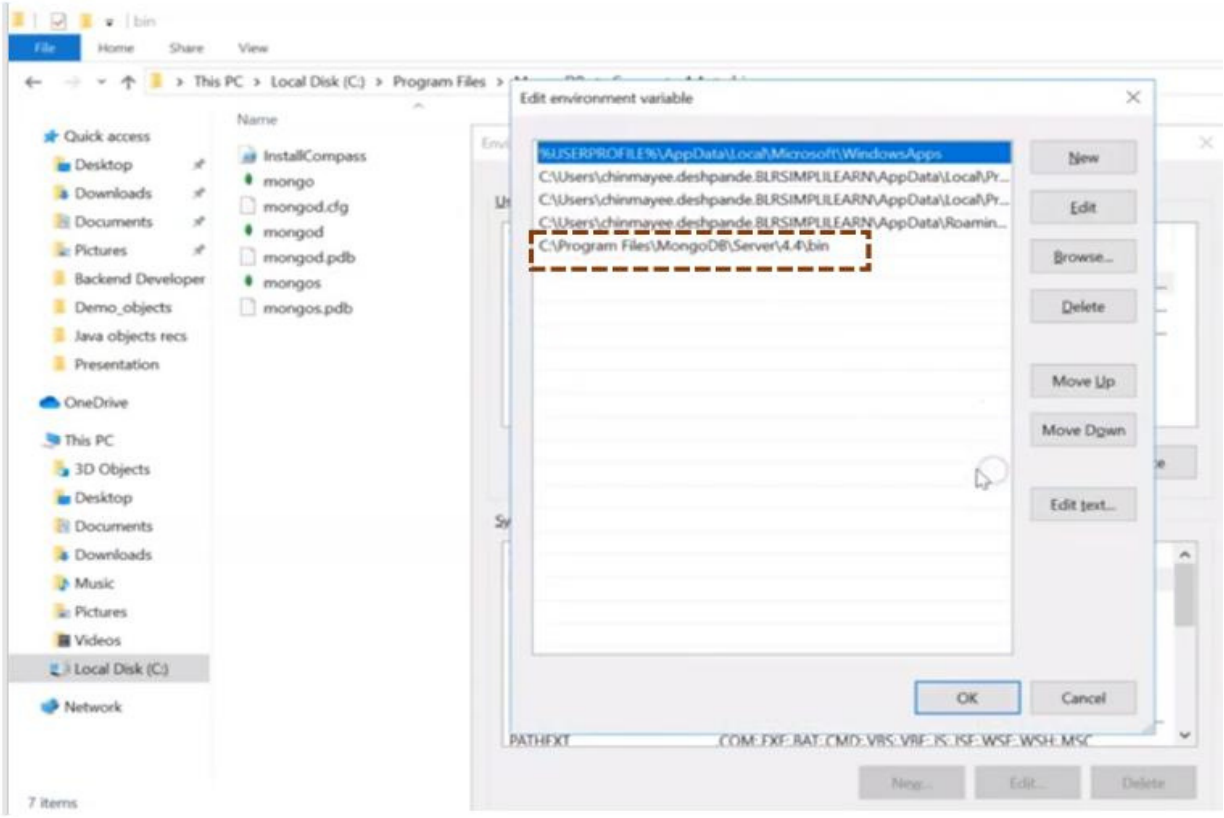
4. MongoDB Installation

You can find the downloaded file in the downloads directory. You can follow the steps mentioned there and install the software.




Guide to Install M
Download MongoDB Community
-tutorial/install-mongodb-on-windows
Big Data
Tutorials
Articles
Ebooks
Free Practice Tests
On-demand Webinars
Live Webinars
TUTORIAL PLAYLIST
MongoDB Tutorial for Beginners
Overview
A Comprehensive Guide On How to Install MongoDB on Windows
Lesson - 1
MongoDB Vs. MySQL: Which One is Better
Lesson - 2
The Ultimate Guide on Python MongoDB
Lesson - 3
MongoDB 4.4.4 2008R2Plus SSL (64 bit) Setup
Change, repair, or remove installation
Select the operation you wish to perform.
Change
Lets you change the way features are installed.
Repair
Repairs errors in the most recent installation by fixing missing and corrupt files, shortcuts, and registry entries.
Remove
Removes MongoDB 4.4.4 2008R2Plus SSL (64 bit) from your computer.
Back
Next
Cancel


y-Step Guide to Install M
Download MongoDB Community
godb-tutorial/install-mongodb-on-windows
On completing the installation successfully, you will find the software package in your C drive. C:\Program Files\MongoDB\Server\4.4\bin.
TUTORIAL PLAYLIST
MongoDB Tutorial for Beginners
Overview
A Comprehensive Guide On How to Install MongoDB on Windows
Lesson - 1
MongoDB Vs. MySQL: Which One is Better
Lesson - 2
The Ultimate Guide on Python MongoDB
Lesson - 3
bin
This PC > Local Disk (C:) > Program Files > MongoDB > Server > 4.4 > bin
InstallCompass
mongo
mongod.cfg
mongod
mongod.pdb
mongos
mongos.pdb
You can see that there are mongo and mongod executable files. The mongod file is the daemon process that does the background jobs like accessing, retrieving, and updating the database.
Create an Environment Variable
It's best practice to create an environment variable for the executable file so that you don't have to change the directory structure every time you want to execute the file.



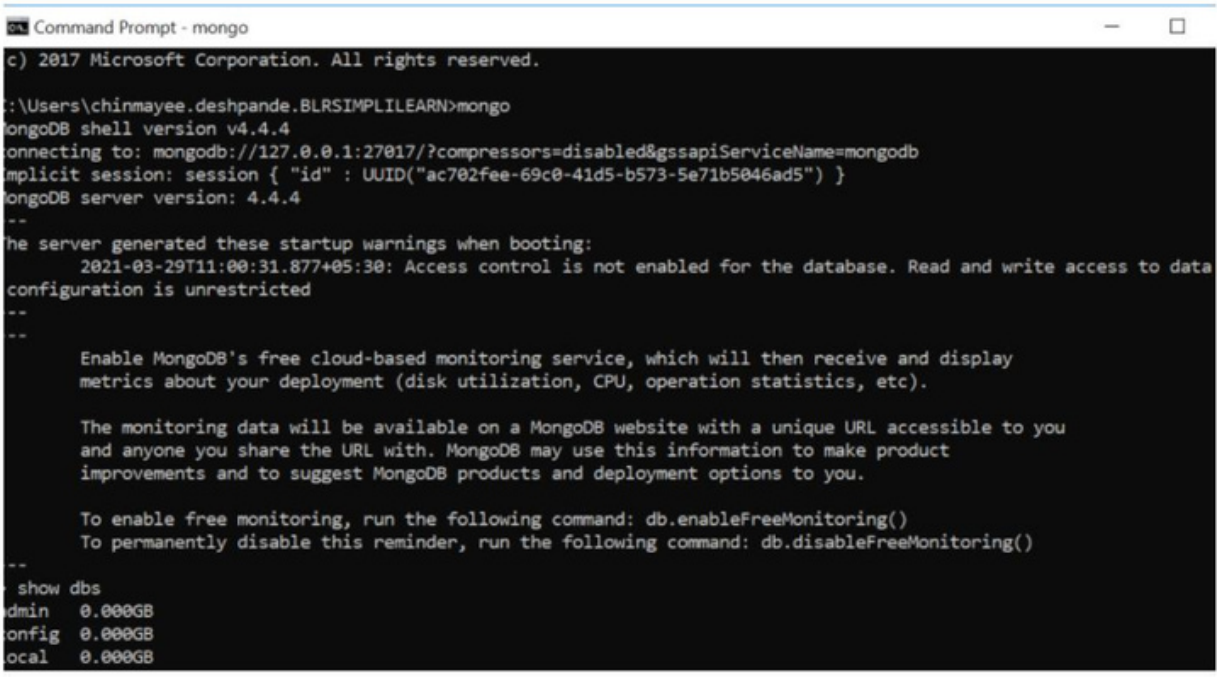
5. Execute the Mongo App

After creating an environment path, you can open the command prompt and just type in mongo and press enter.

```
Command Prompt - mongo
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\chinmayee.deshpande.BLRSIMPLILEARN>mongo
MongoDB shell version v4.4.4
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("ac702fee-69c0-41d5-b573-5e71b5046ad5") }
MongoDB server version: 4.4.4
---
The server generated these startup warnings when booting:
        2021-03-29T11:00:31.877+05:30: Access control is not enabled for the database. Read and write access to data
 configuration is unrestricted
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

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

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

6. Verify the Step

To verify if it did the setup correctly, type in the command show DBS.



This demo sample has also created a database called mydatabase, with some data added to it. It has also displayed the same using the find() method.

Ex. No: 2 Create a database and collection using MongoDB environment.	
Date:	

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

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

Database Name: Restaurant

Collection Name: Restaurant_Records

> use **Restaurant**

switched to db Restaurant

1. Insert Document:

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

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

restaurant_id: "AEPY06",
restaurant_name: "do it yourself",
customer_name:"abhyash",
locality:"pune",
date:03/03/2023,
cuisine:"korean",
grade:"5 star",
comments: "Good"
})

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

restaurant_id: "AEPY07",
restaurant_name: "Spice Magic",
customer_name:"Venkat",
locality:"Mumbai",
date:03/05/2022,
cuisine:"Chinese",
grade:"4 star",
comments: "Excellent"
})

>db.**Restaurant_Records.insertMany**([
{
restaurant_id: "AEPY08",
restaurant_name: "Adurs",
customer_name:"Prabhu",
locality:"Vijayawada",
date:03/05/2021,
cuisine:"Indian",
grade:"4 star",
comments: "Excellent"
},
{
restaurant_id: "AEPY09",
restaurant_name: "chandrika",
customer_name:"Ram",
locality:"Bhimavaram",
date:03/03/2022,
cuisine:"Indian",
grade:"5 star",
comments: "Excellent"
}
]);

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

> db.Restaurant_Records.find();

Output:

{ "_id" : ObjectId("64797bc50655b1cad1d7789d"), "restaurant_id" : "AEPY06", "restaurant_name" : "do it yourself", "customer_name" : "abhyash", "locality" : "pune", "date" : 0.0004943153732081067, "cuisine" : "korean", "grade" : "5 star", "comments" : "Good" }
{ "_id" : ObjectId("64797bc50655b1cad1d7789e"), "restaurant_id" : "AEPY07", "restaurant_name" : "Spice Magic", "customer_name" : "Venkat", "locality" : "Mumbai", "date" : 0.0002967359050445104, "cuisine" : "Chinese", "grade" : "4 star", "comments" : "Excellent" }
{ "_id" : ObjectId("64797bc50655b1cad1d7789f"), "restaurant_id" : "AEPY08", "restaurant_name" : "Adurs", "customer_name" : "Prabhu", "locality" : "Vijayawada", "date" : 0.00029688273132112816, "cuisine" : "Indian", "grade" : "4 star", "comments" : "Excellent" }
{ "_id" : ObjectId("64797bc50655b1cad1d778a0"), "restaurant_id" : "AEPY09", "restaurant_name" : "chandrika", "customer_name" : "Ram", "locality" : "Bhimavaram", "date" : 0.0004945598417408506, "cuisine" : "Indian", "grade" : "5 star", "comments" : "Excellent" }

2. Updating documents

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

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

Output:

>db.Restaurant_Records.find();

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
{ "_id" : ObjectId("64797e29f81746b9c76c7c5d"), "restaurant_id" : "AEPY06", "restaurant_name" : "do it yourself", "customer_name" : "abhyash", "locality" : "pune", "date" : 0.0004943153732081067, "cuisine" : "korean", "grade" : "5 star", "comments" : "Good" }
{ "_id" : ObjectId("64797e29f81746b9c76c7c5e"), "restaurant_id" : "AEPY07", "restaurant_name" : "Salem Briyani Restaurant", "customer_name" : "Venkat", "locality" : "Mumbai", "date" : 0.0002967359050445104, "cuisine" : "Chinese", "grade" : "4 star", "comments" : "Excellent" }
{ "_id" : ObjectId("64797e29f81746b9c76c7c5f"), "restaurant_id" : "AEPY08", "restaurant_name" : "Adurs", "customer_name" : "Prabhu", "locality" : "Vijayawada", "date" : 0.0002968827313211 2816, "cuisine" : "Indian", "grade" : "4 star", "comments" : "Excellent" }
{ "_id" : ObjectId("64797e29f81746b9c76c7c60"), "restaurant_id" : "AEPY09", "restaurant_name" : "chandrika", "customer_name" : "Ram", "locality" : "Bhimavaram", "date" : 0.0004945598417408506, "cuisine" : "Indian", "grade" : "5 star", "comments" : "Excellent" }

3. **Upsert with Operator Expressions:**

If no document matches the filter from the given collection and the update parameter is a document that contains update operators, also the value of upsert option is set to true, then the update operation creates new documents from the equality clauses in the given query parameter and applies the expressions from the update parameter. Or in other words, when the value of upsert option is true and no document matches the given filter, then the update operation inserts a new document in the given collection, and the fields inserted in this new document are the fields that specify in the query and update documents.

Now we are going to insert a new document in the example collection by setting the value of the upsert option to true.

db.Restaurant_Records.update ({'restaurant_name': "Suprabath"},
{$set: {'restaurant_id' : "AEPY11", 'customer_name' : "Ramesh", 'locality' : "Bhimavaram ",

'date' : 03/03/2021, 'grade' : "5 star", 'comments' : "Excellent" },
$setOnInsert: {"cuisine" : "Indian"}},
{upsert: true})
db.Restaurant_Records.find();
Output:
>db.Restaurant_Records.find();

WriteResult({
"nMatched" : 0,
"nUpserted" : 1,
"nModified" : 0,
"_id" : ObjectId("6479810f66d73441ef0a44ae")
})
{ "_id" : ObjectId("6479810f93e57a265ca84438"), "restaurant_id" : "AEPY06", "restaurant_name" : "do it yourself", "customer_name" : "abhyash", "locality" : "pune", "date" : 0.0004943153732081067, "cuisine" : "korean", "grade" : "5 star", "comments" : "Good" }
{ "_id" : ObjectId("6479810f93e57a265ca84439"), "restaurant_id" : "AEPY07", "restaurant_name" : "Salem Briyani Restaurant", "customer_name" : "Venkat", "locality" : "Mumbai", "date" : 0.0002967359050445104, "cuisine" : "Chinese", "grade" : "4 star", "comments" : "Excellent" }
{ "_id" : ObjectId("6479810f93e57a265ca8443a"), "restaurant_id" : "AEPY08", "restaurant_name" : "Adurs", "customer_name" : "Prabhu", "locality" : "Vijayawada", "date" : 0.00029688273132112816, "cuisine" : "Indian", "grade" : "4 star", "comments" : "Excellent" }
{ "_id" : ObjectId("6479810f93e57a265ca8443b"), "restaurant_id" : "AEPY09", "restaurant_name" : "chandrika", "customer_name" : "Ram", "locality" : "Bhimavaram", "date" : 0.0004945598417408506, "cuisine" : "Indian", "grade" : "5 star", "comments" : "Excellent" }
{ "_id" : ObjectId("6479810f66d73441ef0a44ae"), "restaurant_name" : "Suprabath", "comments" : "Excellent", "cuisine" : "Indian", "customer_name" : "Ramesh", "date" : 0.0004948045522018803, "grade" : "5 star", "locality" : "Bhimavaram ", "restaurant_id" : "AEPY11" }

4. Deleting documents

Deletes a Single document or all documents from collection

>db.Restaurant_Records.deleteOne({"restaurant_id" : "AEPY09", })

Output:

>db.Restaurant_Records.find();

{ "acknowledged" : true, "deletedCount" : 1 }
{ "_id" : ObjectId("647981eee4e3610a37abe130"), "restaurant_id" : "AEPY06", "restaurant_name" : "do it yourself", "customer_name" : "abhyash", "locality" : "pune", "date" : 0.0004943153732081067, "cuisine" : "korean", "grade" : "5 star", "comments" : "Good" }
{ "_id" : ObjectId("647981eee4e3610a37abe131"), "restaurant_id" : "AEPY07", "restaurant_name" : "Salem Briyani Restaurant", "customer_name" : "Venkat", "locality" : "Mumbai", "date" : 0.0002967359050445104, "cuisine" : "Chinese", "grade" : "4 star", "comments" : "Excellent" }
{ "_id" : ObjectId("647981eee4e3610a37abe132"), "restaurant_id" : "AEPY08", "restaurant_name" : "Adurs", "customer_name" : "Prabhu", "locality" : "Vijayawada", "date" : 0.00029688273132112816, "cuisine" : "Indian", "grade" : "4 star", "comments" : "Excellent" }
{ "_id" : ObjectId("647981ee646ec19b5ed783ac"), "restaurant_name" : "Suprabath", "comments" : "Excellent", "cuisine" : "Indian", "customer_name" : "Ramesh", "date" : 0.0004948045522018803, "grade" : "5 star", "locality" : "Bhimavaram ", "restaurant_id" : "AEPY11" }

Indexes in MongoDB

Indexes in MongoDB let you prioritize some fields in a document and turn them into query parameters later on. When creating a collection, MongoDB creates a default ID index. But you can add more if you have higher queries or arrangement needs.

Unique indexes also prevent duplicates during data entry. They come in handy for rejecting entries that already exist in the database. So you can use unique indexing on usernames and email fields, for instance.

Indexes help you grab specifically what you want using your defined query format at the required quantity without scanning an entire collection. When creating an index, you can specify how you want your data sorted whenever you query it.

For instance, if you decide to sort data points by their entry dates without using an index, you'll typically provide query criteria and sort order.

db.collectionName.find({age: {$gt: 50}}).sort({date: -1})

But if you had an index on the entry dates, you'd only need to provide the query criteria. This is so because you'd have already sorted the date descendingly while creating the index.

In that case, the above query becomes:

db.collectionName.find({age: {$gt: 50}})

How to Create an Index in MongoDB

Nevertheless, indexes don't exist alone. You must've created a database and a collection before you can create an index.

For example, an index that arranges age in a reverse order looks like this:

db.userCollection.createIndex({age: -1})

The negative integer (-1) in the code tells MongoDB to arrange age in a reverse order whenever you query data using the age index. Thus, you might not need to specify a sorting order during queries since the index handles that already.

To create an index of the same field ascendingly, replace -1 with 1:

db.userCollection.createIndex({age: 1})

Although there are other index types in MongoDB, this is how to create the single and compound forms—since they're the most used.

Create a Single Index in MongoDB

Single indexing in MongoDB is straightforward. To start, select a database that already contains collections.

Say our database name is MUO:

use MUO

Once you choose a database, use the createIndex() command in the shell to create a single index.

For example, to create a single username index in a collection and use it to query in reverse order:

db.collectionName.createIndex({username: -1})

Multikey Indexing in MongoDB

Multikey indexes come in handy for indexing a field in a complex data array. For instance, a collection might contain complex data of users with their information in another array. For example, say name, height, and age.

You can create a multikey index on the height of each user in that array like this:

db.customers.createIndex({user.height: 1})

The height in the above code is a subset of the user field.

Create a Compound Index

A compound index contains more than one index. To create a compound index of address and product type on a customer collection, for instance:

db.customer.createIndex({address: 1, products: 1})

Since you didn't provide a name while creating the above index, MongoDB creates one by default. But it separates the name of each index by an underscore. So it's less readable, especially if you have more than two indexes in a compound.

To specify a name when creating a compound index:

db.customers.createIndex({location: 1, products: 1, weight: -1}, {name: "myCompundIndex"})

Get All Indexes in a Collection

To view all indexes in a collection:

db.collectionName.getIndexes()

The above code outputs all the indexes in a named collection.

Ex. No: 3 **Date:**	Perform Simple MongoDB queries such as displaying all the records, display selected records with conditions

Aim:

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

Program:

Insert Document

```
>db.Restaurant_Records.insertMany([{
restaurant_id: "AEPY07",
restaurant_name: "Spice Magic",
customer_name:"Venkat",
locality:"Mumbai",
date:03/05/2022,
cuisine:"Chinese",
grade:"4 star",
comments: "Excellent"
},
{
        restaurant_id: "AEPY08",
        restaurant_name: "Adurs",
        customer_name:"Prabhu",
        locality:"Vijayawada",
        date:03/05/2021,
        cuisine:"Indian",
},      grade:"4 star",
{       comments: "Excellent"

        restaurant_id: "AEPY09",
        restaurant_name: "chandrika",
        customer_name:"Ram",
        locality:"Bhimavaram",
        date:03/03/2022,
        cuisine:"Indian",
},      grade:"5 star",
{       comments: "Excellent"

        restaurant_id: "AEPY06",
        restaurant_name: "do it yourself",
        customer_name:"abhyash",
        locality:"pune",
```

date:03/03/2023,
cuisine:"korean",
grade:"5 star",
comments: "Good"
}
])

Queries

1. Write a MongoDB query to display all the documents in the collection restaurants.
>db.restaurants.find();

Output:

{
"acknowledged" : true,
"insertedIds" : [
ObjectId("647998c388f1b5a8bbfe7efc"),
ObjectId("647998c388f1b5a8bbfe7efd"),
ObjectId("647998c388f1b5a8bbfe7efe"),
ObjectId("647998c388f1b5a8bbfe7eff")
]
}
{ "_id" : ObjectId("647998c388f1b5a8bbfe7efc"), "restaurant_id" : "AEPY07",
"restaurant_name" : "Spice Magic", "customer_name" : "Venkat", "locality" : "Mumbai", "date" :
0.0002967359050445104, "cuisine" : "Chinese", "grade" : "4 star", "comments" : "Excellent" }
{ "_id" : ObjectId("647998c388f1b5a8bbfe7efd"), "restaurant_id" : "AEPY08",
"restaurant_name" : "Adurs", "customer_name" : "Prabhu", "locality" : "Vijayawada", "date" :
0.00029688273132112816, "cuisine" : "Indian", "grade" : "4 star", "comments" : "Excellent" }
{ "_id" : ObjectId("647998c388f1b5a8bbfe7efe"), "restaurant_id" : "AEPY09",
"restaurant_name" : "chandrika", "customer_name" : "Ram", "locality" : "Bhimavaram", "date" :
0.0004945598417408506, "cuisine" : "Indian", "grade" : "5 star", "comments" : "Excellent" }
{ "_id" : ObjectId("647998c388f1b5a8bbfe7eff"), "restaurant_id" : "AEPY06",
"restaurant_name" : "do it yourself", "customer_name" : "abhyash", "locality" : "pune", "date" :
0.0004943153732081067, "cuisine" : "korean", "grade" : "5 star", "comments" : "Good" }

2. Write a MongoDB query to display the fields restaurant_id, restaurant_name, and cuisine for all the documents in the collection restaurant.
> db.Restaurant_Records.find({},{"restaurant_id" : 1,"restaurant_name":1,"cuisine" :1});

Output:

{
"acknowledged" : true,
"insertedIds" : [
ObjectId("647999dc5f862d472d99f416"),
ObjectId("647999dc5f862d472d99f417"),
ObjectId("647999dc5f862d472d99f418"),

ObjectId("647999dc5f862d472d99f419")
]
}
{ "_id" : ObjectId("647999dc5f862d472d99f416"), "restaurant_id" : "AEPY07",
"restaurant_name" : "Spice Magic", "cuisine" : "Chinese" }
{ "_id" : ObjectId("647999dc5f862d472d99f417"), "restaurant_id" : "AEPY08",
"restaurant_name" : "Adurs", "cuisine" : "Indian" }
{ "_id" : ObjectId("647999dc5f862d472d99f418"), "restaurant_id" : "AEPY09",
"restaurant_name" : "chandrika", "cuisine" : "Indian" }
{ "_id" : ObjectId("647999dc5f862d472d99f419"), "restaurant_id" : "AEPY06",
"restaurant_name" : "do it yourself", "cuisine" : "korean" }

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

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

Output:

{
"acknowledged" : true,
"insertedIds" : [
ObjectId("64799aeb1ff0d3240df31074"),
ObjectId("64799aeb1ff0d3240df31075"),
ObjectId("64799aeb1ff0d3240df31076"),
ObjectId("64799aeb1ff0d3240df31077")
]
}
{ "_id" : ObjectId("64799aeb1ff0d3240df31075"), "restaurant_id" : "AEPY08",
"restaurant_name" : "Adurs", "customer_name" : "Prabhu", "locality" : "Vijayawada", "date" :
0.00029688273132112816, "cuisine" : "Indian", "grade" : "4 star", "comments" : "Excellent" }
{ "_id" : ObjectId("64799aeb1ff0d3240df31076"), "restaurant_id" : "AEPY09",
"restaurant_name" : "chandrika", "customer_name" : "Ram", "locality" : "Bhimavaram", "date" :
0.0004945598417408506, "cuisine" : "Indian", "grade" : "5 star", "comments" : "Excellent" }

Ex. No: 4 Experiment with MongoDB comparison and logical query operators - $gt, $gte, $lt, $lte, $in, #nin, $ne, $and, $or, $not Date:

Aim:

To Experiment with MongoDB comparison and logical query operators

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

Operators	Description
$eq	It is used to match the values of the fields that are equal to a specified value.
$ne	It is used to match all values of the field that are not equal to a specified value.
$gt	It is used to match values of the fields that are greater than a specified value.
$gte	It is used to match values of the fields that are greater than equal to the specified value.
$lt	It is used to match values of the fields that are less than a specified valueo
$lte	It is used to match values of the fields that are less than equals to the specified value
$in	It is used to match any of the values specified in an array.
$nin	It is used to match none of the values specified in an array.

In the following examples, we are working with:
Database: Sample
Collection: contributor
Document: three documents that contain the details of the contributors in the form of field-value pairs.

```
>use Sample
Switched to db Sample

>db.contributor.find()
{
"_id":Objectid("5e6f7a6692e6dfa3fc48ddbe"),
"name":"Rohit",
"branch":"CSE",
"joiningYear":2018,
"language":["C#","Python","Java"],
```

"personal":{"contactinfo":0,"state":"Delhi","age":24,"semesterMarks":[70,73.3,76.5,78.6] },
"salary":1000
}
{
"_id":Objectid("5e7b9f0a92e6dfa3fc48ddbf"),
"name":"Amit",
"branch""ECE",
"joiningYear":2017,
"language":["Python","C#"],
"personal":{"contactinfo":234556789,"state":"UP","age":25,"semesterMarks":[80,80.1,98,70 },
"salary":10000
}
{
"_id":Objectid("5e7b9f0a92e6dfa3fc48ddc0"),
"name":"Sumit",
"branch":"CSE",
"joiningYear":2017,
"language":["Java","Perl"],
"personal":{"contactinfo":2300056789,"state":"MP","age":24,"semesterMarks":[89,80.1,78,71]},
"salary":15000
}

1. Matching values using $nin operator:

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

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

{
"_id":Objectid("5e6f7a6692e6dfa3fc48ddbe"),
"name":"Rohit",
"branch":"CSE",
"joiningYear":2018,
"language":["C#","Python","Java"],
"personal":{"contactinfo":0,"state":"Delhi","age":24,"semesterMarks":[70,73.3,76.5,78.6] },
"salary":1000
}
{
"_id":Objectid("5e7b9f0a92e6dfa3fc48ddc0"),
"name":"Sumit",
"branch":"CSE",
"joiningYear":2017,
"language":["Java","Perl"],
"personal":{"contactinfo":2300056789,"state":"MP","age":24,"semesterMarks":[89,80.1,78,71]},
"salary":15000
}

2. Matching values using $in operator:

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

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

{
"_id":Objectid("5e7b9f0a92e6dfa3fc48ddbf"),
"name":"Amit",
"branch""ECE",
"joiningYear":2017,
"language":["Python","C#"],
"personal":{"contactinfo":234556789,"state":"UP","age":25,"semesterMarks":[80,80.1,98,70 },
"salary":10000
}

3. Matching values using $lt operator:

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

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

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

4. Matching values using $eq operator:

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

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

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

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

5. Matching values using $ne operator:

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

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

{
"_id":Objectid("5e7b9f0a92e6dfa3fc48ddbf"),
"name":"Amit",
"branch""ECE",
"joiningYear":2017,
"language":["Python","C#"],
"personal":{"contactinfo":234556789,"state":"UP","age":25,"semesterMarks":[80,80.1,98,70 },
"salary":10000
}

6. Matching values using $gt operator:

In this example, we are selecting those documents where the value of the salary field is greater than 1000.

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

{
"_id":Objectid("5e7b9f0a92e6dfa3fc48ddbf"),
"name":"Amit",
"branch""ECE",
"joiningYear":2017,
"language":["Python","C#"],
"personal":{"contactinfo":234556789,"state":"UP","age":25,"semesterMarks":[80,80.1,98,70 },
"salary":10000
}
{
"_id":Objectid("5e7b9f0a92e6dfa3fc48ddc0"),
"name":"Sumit",
"branch":"CSE",
"joiningYear":2017,
"language":["Java","Perl"],
"personal":{"contactinfo":2300056789,"state":"MP","age":24,"semesterMarks":[89,80.1,78,71]},
"salary":15000
}

7. Matching values using $gte operator:

In this example, we are selecting those documents where the value of the joining Year field is greater than equals to 2018.

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

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

Ex. No: 5 Date:	Practice exercise on element, array based and evaluation query operators - $exists, $type, $mod, $regex

Aim:

To Practice exercise on element, array based and evaluation query operators -$exists, $type, $mod, $regex

Element Operators

The element query operators are used to find documents based on the document's fields. The Operators are

1. **$exists**
2. **$type**

Example documents:

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

```
> use Sample
Switched to db Sample

>db.contributor.insertMany([
{
name: "John",
age: 30,
address: { city: "New York", state: "NY" }
},
{
name: "Jane",
age: 25
},
{
name: "Bob",
age: 40,
address: "bvrm"
}
]);
```

Query examples:

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

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

Output:

{ "_id" : ObjectId("6479a0ff2615b529741554d8"), "name" : "John", "age" : 30, "address" : { "city" : "New York", "state" : "NY" } }

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

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

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

Output:

{ "_id" : ObjectId("6479a0ff2615b529741554d8"), "name" : "John", "age" : 30, "address" : { "city" : "New York", "state" : "NY" } }

Array Operators

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

1. **$all**
2. **$size**
3. **$elemMatch**

Example documents:

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

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

{ name: "Bob", hobbies: null }

Query examples:

1. $all: find all documents where hobbies include both "reading" and "hiking"

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

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

db.inventory.find({
sizes: { $elemMatch: { h: { $gt: 15 }, w: { $gt: 20 } } },
status: "A"
})

3. $size: selects documents if the array field is a specified size.

db.inventory.find({ tags: { $size: 3 } })

Evaluation Operators

The MongoDB evaluation operators can assess a document's overall data structure as well as individual fields. Because these operators might be considered advanced MongoDB capabilities, we are merely looking at their fundamental functionality. The following is a list of MongoDB's most popular evaluation operators.

Example documents:

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

```
> use Sample
Switched to db Sample

> db.editors.insertMany( [
{ "_id" : "1001", "name" : { "first" : "Daniel", "last" : "Atlas" }, "age" : 27, "grades" : {
"JavaCodeGeek" : "A", "WebCodeGeek" : "A+", "DotNetCodeGeek" : "A" } },
{ "_id" : "1002", "name" : { "first" : "Charlotte", "last" : "Neil" }, "age" : 24 },
{ "_id" : "1003", "name" : { "first" : "James", "last" : "Breen" }, "age" : 17 },
{ "_id" : "1004", "name" : { "first" : "John", "last" : "Gordon" }, "age" : 20 },
{ "_id" : "1005", "name" : { "first" : "Rick", "last" : "Ford" }, "age" : 25, "grades" : {
"JavaCodeGeek" : "A+", "WebCodeGeek" : "A+", "DotNetCodeGeek" : "A" } },
{ "_id" : "1006", "name" : { "first" : "Susan", "last" : "Dixit" }, "age" : 32 },
{ "_id" : "1007", "name" : { "first" : "John", "last" : "Snow" }, "age" : 21 },
{ "_id" : "1008", "name" : { "first" : "Arya", "last" : "Stark" }, "age" : 25 },
{ "_id" : "1009", "name" : { "first" : "April", "last" : "Paul" }, "age" : 28, "grades" : {
"JavaCodeGeek" : "A+", "WebCodeGeek" : "A", "DotNetCodeGeek" : "A+" } },
{ "_id" : "1010", "name" : { "first" : "Samir", "last" : "Pathak" }, "age" : 31 }
] )

> db.editors.find()
```

Output:

Output:

```
{
"acknowledged" : true,
"insertedIds" : [
"1001",
"1002",
"1003",
"1004",
```

"1005",
"1006",
"1007",
"1008",
"1009",
"1010"
]
}
{ "_id" : "1001", "name" : { "first" : "Daniel", "last" : "Atlas" }, "age" : 27, "grades" : {
"JavaCodeGeek" : "A", "WebCodeGeek" : "A+", "DotNetCodeGeek" : "A" } }
{ "_id" : "1002", "name" : { "first" : "Charlotte", "last" : "Neil" }, "age" : 24 }
{ "_id" : "1003", "name" : { "first" : "James", "last" : "Breen" }, "age" : 17 }
{ "_id" : "1004", "name" : { "first" : "John", "last" : "Gordon" }, "age" : 20 }
{ "_id" : "1005", "name" : { "first" : "Rick", "last" : "Ford" }, "age" : 25, "grades" : {
"JavaCodeGeek" : "A+", "WebCodeGeek" : "A+", "DotNetCodeGeek" : "A" } }
{ "_id" : "1006", "name" : { "first" : "Susan", "last" : "Dixit" }, "age" : 32 }
{ "_id" : "1007", "name" : { "first" : "John", "last" : "Snow" }, "age" : 21 }
{ "_id" : "1008", "name" : { "first" : "Arya", "last" : "Stark" }, "age" : 25 }
{ "_id" : "1009", "name" : { "first" : "April", "last" : "Paul" }, "age" : 28, "grades" : {
"JavaCodeGeek" : "A+", "WebCodeGeek" : "A", "DotNetCodeGeek" : "A+" } }

1. $mod Operator: mod operator allows the user to get those documents from a collection where the specific field when divided by a divisor has an even or odd remainder. This operator works like the WHERE clause of the SQL programming language. The $mod operator works only on the integer values and here is what the query syntax will look like.

Syntax

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

Where:

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

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

Query 1:

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

{

```
        "acknowledged" : true,
        "insertedIds" : [
                "1001",
                "1002",
                "1003",
                "1004",
}               "1005",
{               "1006",
                "1007",
                "1008",
                "1009",
                 "1010"
}       ]
{


        "_id" : "1004",
        "name" : {
        "first" : "John",
                 "last" : "Gordon"
        },
        "age" : 20


        "_id" : "1005",
}       "name" : {
{       "first" : "Rick",
        "last" : "Ford"
        },
        "age" : 25,
        "grades" : {
}                "JavaCodeGeek" : "A+",
2.               "WebCodeGeek" : "A+",
                "DotNetCodeGeek" : "A"
        }


    "_id" : "1008",
    "name" : {
    "first" : "Arya",
    "last" : "Stark"
    },
    "age" : 25
```

$regex Operator:

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

collection where a string matches a specified pattern.

Syntax:

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

Where:

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

A pattern is a regular expression for a complex search

Query:

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

Output:

```
{
"acknowledged" : true,
"insertedIds" : [ "1001","1002","1003","1004","1005","1006",
"1007","1008","1009","1010"
]
}
{
"_id" : "1008",
"name" : {
"first" : "Arya",
"last" : "Stark"
},
"age" : 25
}
{
"_id" : "1009",
"name" : {
"first" : "April",
"last" : "Paul"
},
"age" : 28,
"grades" : {
"JavaCodeGeek" : "A+",
"WebCodeGeek" : "A",
"DotNetCodeGeek" : "A+"
}
}
```

Ex. No: 6 Date:	Exercise on MongoDB shell commands and user management

Aim:

To exercise on MongoDB Shell Commands and User Management Commands

MongoDB Shell Commands

1. Connect a Database

To connect to a local database instance, then by running mongosh it will connect to localhost as default.

mongosh "mongodb://localhost:27017" -u [user]

Once you have connected, you can run help to get some basic commands:

> help

```
> help

  Shell Help:

    use               Set current database
    show              'show databases'/'show dbs': Print a list of all available databases.
                      'show collections'/'show tables': Print a list of all collections for current dat
                      'show profile': Prints system.profile information.
                      'show users': Print a list of all users for current database.
                      'show roles': Print a list of all roles for current database.
                      'show log <type>': log for current connection, if type is not set uses 'global'
                      'show logs': Print all logs.

    exit              Quit the MongoDB shell with exit/exit()/.exit
    quit              Quit the MongoDB shell with quit/quit()
    Mongo             Create a new connection and return the Mongo object. Usage: new Mongo(URI, option
    connect           Create a new connection and return the Database object. Usage: connect(URI, usern
    it                result of the last line evaluated; use to further iterate
    version           Shell version
    load              Loads and runs a JavaScript file into the current shell environment
    enableTelemetry   Enables collection of anonymous usage data to improve the mongosh CLI
    disableTelemetry  Disables collection of anonymous usage data to improve the mongosh CLI
    passwordPrompt    Prompts the user for a password
    sleep             Sleep for the specified number of milliseconds
    print             Prints the contents of an object to the output
    printjson         Alias for print()
    cls               Clears the screen like console.clear()
    isInteractive     Returns whether the shell will enter or has entered interactive mode

  For more information on usage: https://docs.mongodb.com/manual/reference/method
```

2. Listing the databases

> show dbs

We want to be able to switch between the databases, so lets switch to the admin database.

> use admin

3. Creating a database

To create a new database to store our data. To do this, we can run:

> use entertainment

4. Deleting a database

> use entertainment

switched to db entertainment

And then let's drop the database.

> db.dropDatabase()

{ ok: 1, dropped: 'entertainment' }

5. Creating a collection

Using the commands from above, create a database called entertainment again. Then switch to that database. We now want to create a collection. Let's just make sure we are on the same page again. Let's check the name of the database we are both on.

> db.getName()

entertainment

Your output should match mine, entertainment. Now, let's create a collection called films.

> db.createCollection("films")

6. Showing collections

> show collections

films

7. Inserting documents

Now we have a database (entertainment), and a collection (films), let's create some documents.

>**db.films.insertOne({ title: "The Big Lebowski", url:"https://en.wikipedia.org/wiki/The_Big_Lebow**

{

acknowledged: true,

insertedId: ObjectId("626eaad80dd6e8884b86fda3")

}

This has created a document in the films collection with two attributes: title, and url. Behind the scenes, MongoDB will add an additional attribute: _id which cannot be changed.

You can also create many documents at the same time.

```
>db.films.insertMany([{ title: "How to loose a guy in 10 days", url: "https://en.wikipedia.org/wiki/How_to_Lose_a_Guy_in_10_Days" },
{ title: "Father of the Bride", url: "https://www.imdb.com/title/tt0101862/" }])
{
acknowledged: true,
insertedIds: {
'0': ObjectId("626eaae00dd6e8884b86fda4"),
'1': ObjectId("626eaae00dd6e8884b86fda5")
}}
```

8. Getting documents

```
> db.films.find({})
[ {
_id: ObjectId("626eaad80dd6e8884b86fda3"),
title: 'The Big Lebowski',
url: 'https://en.wikipedia.org/wiki/The_Big_Lebowski'
},
{
_id: ObjectId("626eaae00dd6e8884b86fda4"),
title: 'How to loose a guy in 10 days',
        url: 'https://en.wikipedia.org/wiki/How_to_Lose_a_Guy_in_10_Days'
},
{
_id: ObjectId("626eaae00dd6e8884b86fda5"),
title: 'Father of the Bride',
url: 'https://www.imdb.com/title/tt0101862/'
}
]
```

9. Filtering documents

If you know the id of the document:

```
    > db.films.find({ _id: ObjectId("626eaae00dd6e8884b86fda5") })
[
{
_id: ObjectId("626eaae00dd6e8884b86fda5"),
title: 'Father of the Bride',
```

```
 url: 'https://www.imdb.com/title/tt0101862/'
}]
          > db.films.findOne({ _id: ObjectId("626eaae00dd6e8884b86fda5") })
{
_id: ObjectId("626eaae00dd6e8884b86fda5"),
title: 'Father of the Bride',
url: 'https://www.imdb.com/title/tt0101862/'
}
```

10. Updating documents

```
db.films.updateOne({ title: "The Big Lebowski" }, { $set: { rating: 5 }})
{
              acknowledged: true, insertedId: null, matchedCount: 1, modifiedCount: 1,
upsertedCount: 0
}
> db.films.find({})
[ {
_id: ObjectId("626eaad80dd6e8884b86fda3"),
title: 'The Big Lebowski',
url: 'https://en.wikipedia.org/wiki/The_Big_Lebowski',
rating: 5
},
{
_id: ObjectId("626eaae00dd6e8884b86fda4"),
title: 'How to loose a guy in 10 days',
url: 'https://en.wikipedia.org/wiki/How_to_Lose_a_Guy_in_10_Days'
},
{
_id: ObjectId("626eaae00dd6e8884b86fda5"),
title: 'Father of the Bride',
url: 'https://www.imdb.com/title/tt0101862/'
}
]
```

11. Deleting documents

```
        > db.films.deleteOne({_id: ObjectId("626eaae00dd6e8884b86fda5")})
{ acknowledged: true, deletedCount: 1 }
```

MongoDB User Management Commands

1. Create New User - createUser Command

The createUser command is used to create a new user on the database where you run the command. The createUser command returns an error message if the user already exists.

A database name must have to mention at the time to create a new user with createUser action.

A grantRole action must have to mention on a role's database to grant the role to another user.
If you have the userAdmin or userAdminAnyDatabase role, or if you are authenticated using the localhost exception, you have those actions.

Syntax

```
{
createUser: "<name>",
pwd: "<cleartext password>",
customData: { <any information> },
roles: [
{ role: "<role>", db: "<database>" } | "<role>",
...
],
writeConcern: { <write concern> }
}
```

Parameters:

Name	Type	Description
createUser	string	The name of the new user.
pwd	string	The user's password.
customData	document	Optional.
roles	array	The roles granted to the user
writeConcern	document	Optional. The write concern briefs the guarantee that MongoDB provides at the time of reporting on the success of a write operation. w: 1 as the default write concern.

Assume that our database name is userdetails and there are the following document:

```
db.userdetails.insert({"user_id" : "user1","password" :"1a2b3c" ,
"date_of_join" : "16/10/2010" ,"education" :"M.C.A." ,
"profession" : "CONSULTANT","interest" : "MUSIC",
```

"community_name" :["MODERN MUSIC", "CLASSICAL MUSIC","WESTERN MUSIC"],
"community_moder_id" : ["MR. Alex","MR. Dang","MR Haris"],
"community_members" : [700,200,1500],
"friends_id" : ["kumar","harry","anand"],
"ban_friends_id" :["Amir","Raja","mont"]});

Example

The following createUser command creates a user myNewuser on the userdetails database. The command gives myNewuser the clusterAdmin and readAnyDatabase roles on the admin database and the readWrite role on the userdetails database:

```
db.getSiblingDB("userdetails").runCommand( { createUser: "myNewuser",
pwd: "thisPassword",
customData: { profession : "DOCTOR" },
roles: [
{ role: "clusterAdmin", db: "admin" },
{ role: "readAnyDatabase", db: "admin" },"readWrite"
],
writeConcern: { interest : "SPORTS" }
} )
```

Example: Create more user

The following createUser command creates a user myNewuser1 on the userdetails database. The command gives myNewuser1 the clusterAdmin and the readWrite role on the userdetails database:

```
db.getSiblingDB("userdetails").runCommand( { createUser: "myNewuser1",
pwd: "thisPassword",
customData: { profession : "DOCTOR" },
roles: [
{ role: "clusterAdmin", db: "admin" },"readWrite"
],
writeConcern: { interest : "SPORTS" }
} )
```

If we type wrongly in the above statement myNewuser instead of myNewuser1 then the following error message as an output will appear beacuse the user already exists.

{
"ok" : 0,
"errmsg" : "User \"myNewuser@userdetails\" already exists",
"code" : 11000
}

2. **To see the User Information - usersInfo Command**

The command usersInfo returns information about one or more users.

Syntax:

{
usersInfo: { user: <name>, db: <db> },
showCredentials: <Boolean>,
showPrivileges: <Boolean>
}

Parameters:

Name	Type	Description
usersInfo	various	The user(s) about whom to return information.
showCredentials	Boolean	Optional. If the field set to true the user's password hash will be displayed, it is false by default.
showPrivileges	Boolean	Optional. If the field set to true to show the user's full set of privileges, including expanded information for the inherited roles, it is false by default.

Example to View Specific User

To see information but not the credentials, for the user "myNewuser" defined in "userdetails" database, run the following command:

db.runCommand(
{
usersInfo: { user: "myNewuser", db: "userdetails" },
showPrivileges: false
}
)

Output:

{
"users" : [
{
"_id" : "userdetails.myNewuser",
"user" : "myNewuser", "db" : "userdetails",
"customData" : {
"profession" : "DOCTOR"
},
"roles" : [
{
"role" : "clusterAdmin",
"db" : "admin"
},
{
"role" : "readAnyDatabase",
"db" : "admin"
},
{
"role" : "readWrite",
"db" : "userdetails"
}
]
}
],
"ok" : 1
}

Example to View Multiple User

To view info for several users, use an array, with or without the optional fields showPrivileges and showCredentials.

For example:

db.runCommand({ usersInfo: [{ user: "myNewuser", db: "userdetails" },
{ user: "myNewuser1", db: "userdetails" }],
showPrivileges: false
});

Output:


```
{
"users" : [
{
"_id" : "userdetails.myNewuser",
"user" : "myNewuser",
"db" : "userdetails",
"customData" : {
"profession" : "DOCTOR"
},
"roles" : [
{
"role" : "clusterAdmin",
"db" : "admin"
},
{
          "role" : "readAnyDatabase",
"db" : "admin"
},
{
"role" : "readWrite",
"db" : "userdetails"
}
]
},
{
"_id" : "userdetails.myNewuser1",
"user" : "myNewuser1",
"db" : "userdetails",
"customData" : {
"profession" : "Engineer"
},
"roles" : [
{
"role" : "clusterAdmin",
"db" : "admin"
},
{
"role" : "readWrite",
"db" : "userdetails"
                                                    } ]
}
],
"ok" : 1
}
```

Example to View All Users for a Database

To view all users on the database the following command can be used:

```
db.runCommand( { usersInfo: 1 , showCredentials : false} );
```

Output:

```
{
"users" : [
{
"_id" : "userdetails.myNewuser",
"user" : "myNewuser",
"db" : "userdetails",
"customData" : {
"profession" : "DOCTOR"
},
"roles" : [
{
"role" : "clusterAdmin",
"db" : "admin"
},
{
"role" : "readAnyDatabase",
"db" : "admin"
},
{
"role" : "readWrite",
"db" : "userdetails"
}
]
},
{
"_id" : "userdetails.myNewuser1",
"user" : "myNewuser1",
"db" : "userdetails",
"customData" : {
"profession" : "Engineer"
},
"roles" : [
{
"role" : "clusterAdmin",
"db" : "admin"
},
{
"role" : "readWrite",
"db" : "userdetails"
}
```

```
 ]
}
],
"ok" : 1
}
```

If showCredential is true the output will be the following:

```
{
"users" : [
{
"_id" : "userdetails.myNewuser",
"user" : "myNewuser",
"db" : "userdetails",
"credentials" : {
        "MONGODB-CR" : "28fcdbb5d879ccf086bd6404bd06b230"
},
"customData" : {
"profession" : "DOCTOR"
},
"roles" : [
{
"role" : "clusterAdmin",
"db" : "admin"
},
{
"role" : "readAnyDatabase",
"db" : "admin"
},
{
"role" : "readWrite",
"db" : "userdetails"
}
]
},
{
"_id" : "userdetails.myNewuser1",
"user" : "myNewuser1",
"db" : "userdetails",
"credentials" : {
        "MONGODB-CR" : "9b08b3a68eef34d462bdb2e5b2037438"
},
"customData" : {
"profession" : "Engineer"
},
"roles" : [
{
```

"role" : "clusterAdmin",
"db" : "admin"
},
{
"role" : "readWrite",
"db" : "userdetails"
}
]
}
],
"ok" : 1
}

3. To Update the User Information - updateUser Command

The updateUser command updates the profile of the user on the specific database. This command completely replaces the data of the previous field's values. It is required to specify the updateUser field and at least one other field, other than writeConcern to update a user.

{ updateUser: "<username>",
pwd: "<cleartext password&glt;",
customData: { <any information> },
roles: [
{ role: "<role>", db: "<database>" } | "<role>",
...
],
writeConcern: { <write concern> }
}

Parameters:

Name	Type	Description
updateUser	string	The name of the user to update.
pwd	string	Optional. The user's password.
customData	document	Optional. Any arbitrary information.
roles	array	Optional. The roles granted to the user. An update to the roles array replace the values of the previous array.
writeConcern	document	Optional. The write concern briefs the guarantee that MongoDB provides at the time of reporting on the success of a write operation. w: 1 as the default write concern.

Example

Here in below a user myNewuser in the userdetails database with the following user info:

```
{
"users" : [
{
"_id" : "userdetails.myNewuser",
"user" : "myNewuser",
"db" : "userdetails",
"customData" : {
"profession" : "DOCTOR"
},
"roles" : [
{
"role" : "clusterAdmin",
"db" : "admin"
},
{
"role" : "readAnyDatabase",
"db" : "admin"
},
{
"role" : "readWrite",
"db" : "userdetails"
}
]
}
]
}
```

The following updateUser command completely replaces the user's customData and roles data:

```
use userdetails
db.runCommand( { updateUser : "myNewuser",
customData : { profession : "Classical Music" },
roles : [
{ role : "read", db : "assets" }
]
} )
```

Here is the output after update the user:

```
> db.runCommand(
{
usersInfo: { user: "myNewuser", db: "userdetails" },
showPrivileges: false
```

```
 }
);

{
"users" : [
{
"_id" : "userdetails.myNewuser",
"user" : "myNewuser",
"db" : "userdetails",
"customData" : {
"profession" : "Classical Music"
},
"roles" : [
{
"role" : "read",
"db" : "assets"
}
]
}
],
"ok" : 1
}
```

4. To Drop the User Information - dropUser Command

The dropUser command is used to remove the user from the concern database.

Syntax:

{

dropUser: "<user>",

writeConcern: { <write concern> }

}

Parameters:

Name	Type	Description
dropUser	string	The name of the user to delete. This command will work only the database where the user exists.
writeConcern	document	Optional. The write concern briefs the guarantee that MongoDB provides at the time of reporting on the success of a write operation. w: 1 as the default write concern. .

Example

If we want to remove the user myNewuser1 from the userdetails database the following sequence of operations in the mongo shell can be used.

> use userdetails

switched to db userdetails

> db.runCommand({ dropUser: "myNewuser1"});

{ "ok" : 1 }

Now if we want to view all the user for database userdetails the following result will appear where the user myNewuser1 will be excluded.

```
> db.runCommand( { usersInfo: 1 , showCredentials : false} );
{
"users" : [
{
"_id" : "userdetails.myNewuser",
"user" : "myNewuser",
"db" : "userdetails",
"customData" : {
"profession" : "Classical Music"
},
"roles" : [
{
"role" : "read",
"db" : "assets"
}
]
}
],
"ok" : 1
}
```

To Drop All the Users Information - dropAllUsersFromDatabase Command

This command removes all users from the concern database on which you run the command.

Syntax:

```
{
dropAllUsersFromDatabase: 1,
writeConcern: { <write concern> }
}
```

Parameters:

Name	Type	Description
dropAllUsersFromDatabase	integer	Specify 1 to drop all the users from the current database.
writeConcern	document	Optional. The write concern briefs the guarantee that MongoDB provides at the time of reporting on the success of a write operation. w: 1 as the default write concern.

Example

If we want to remove the user myNewuser1 from the userdetails database the following sequence of operations in the mongo shell can be used.

```
> use userdetails;
switched to db userdetails
        > db.runCommand( { dropAllUsersFromDatabase: 1, writeConcern: { w: "majority" }});
```

Here is the output

```
{ "n" : 1, "ok" : 1 }
```

Now if we want to view the information of the user for database userdetails the following result will appear.

```
> db.runCommand( { usersInfo: 1 , showCredentials : false} );
{ "users" : [ ], "ok" : 1 }
```

5. To Grants additional roles to a user - grantRolesToUser Command

This command grants additional roles to a user.

Syntax:

```
{ grantRolesToUser: "<user>",
roles: [ <roles> ],
writeConcern: { <write concern> }
}
```

Parameters:

Name	Type	Description
grantRolesToUser	string	The name of the user to give additional roles.
roles	array	An array of additional roles to grant to the user.
writeConcern	document	Optional. The write concern briefs the guarantee that MongoDB provides at the time of reporting on the success of a write operation. w: 1 as the default write concern.

Example

Given a user myNewuser2 in the userdetails database with the following roles:


```
{
"_id" : "userdetails.myNewuser2",
"user" : "myNewuser2",
"db" : "userdetails",
"customData" : {
"profession" : "painter"
},
"roles" : [
{
"role" : "read",
"db" : "usertransact"
},
{
"role" : "readWrite",
"db" : "userdetails"
}
]
}
```


The following grantRolesToUser operation gives myNewuser2 the read role on the usertransact database and the readWrite role on the userdetails database.


```
{
"_id" : "userdetails.myNewuser2",
"user" : "myNewuser2",
"db" : "userdetails",
"customData" : {
"profession" : "painter"
},
"roles" : [
{
"role" : "read",
"db" : "usertransact"
},
{
```

"role" : "read",
"db" : "useraccount"
},
{
"role" : "readWrite",
"db" : "userdetails"
}
]
};

6. To Revoke Roles From a User - revokeRolesFromUser Command

This command is used to remove a one or more roles from a user on the database where the roles exist.

Syntax:

{ revokeRolesFromUser: "<user>",
roles: [
{ role: "<role>", db: "<database>" } | "<role>",
...
],
writeConcern: { <write concern> }
}

Parameters:

Name	Type	Description
revokeRolesFromUser	string	The user to remove roles from.
roles	array	The roles to revoke from the user.
writeConcern	document	Optional. The write concern briefs the guarantee that MongoDB provides at the time of reporting on the success of a write operation. w: 1 as the default write concern.

Example

Given a user myNewuser2 in the userdetails database with the following roles:

{
"_id" : "userdetails.myNewuser2",
"user" : "myNewuser2",
"db" : "userdetails",

```
"customData" : {
"profession" : "painter"
},
"roles" : [
{
"role" : "read",
"db" : "usertransact"
},
{
"role" : "read",
"db" : "useraccount"
},
{
"role" : "readWrite",
"db" : "userdetails"
}
]
};
```

To remove the two of the user's roles such as the read role on the usertransact database and the readWrite role on the userdetails database the following sequence of commands can be used.

```
use userdetails;
db.runCommand( { revokeRolesFromUser: "myNewuser2",
roles: [
{ role: "read", db: "usertransact" },
"readWrite"
],
writeConcern: { w: "majority" }
} );
```

The user myNewuser2 in the userdetails database now has only one remaining role:

```
{
"_id" : "userdetails.myNewuser2",
"user" : "myNewuser2",
"db" : "userdetails",
"customData" : {
"profession" : "painter"
},
"roles" : [
{
"role" : "read",
"db" : "useraccount"
}
]
}
```

Ex. No: 7 Installation and configuration of Cassandra. Find out two use cases where Cassandra is preferred over MongoDB

Date:

Aim: Install and Configure MongoDB to execute NoSQL Commands.

Apache Cassandra requires Java 8 to run on a Windows system. Additionally, the Cassandra command-line shell (cqlsh) is dependent on Python 2.7 to work correctly.

To be able to install Cassandra on Windows, first you need to:

1. Download and Install Java 8 and set environment variables.
2. Download and install Python 2.7 and set environment variables.

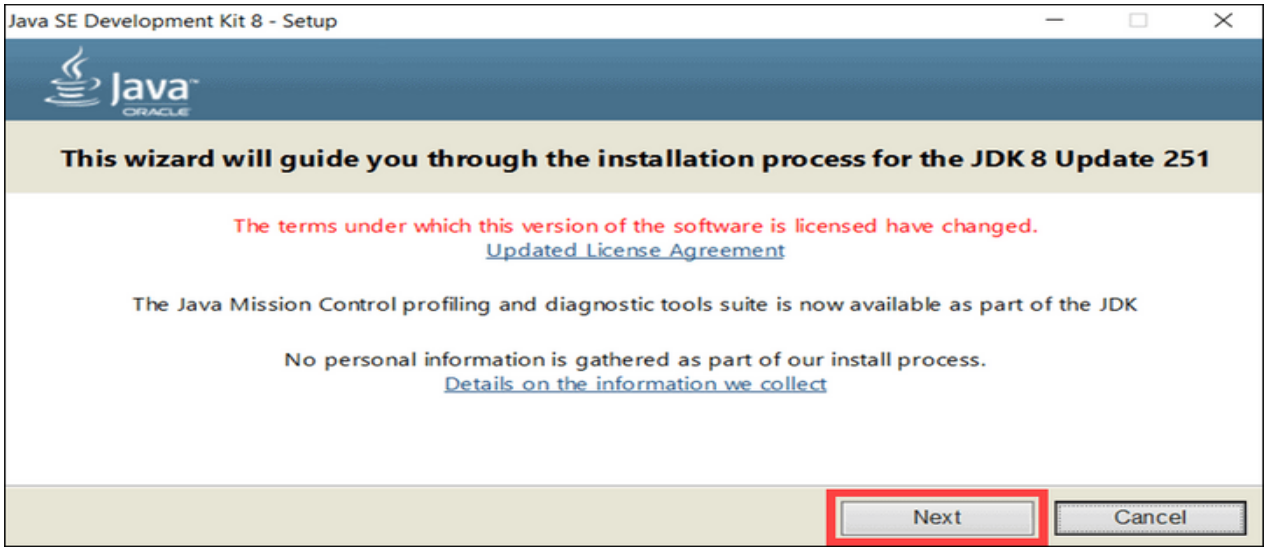
Step 1: Install Java 8 on Windows

The Java development kit contains all the tools and software you need to run applications written in Java. It is a prerequisite for software solutions such as Apache Cassandra.

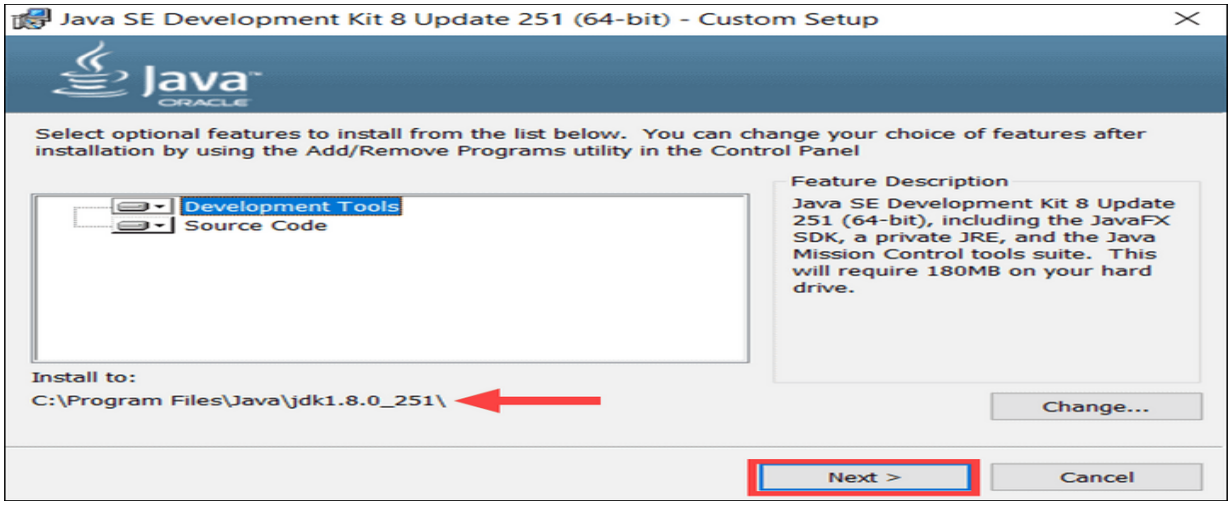
Download Oracle JDK 8 (Java Development Kit)

1. Visit the official Oracle download page and download the Oracle JDK 8 software package.
2. Scroll down and locate the Java SE Development Kit 8u251 for Windows x64 download link. The Java 8 download starts automatically after signup.

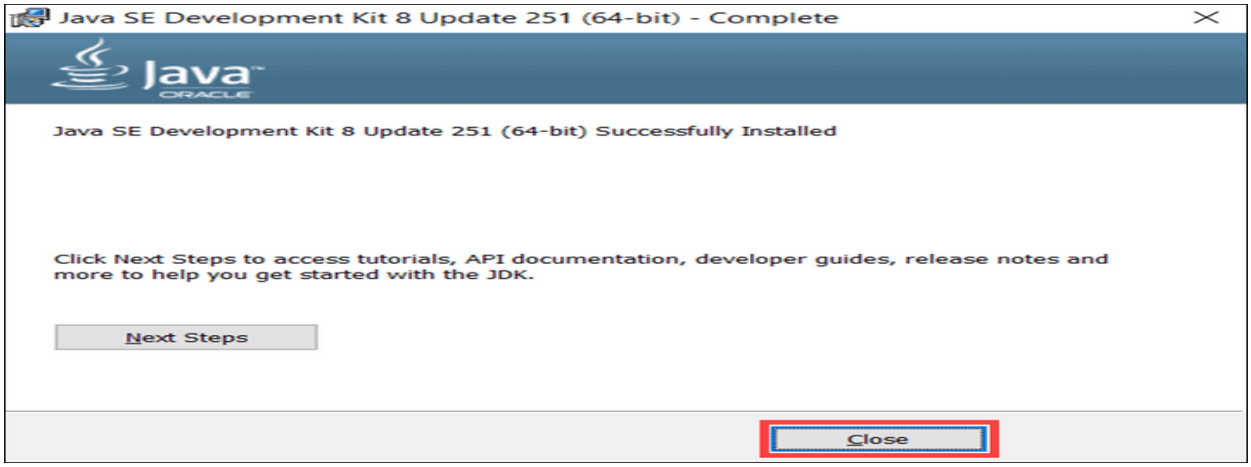
3. Once the download is complete, double-click the downloaded executable file. Select Next on the initial installation screen.



4. The following section allows you to select optional features and define the location of the installation folder. Accept the default settings and take note of the full path to the installation folder, C: Program FilesJavajdk1.8.0_251. Once you are ready to proceed with the installation, click Next.



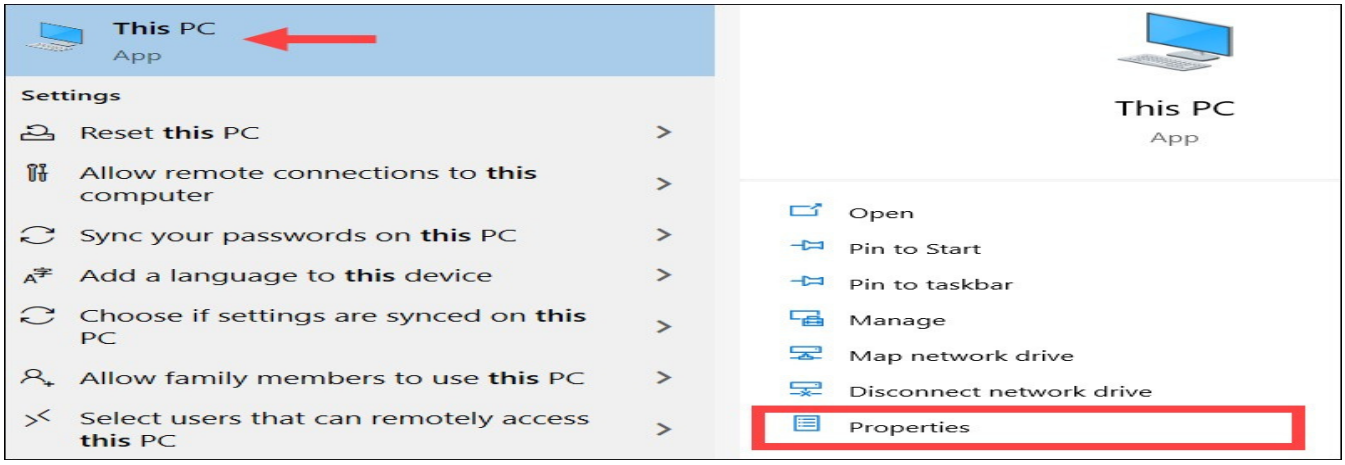
5. The installation process can take several minutes. Select Close once the process is completed.



Configure Environment Variables for Java 8

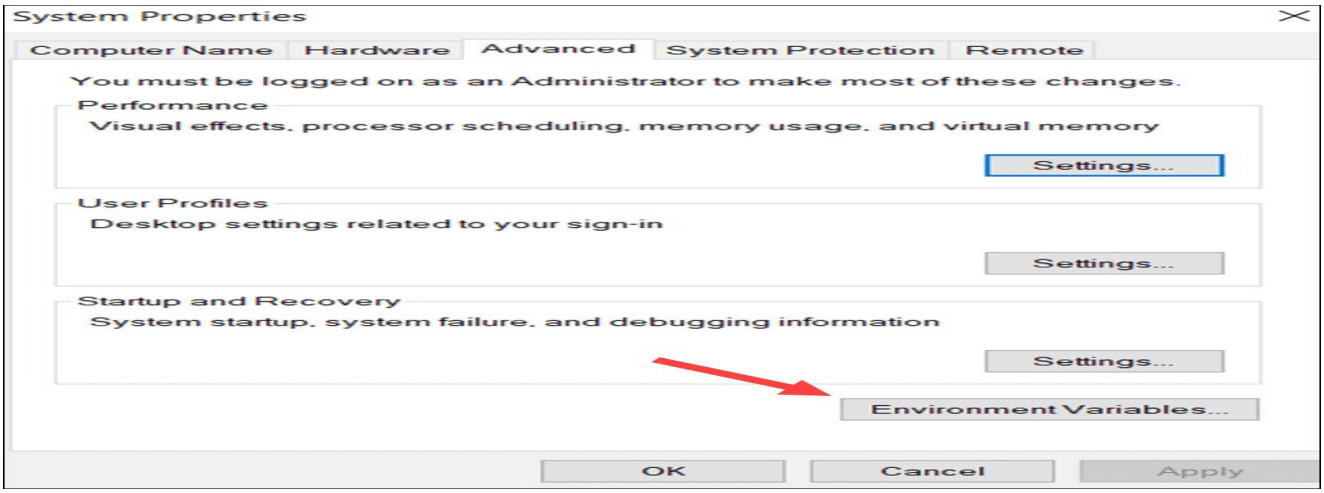
It is vital to configure the environment variables in Windows and define the correct path to the Java 8 installation folder.

1. Navigate to This PC > Properties.

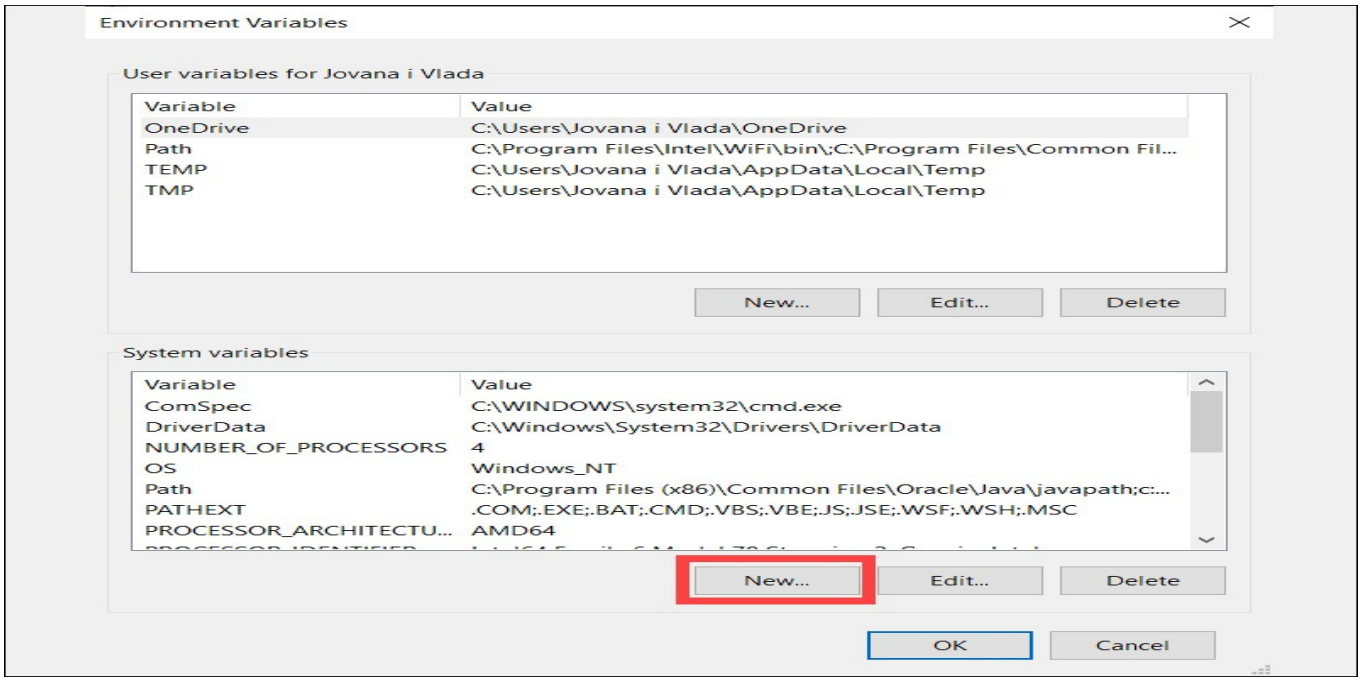


2. Select Advanced system settings.

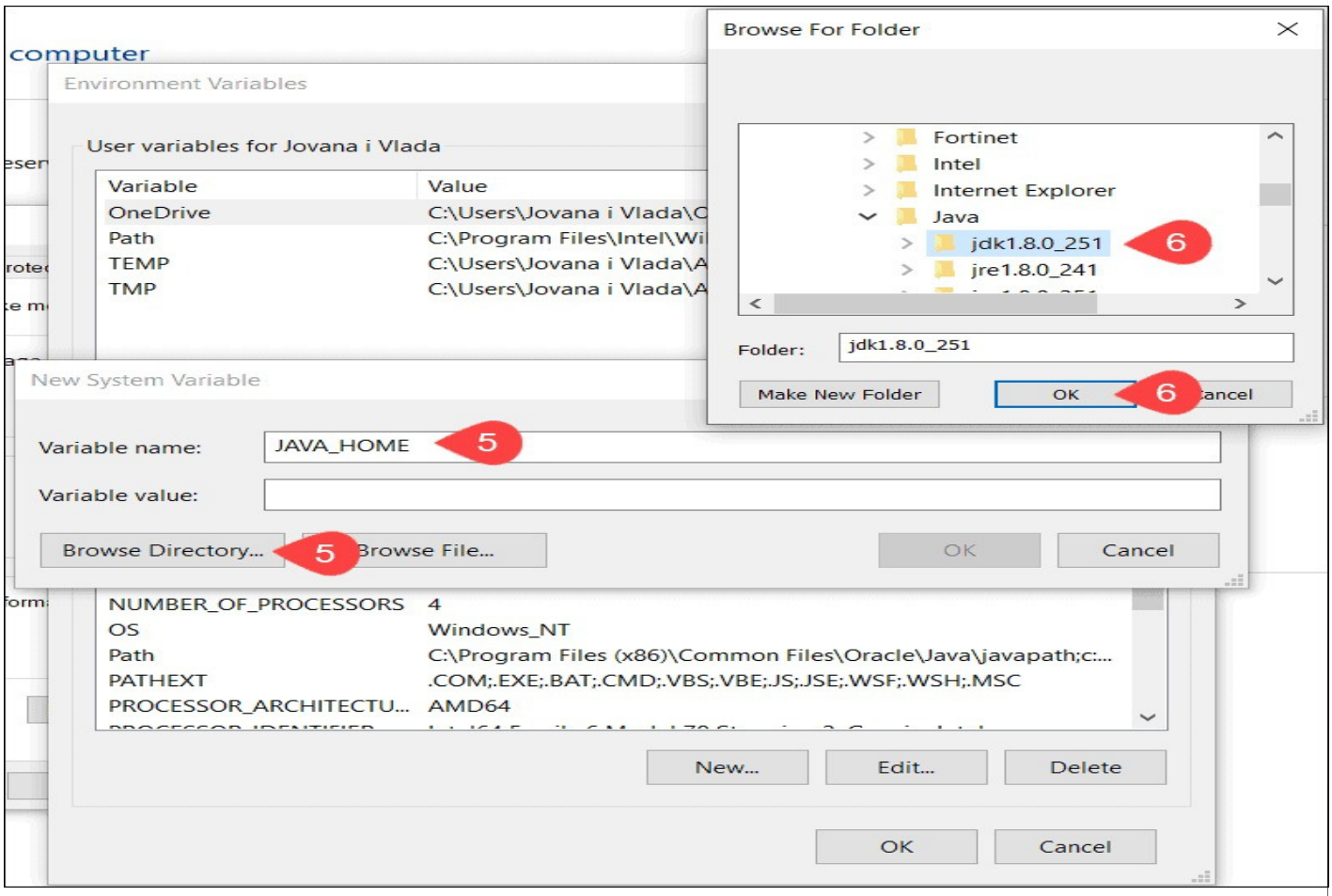
3. Click the Environment Variables... button.



4. Select New in the *System Variable* section.

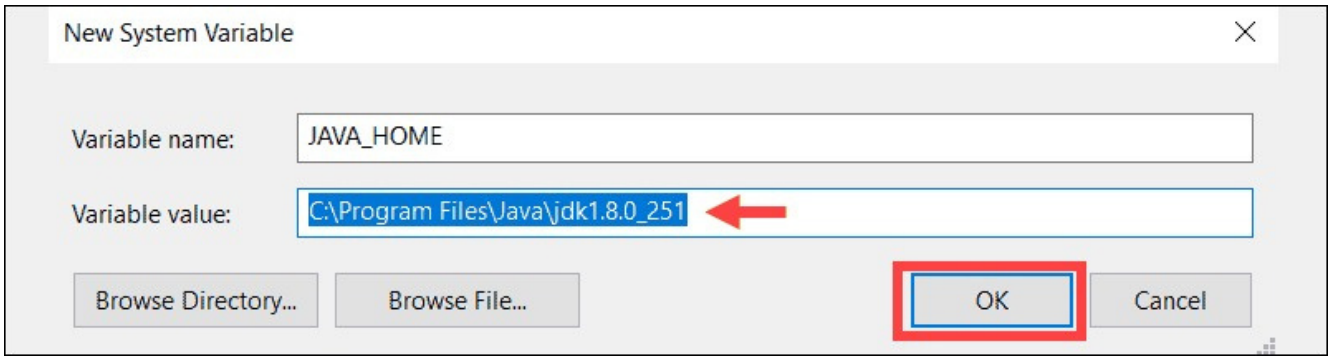


5. Enter JAVA_HOME for the new variable name. Select the *Variable value* field and then the Browse Directory option.

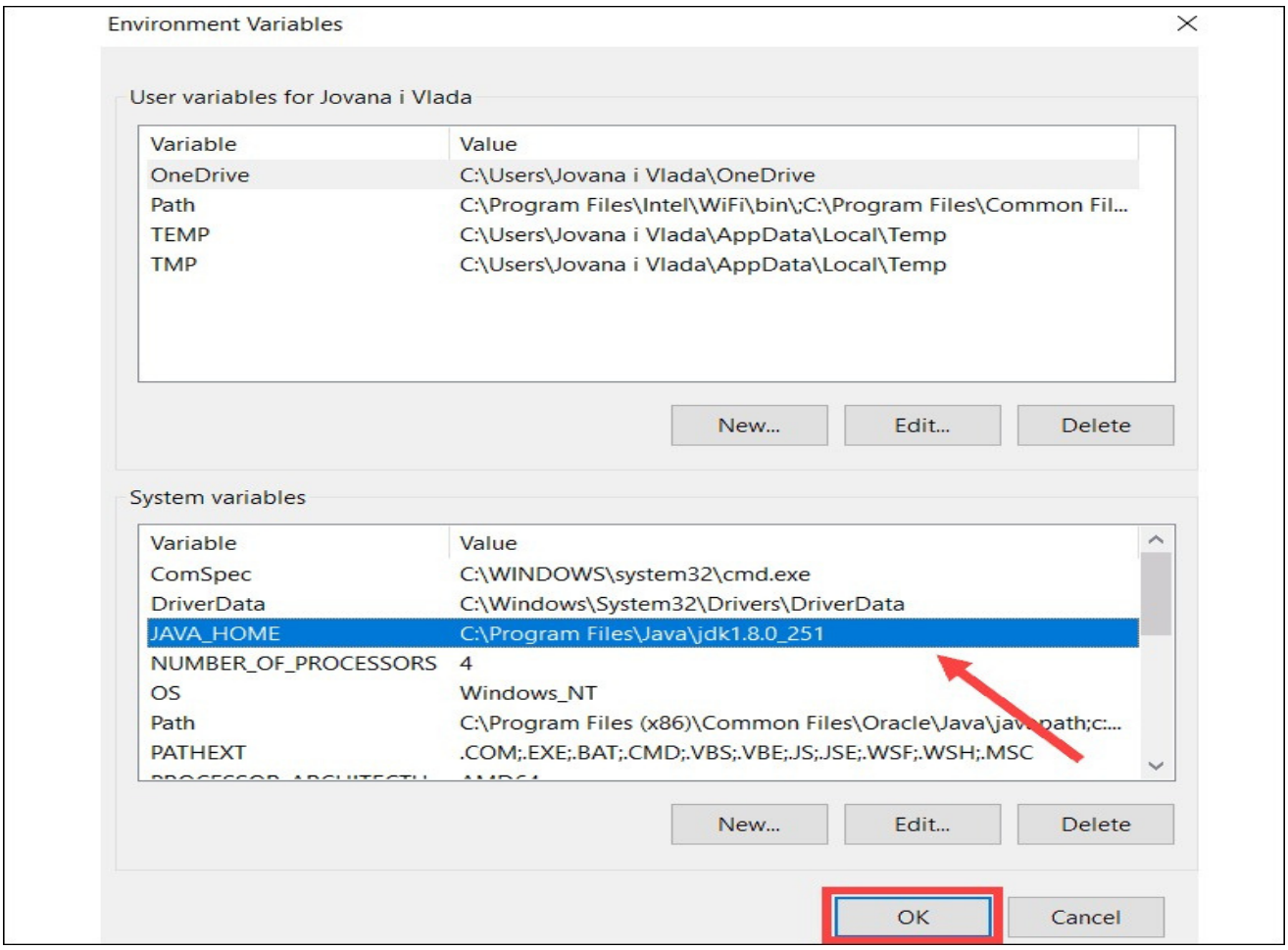


6. Navigate to This PC > Local Disk C: > Program Files > Java > jdk1.8.0_251 and select OK.

7. Once the correct path to the JDK 8 installation folder has been added to the JAVA_HOME system variable, click OK.



8. You have successfully added the JAVA_HOME system variable with the correct JDK 8 path to the variable list. Select OK in the main *Environment Variables* window to complete the process.

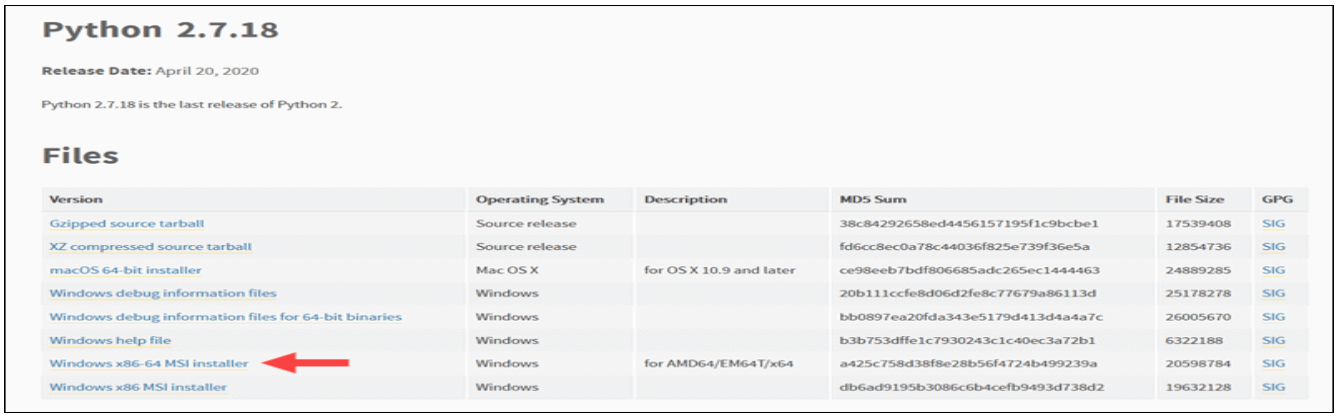


Step 2: Install and Configure Python 2.7 on Windows

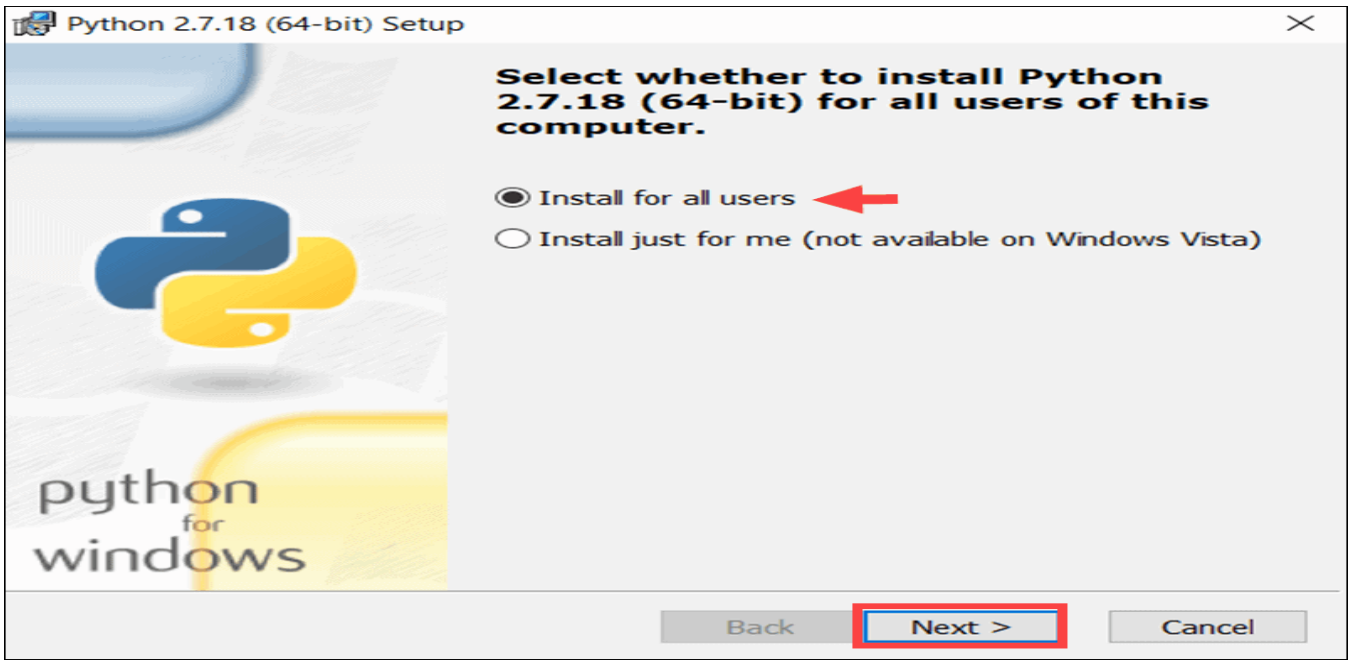
Users interact with the Cassandra database by utilizing the cqlsh bash shell. You need to install Python 2.7 for cqlsh to handle user requests properly.

Install Python 2.7 on Windows

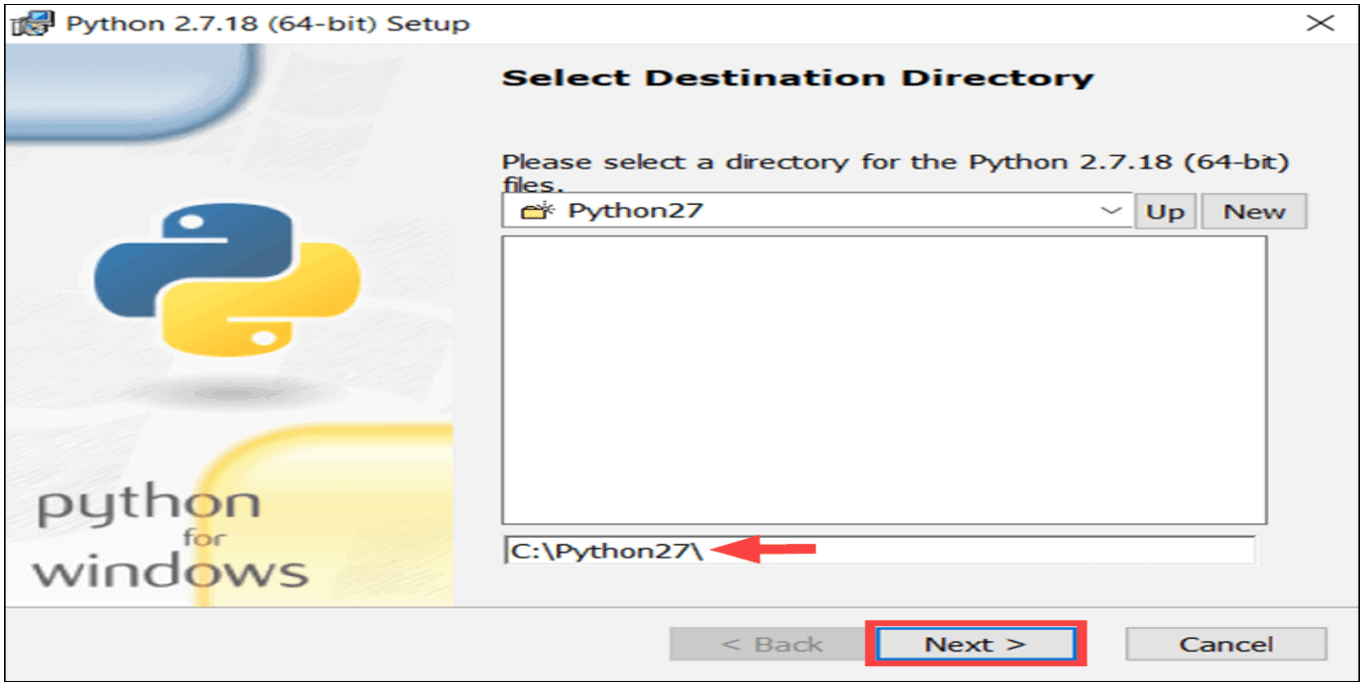
1. Visit the Python official download page and select the Windows x64 version link.



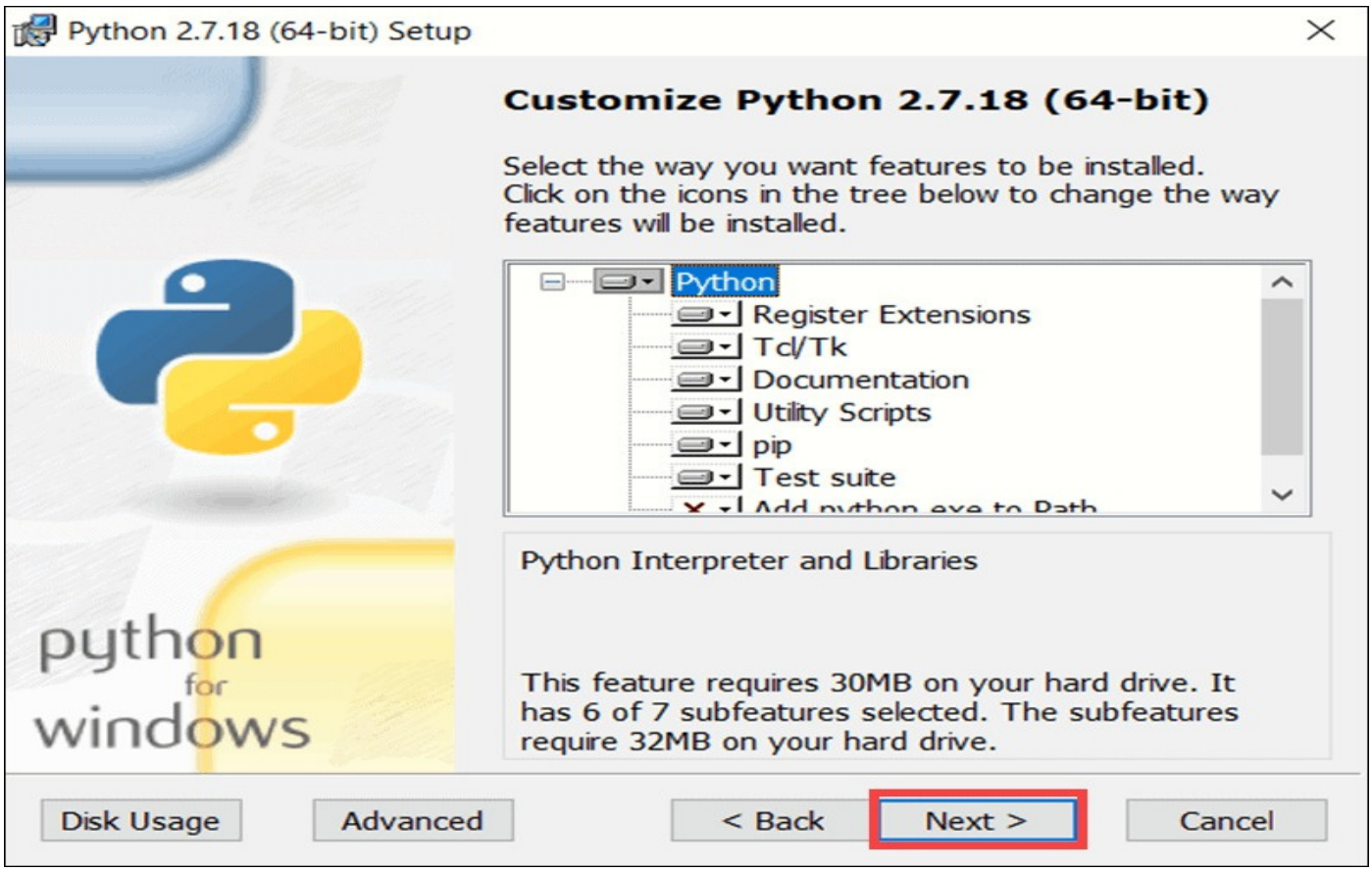
2. Define if you would like Python to be available to all users on this machine or just for your user account and select Next.



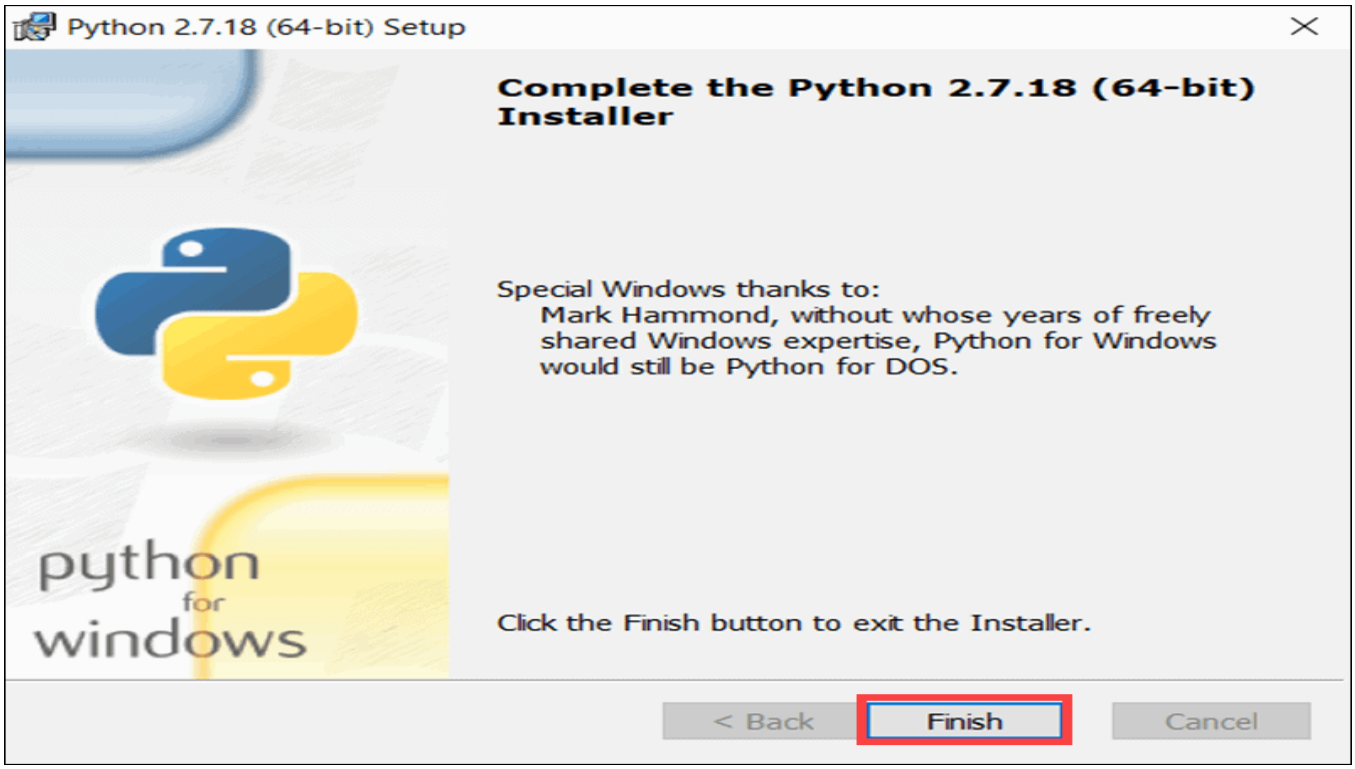
3. Specify and take note of the Python installation folder location. Feel free to leave the default location C:Python27 by clicking Next.



4. The following step allows you to customize the Python installation package. Select Next to continue the installation using the default settings.

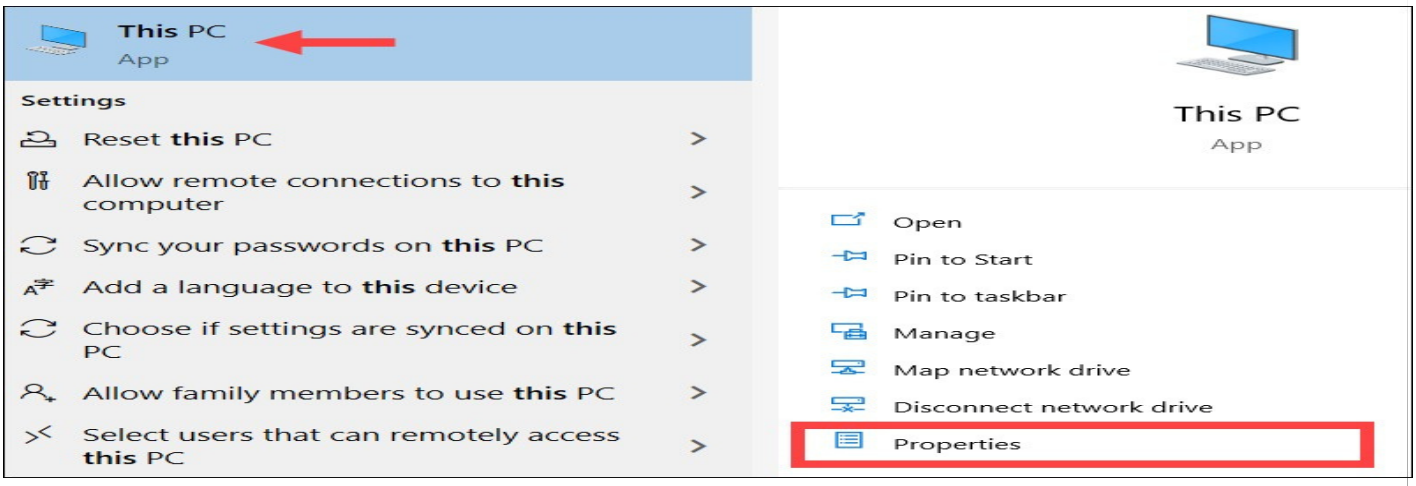


5. The installation process takes a few moments. Once it is complete, select Finish to conclude the installation process.



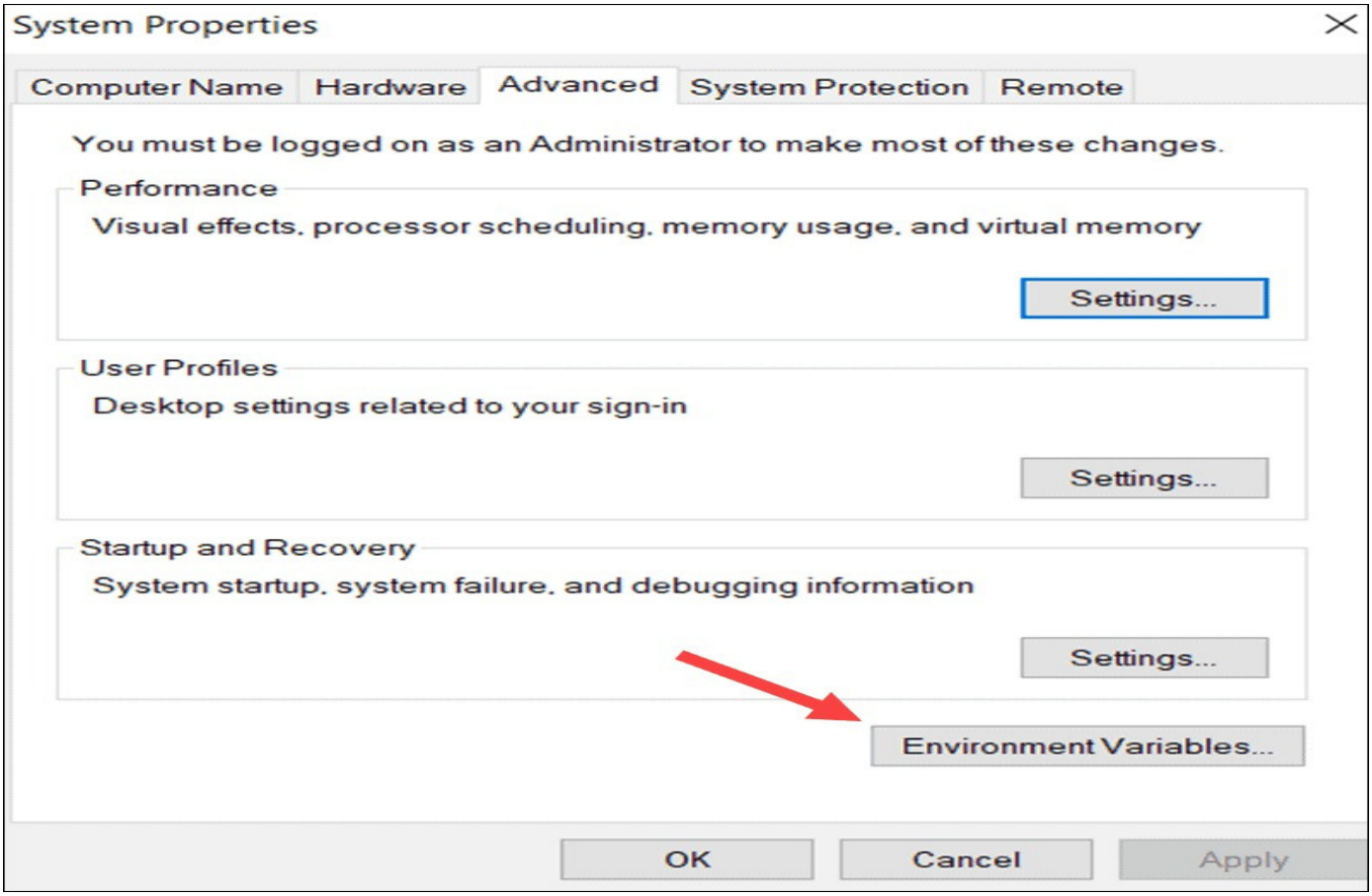
Edit Environment Variable for Python 2.7

1. Navigate to This PC > Properties.

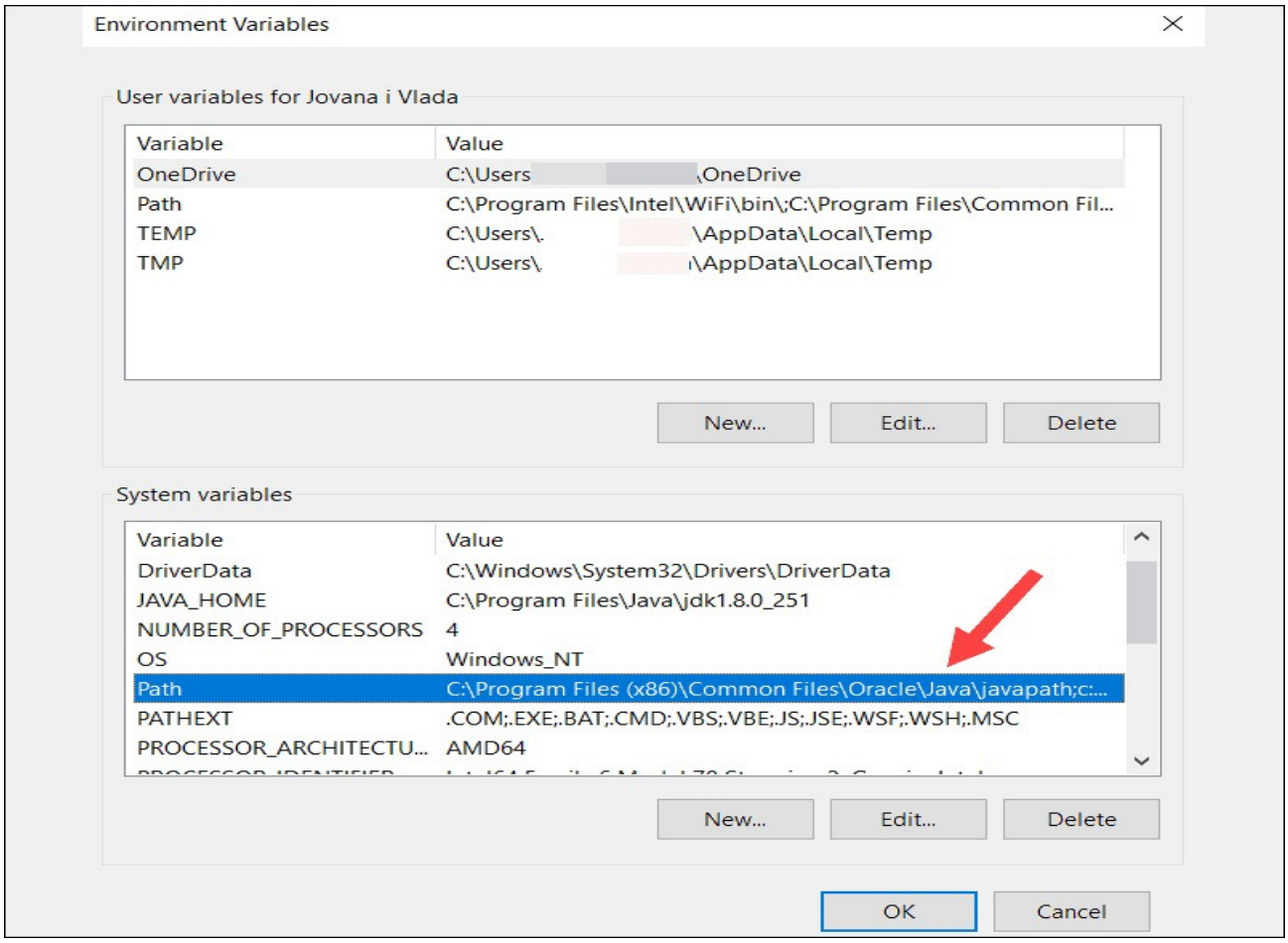


2. Select the Advanced system settings option.

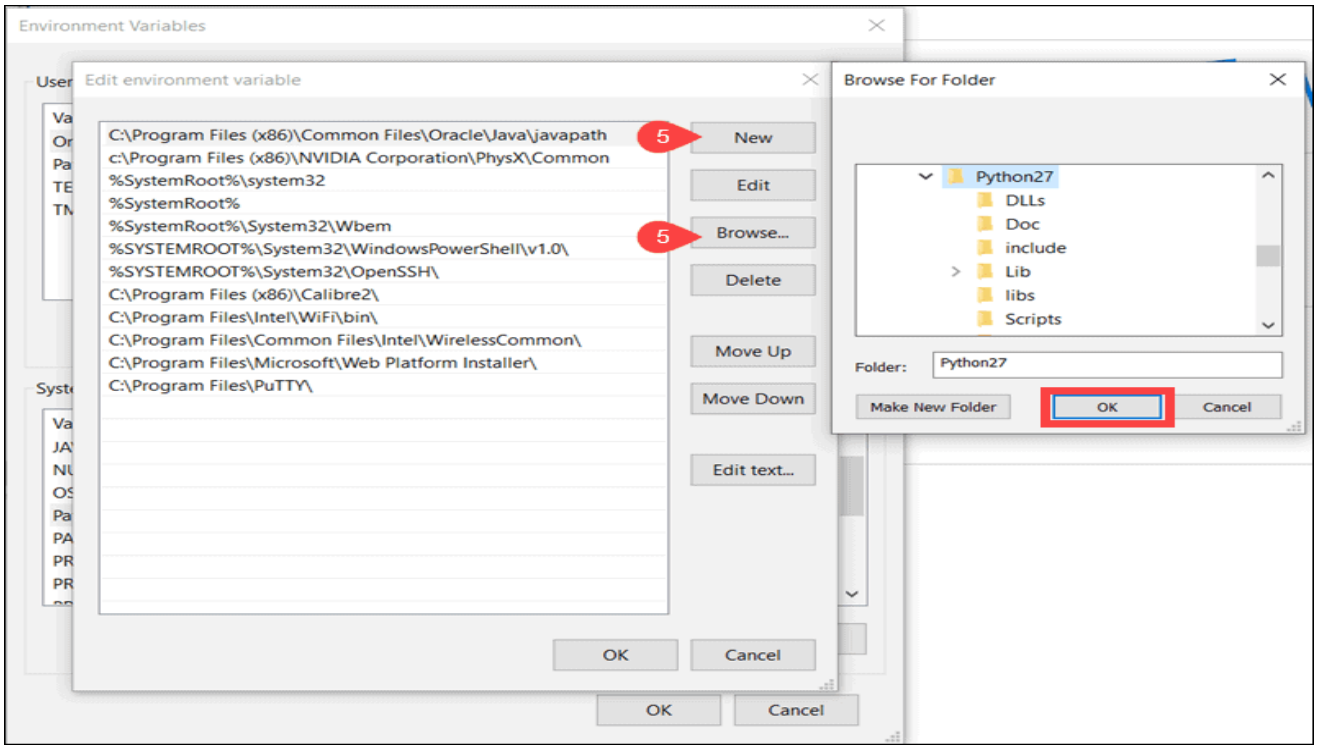
3. Click Environment Variables...



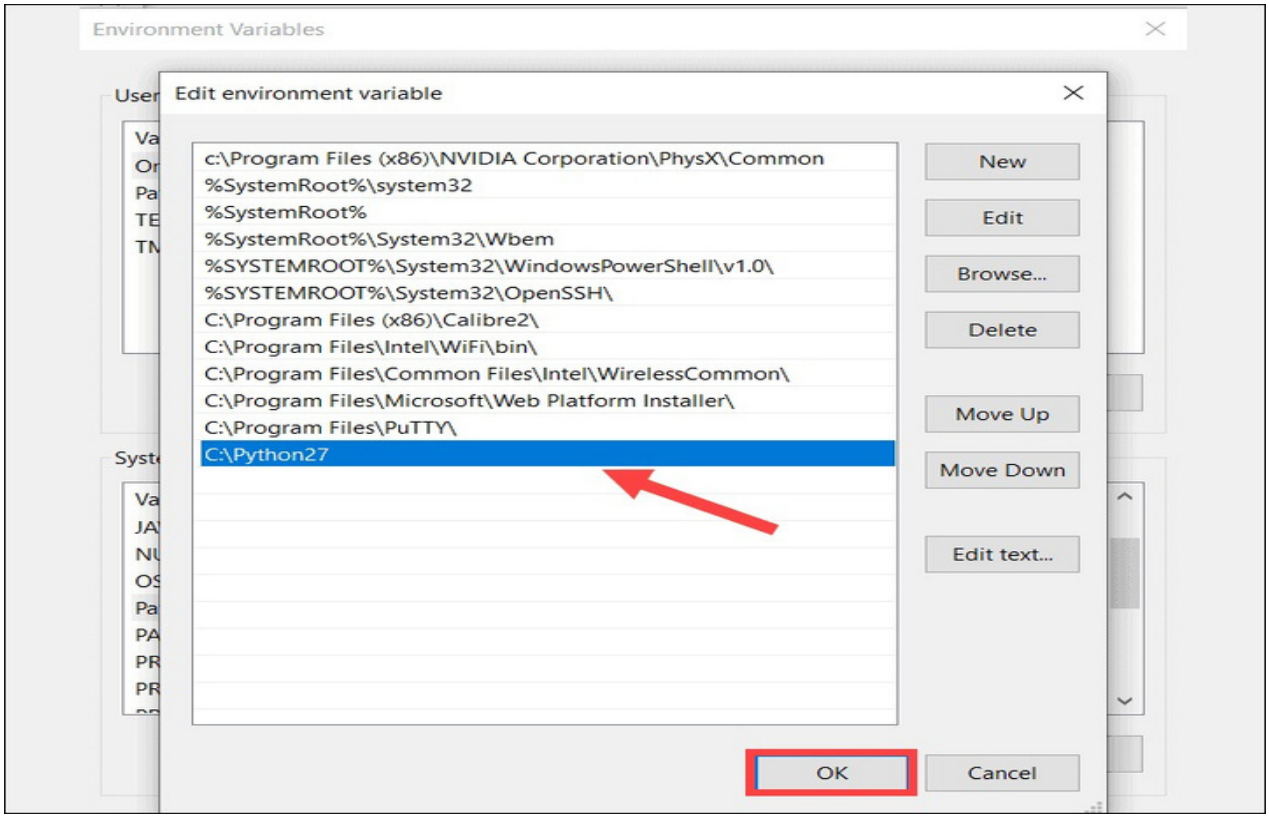
4. Double-click on the existing Path system variable.



5. Select New and then Browse to locate the Python installation folder quickly. Once you have confirmed that the path is correct, click OK.



6. Add the Python 2.7 path to the Path system variable by selecting OK.



Step 3: Download and Set Up Apache Cassandra

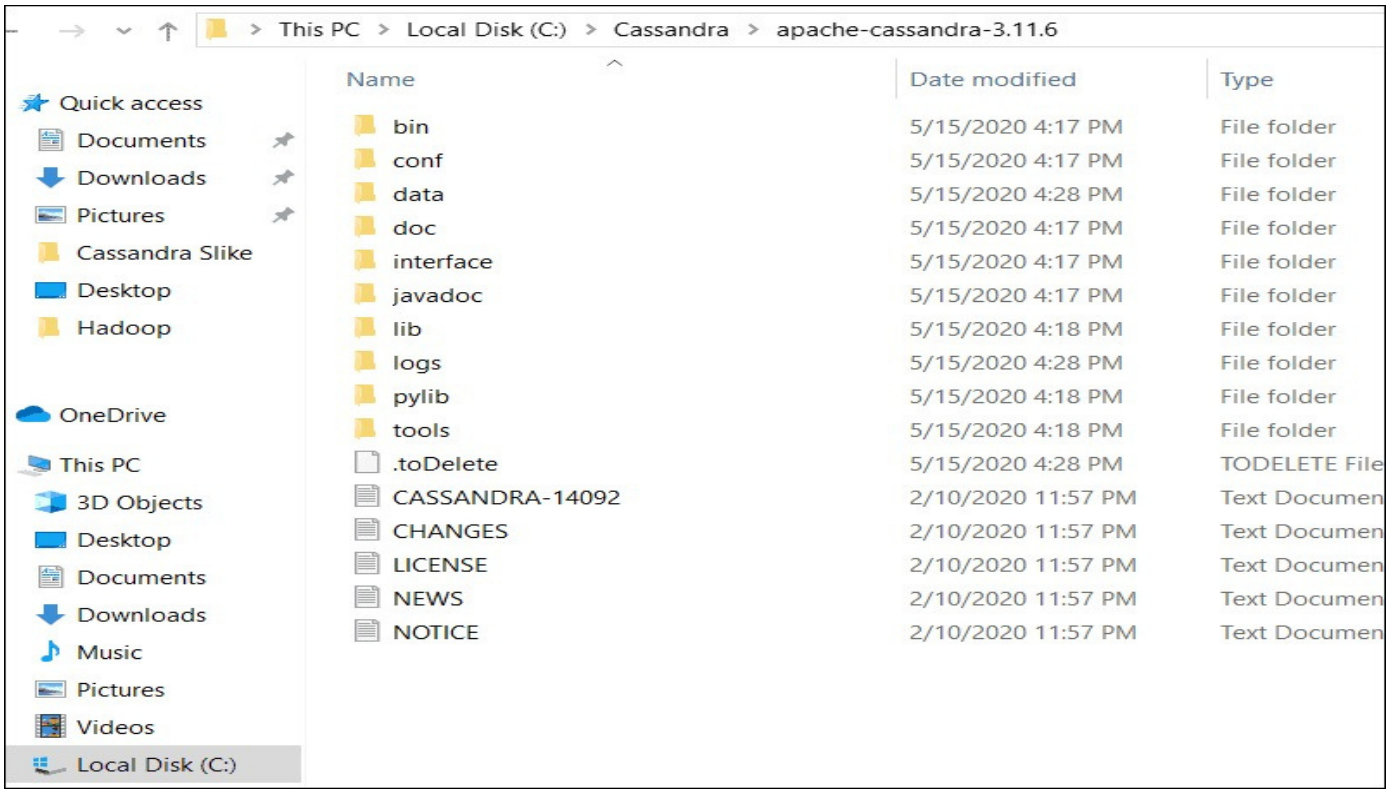
Download and Extract Cassandra tar.gz Folder

1. Visit the official Apache Cassandra Download page and select the version you would prefer to download. Currently, the latest available version is 3.11.6.

2. Click the suggested Mirror download link to start the download process.

Note: It is always recommended to verify downloads originating from mirror sites. The instructions for using GPG or SHA-512 for verification are usually available on the official download page.

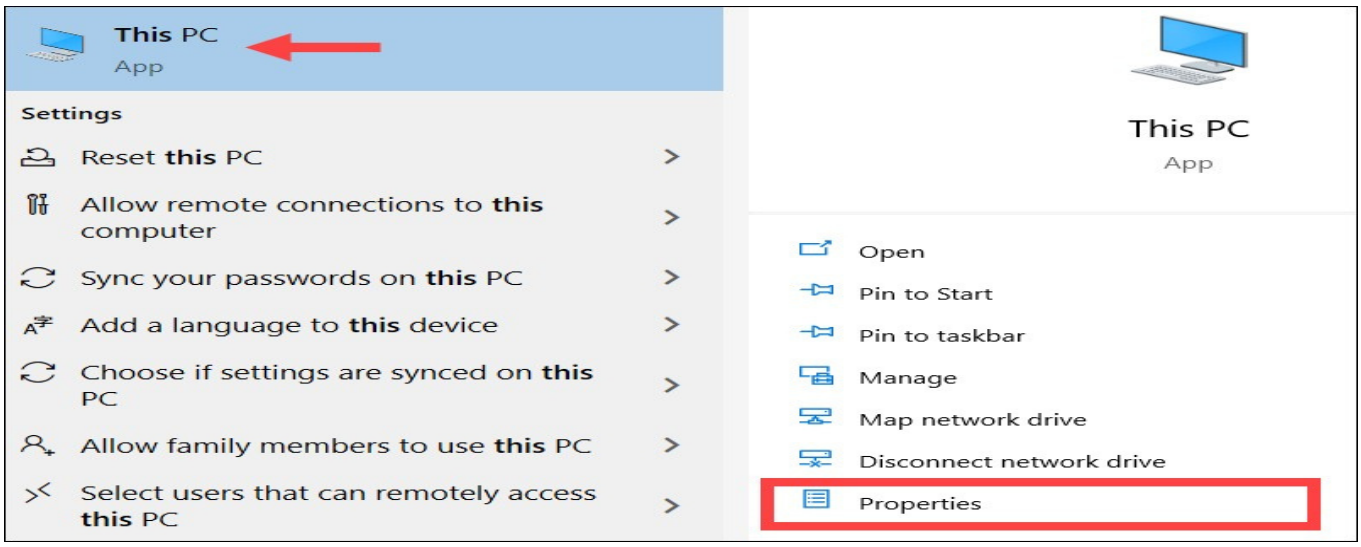
4. Unzip the compressed tar.gz folder using a compression tool such as 7-Zip or WinZip. In this example, the compressed folder was unzipped, and the content placed in the C:Cassandraapache- cassandra-3.11.6 folder.



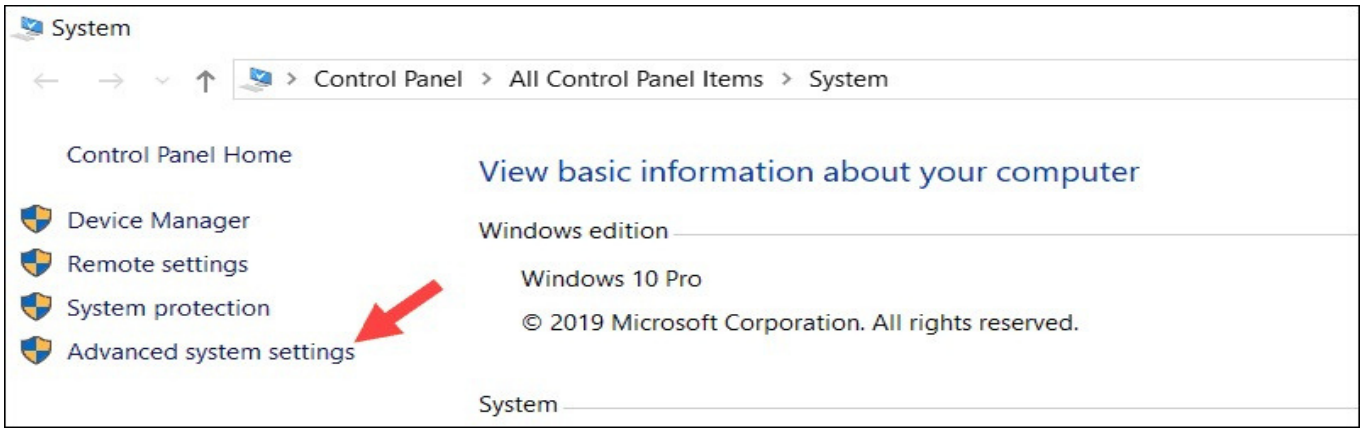
Configure Environment Variables for Cassandra

Set up the environment variables for Cassandra to enable the database to interact with other applications and operate on Windows.

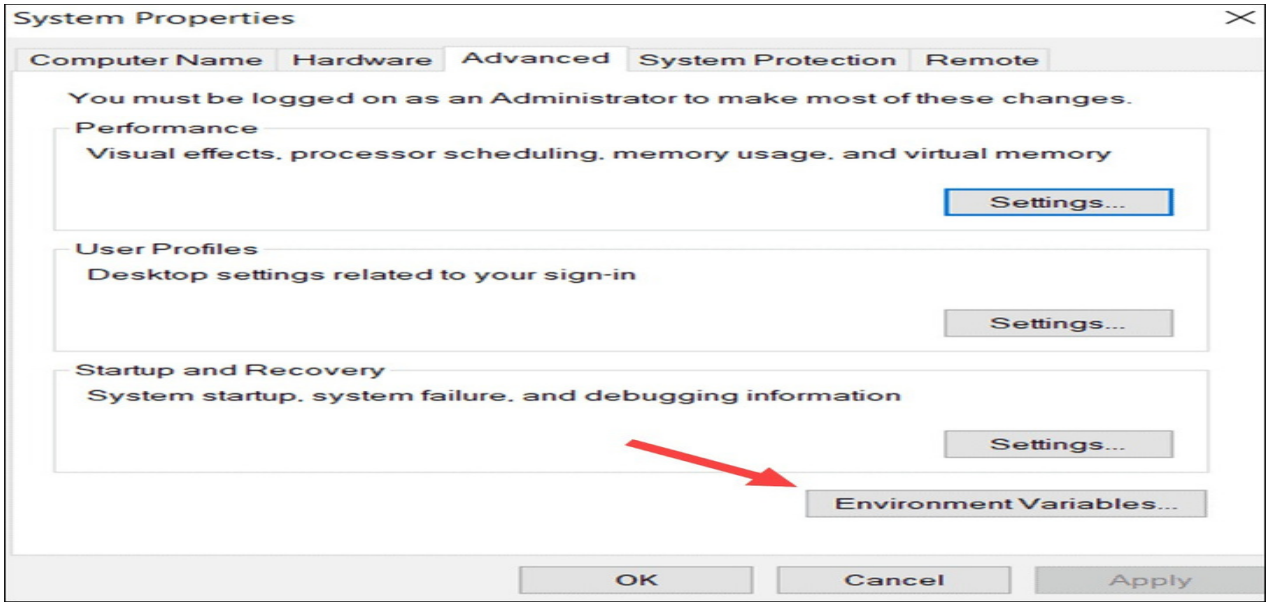
1. Go to This PC > Properties.



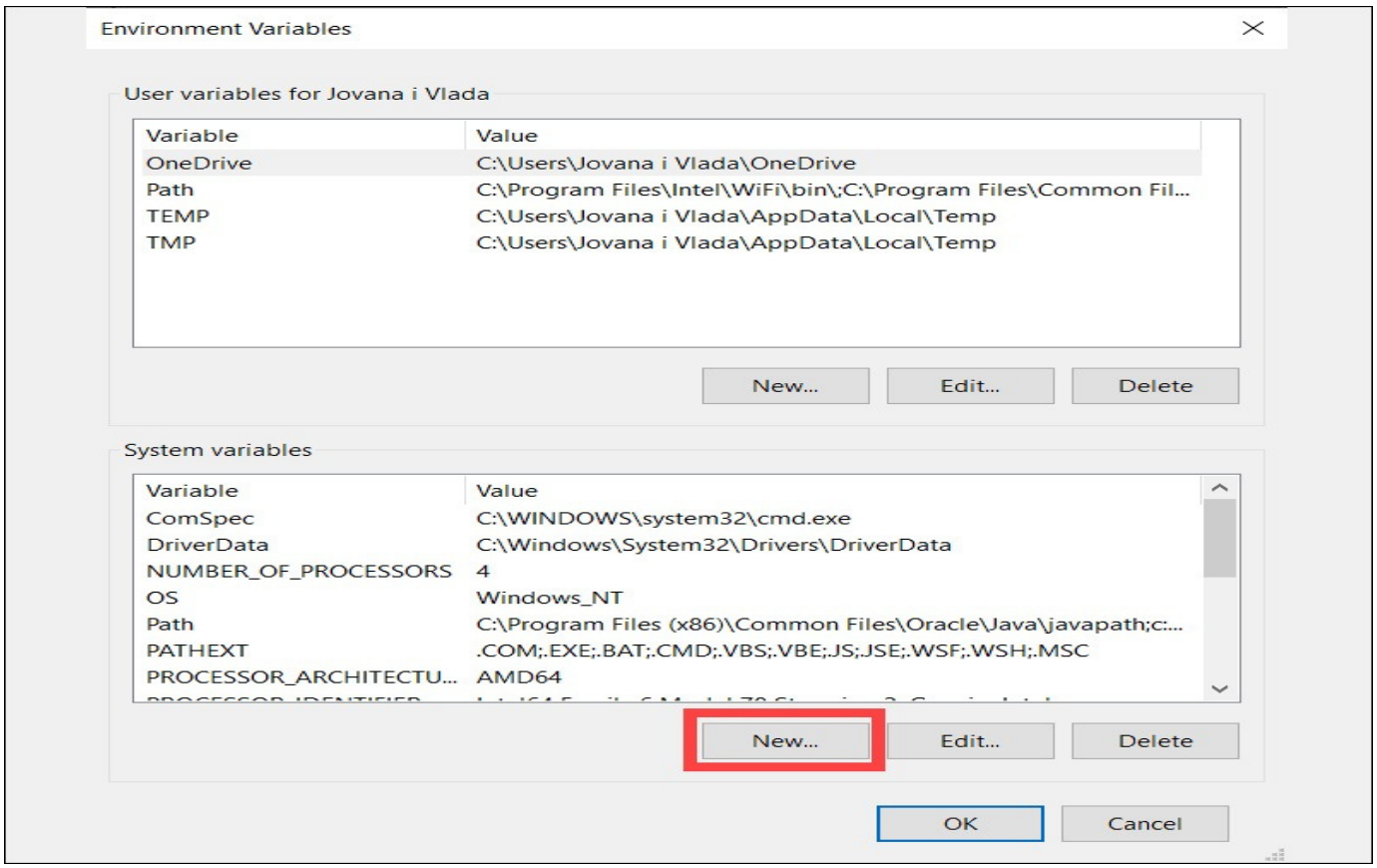
2. Go to Advanced system settings.



3. Click the Environment Variables... button.

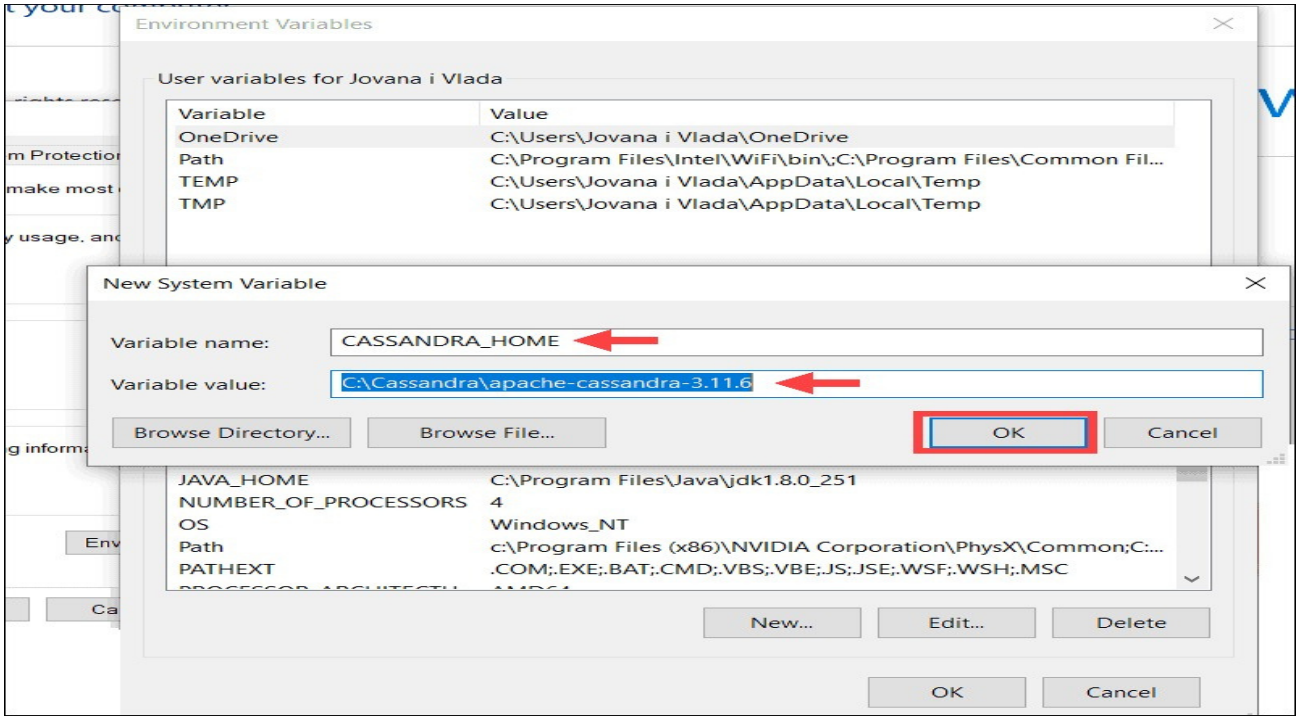


4. Add a completely new entry by selecting the New option.

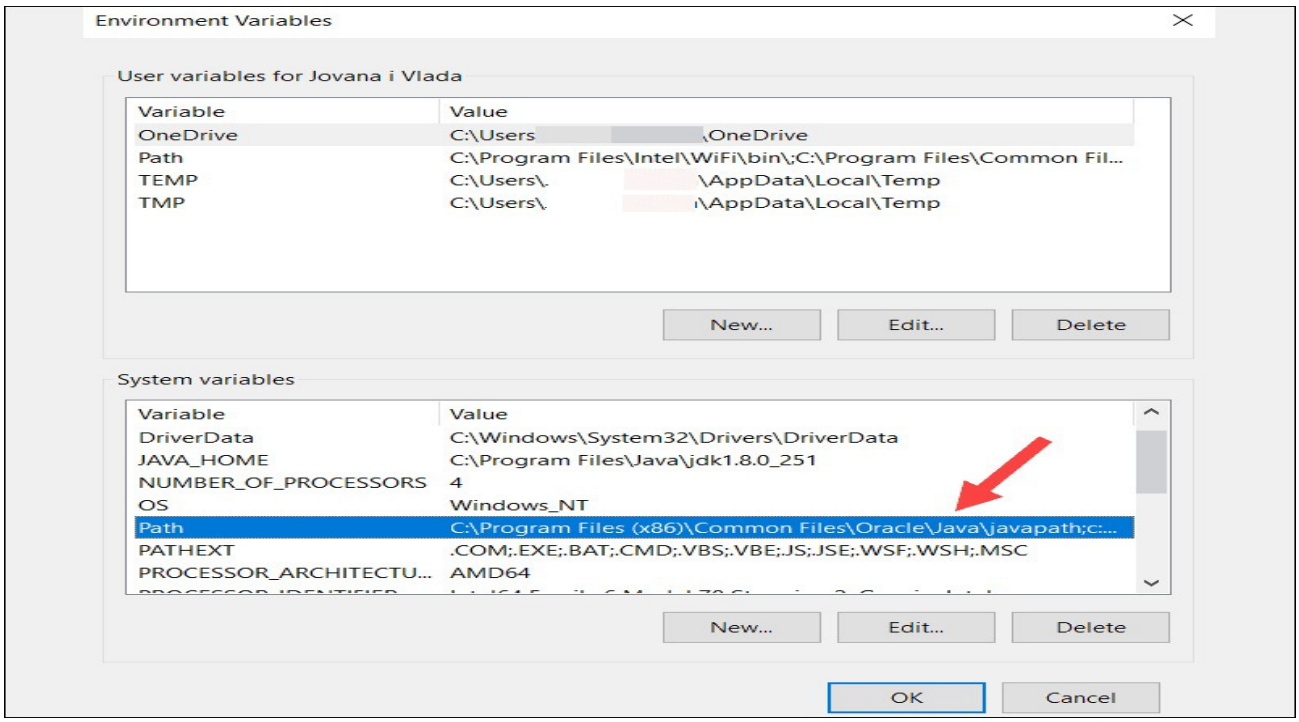


5. Type CASSANDRA_HOME for *Variable name*, then for the*Variable value* column select the location of the unzipped Apache Cassandra folder.

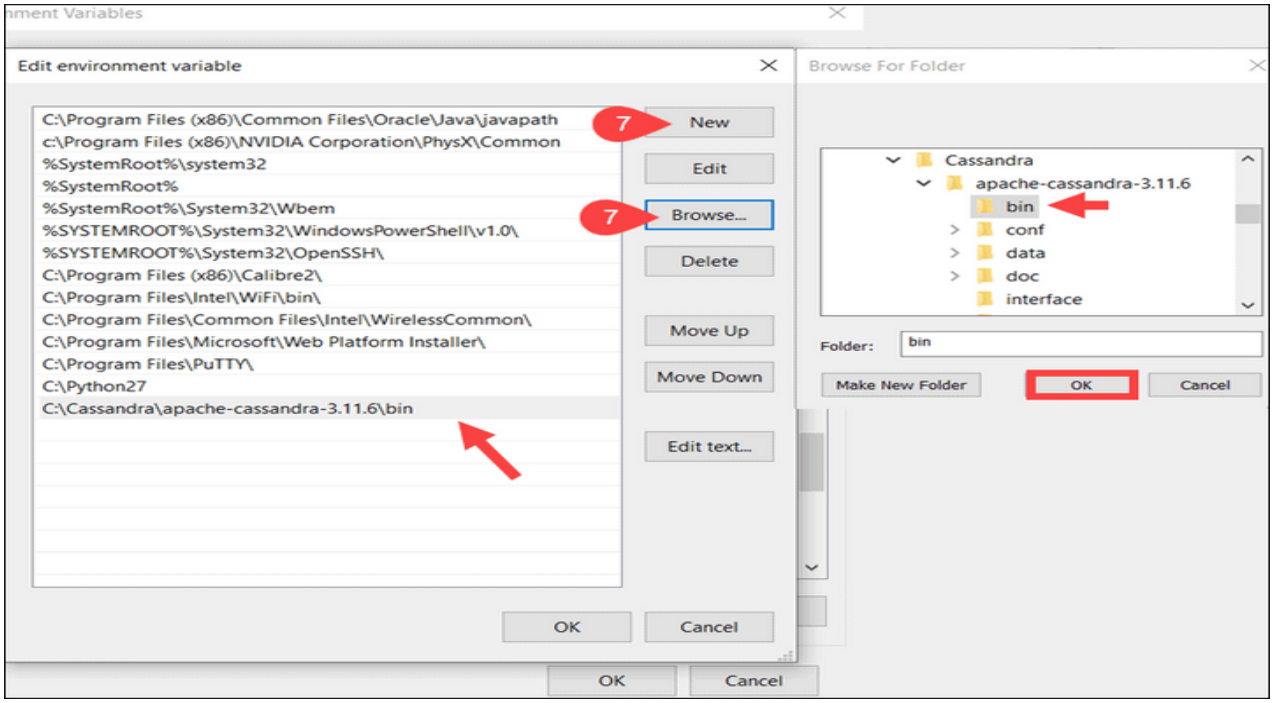
Based on the previous steps, the location is C:Cassandraapache-cassandra-3.11.6. Once you have confirmed that the location is correct, click OK.



6. Double click on the Path variable.



7. Select New and then Browse. In this instance, you need to add the full path to the bin folder located within the Apache Cassandra folder, C:Cassandraapache-cassandra-3.11.6bin.

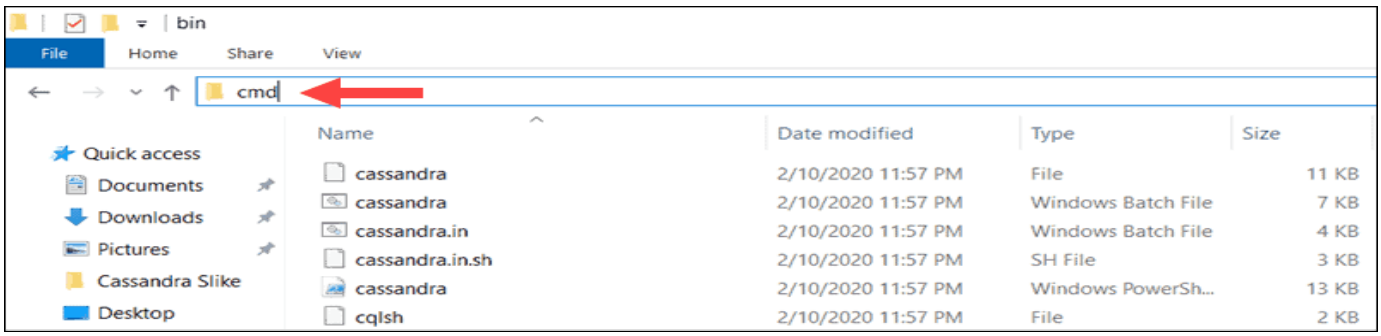


8. Hit the OK button and then again OK to save the edited variables.

Note: Check out our article to learn more about the difference between MongoDB and Cassandra.

Step 4: Start Cassandra from Windows CMD

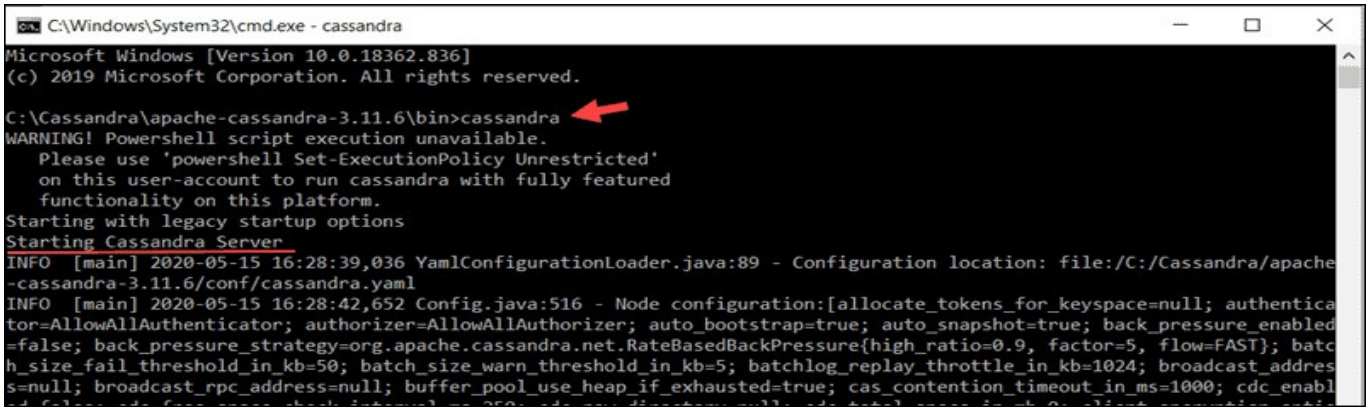
Navigate to the Cassandra bin folder. Start the Windows Command Prompt directly from within the bin folder by typing ***cmd*** in the address bar and pressing Enter.



Type the following command to start the Cassandra server:

```
cassandra
```

The system proceeds to start the Cassandra Server.



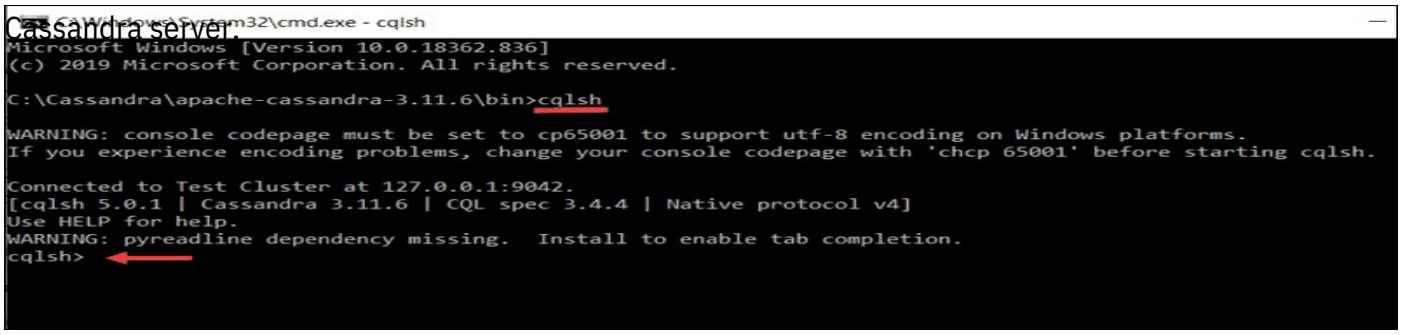
Do not close the current ***cmd*** session.

Step 5: Access Cassandra cqlsh from Windows CMD

While the initial command prompt is still running open a new command line prompt from the same bin folder. Enter the following command to access the Cassandra cqlsh bash shell:

```
cqlsh
```

You now have access to the Cassandra shell and can proceed to issue basic database commands to your Cassandra server.



Conclusion

You have successfully installed Cassandra on Windows.

Ex. No: 8 Date:	Create database in Casandra using – Create, Alter and Drop. Add records using Insert, Update, Delete and Truncate.

Aim:

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

Create Keyspace

A keyspace is an object that is used to hold column families, user defined types. A keyspace is like RDBMS database which contains column families, indexes, user defined types, data center awareness, strategy used in keyspace, replication factor, etc.
In Cassandra, "Create Keyspace" command is used to create keyspace.

Syntax:

CREATE KEYSPACE <identifier> WITH <properties>
Or
Create keyspace KeyspaceName with replicaton={'class':strategy name,
'replication_factor': No of replications on different nodes}

Different components of Cassandra Keyspace

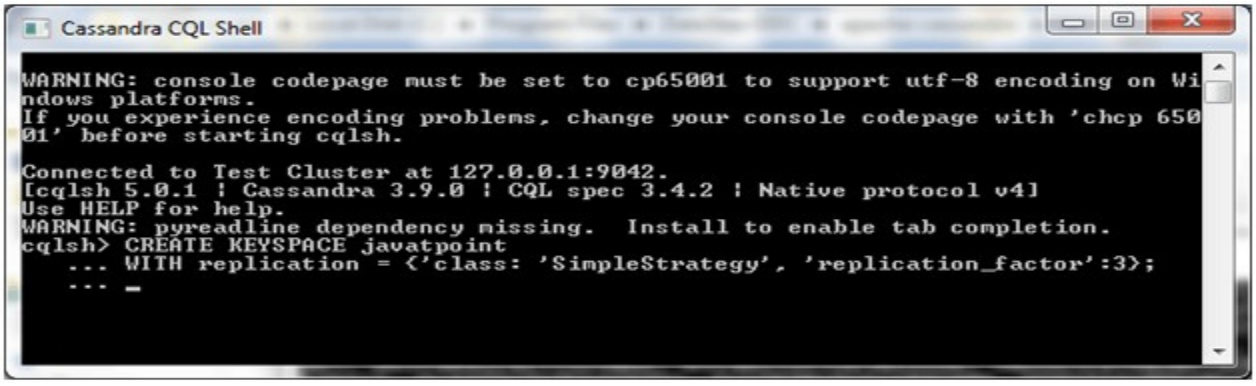
Strategy: There are two types of strategy declaration in Cassandra syntax:

- **Simple Strategy:** Simple strategy is used in the case of one data center. In this strategy, the first replica is placed on the selected node and the remaining nodes are placed in clockwise direction in the ring without considering rack or node location.
- **Network Topology Strategy:** This strategy is used in the case of more than one data centers. In this strategy, you have to provide replication factor for each data center separately.

Replication Factor: Replication factor is the number of replicas of data placed on different nodes. More than two replication factor are good to attain no single point of failure. So, 3 is good replication factor.

Example:

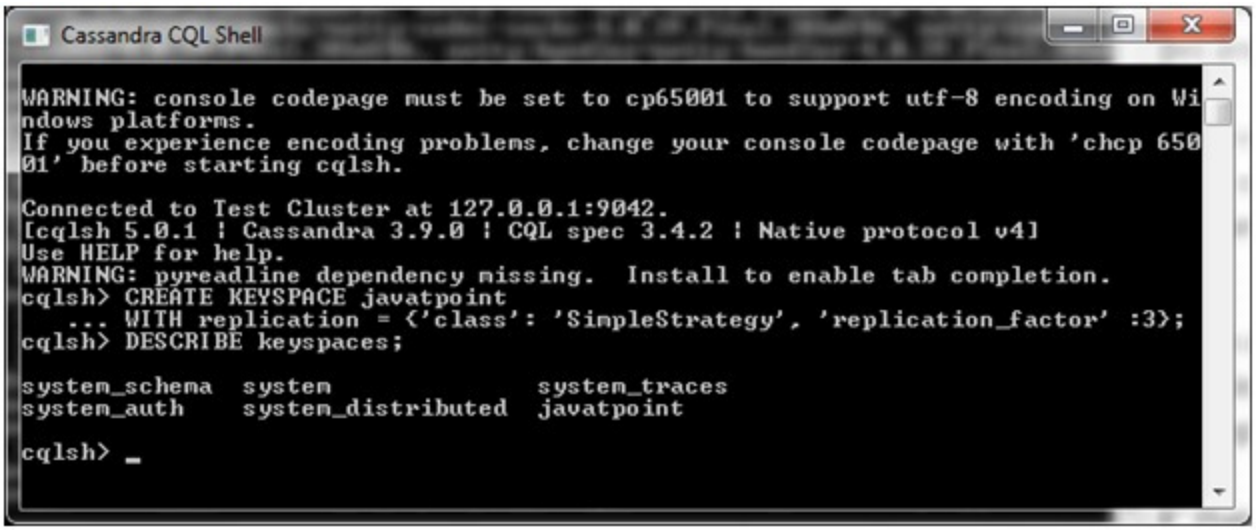
Let's take an example to create a keyspace named "javatpoint".
CREATE KEYSPACE javatpoint
WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 3};



Keyspace is created now.

Verification:

To check whether the keyspace is created or not, use the "DESCRIBE" command. By using this command you can see all the keyspaces that are created.



There is another property of CREATE KEYSPACE in Cassandra.

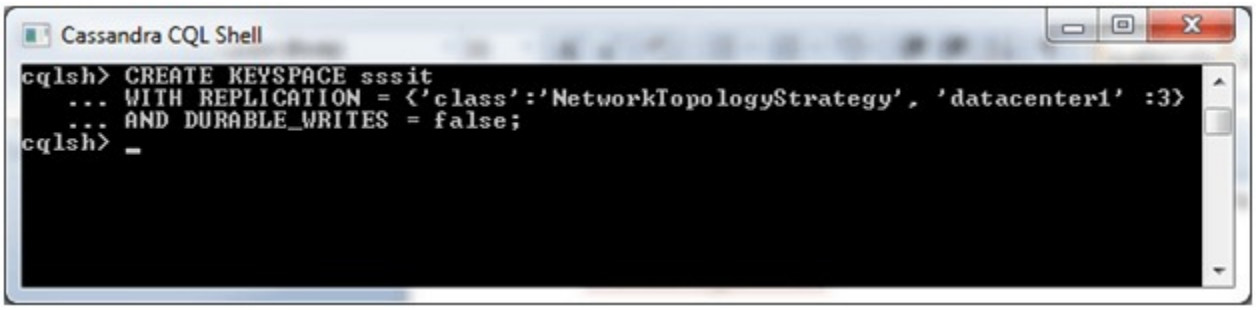
Durable_writes

By default, the durable_writes properties of a table is set to true, you can also set this property to false. But, this property cannot be set to simplex strategy.

Example:

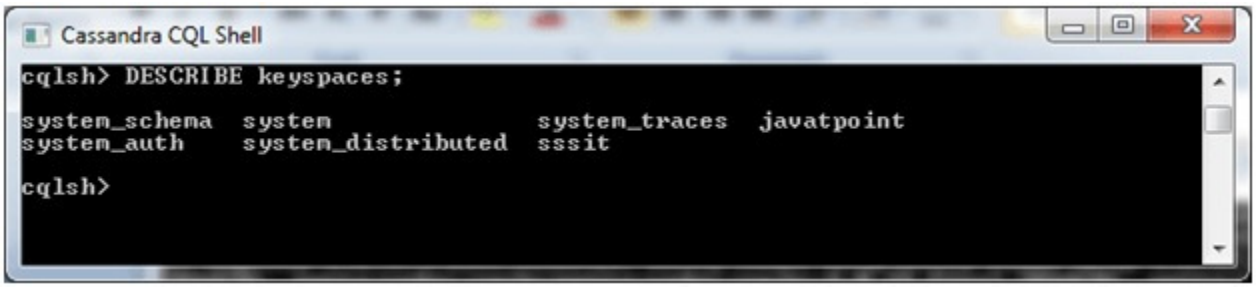
Let's take an example to see the usage of durable_write property.

```
CREATE KEYSPACE sssit
WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy', 'datacenter1' : 3 }
AND DURABLE_WRITES = false;
```



Verification:

To check whether the keyspace is created or not, use the "DESCRIBE" command. By using this command you can see all the keyspaces that are created.



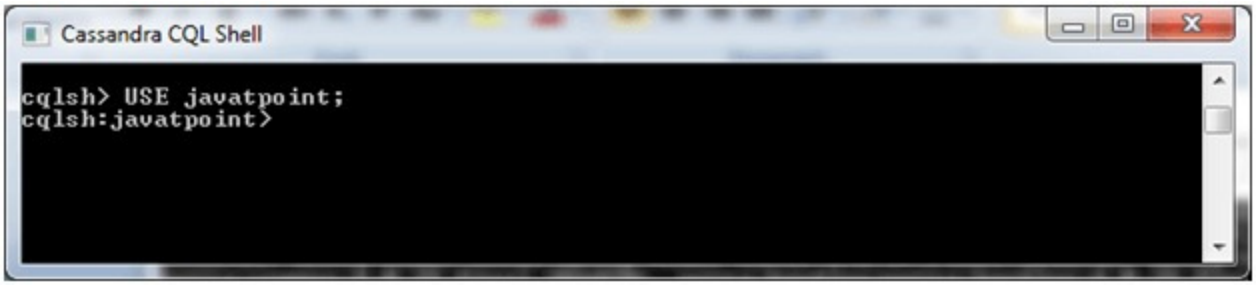
Using a Keyspace
To use the created keyspace, you have to use the USE command.

Syntax:

USE <identifier>

See this example:

Here, we are using javatpoint keyspace.



Cassandra Alter Keyspace

The "ALTER keyspace" command is used to alter the replication factor, strategy name and durable writes properties in created keyspace in Cassandra.

Syntax:

ALTER KEYSPACE <identifier> WITH <properties>
Or
ALTER KEYSPACE "KeySpace Name"
WITH replication = {'class': 'Strategy name', 'replication_factor' : 'No.Of replicas'};
Or
Alter Keyspace KeyspaceName with replication={'class':'StrategyName',
'replication_factor': no of replications on different nodes}
with DURABLE_WRITES=true/false

Main points while altering Keyspace in Cassandra

o **Keyspace Name:** Keyspace name cannot be altered in Cassandra.
o **Strategy Name:** Strategy name can be altered by using a new strategy name.
o **Replication Factor:** Replication factor can be altered by using a new replication factor.
o **DURABLE_WRITES:** DURABLE_WRITES value can be altered by specifying its value true/false. By default, it is true. If set to false, no updates will be written to the commit log and vice versa.

Example:

Let's take an example to demonstrate "Alter Keyspace". This will alter the keyspace strategy from 'SimpleStrategy' to 'NetworkTopologyStrategy' and replication factor from 3 to 1 for DataCenter1.

ALTER KEYSPACE javatpoint
WITH replication = {'class':'NetworkTopologyStrategy', 'replication_factor' : 1};

Cassandra Drop Keyspace

In Cassandra, "DROP Keyspace" command is used to drop keyspaces with all the data, column families, user defined types and indexes from Cassandra.

Cassandra takes a snapshot of keyspace before dropping the keyspace. If keyspace does not exist in the Cassandra, Cassandra will return an error unless IF EXISTS is used.

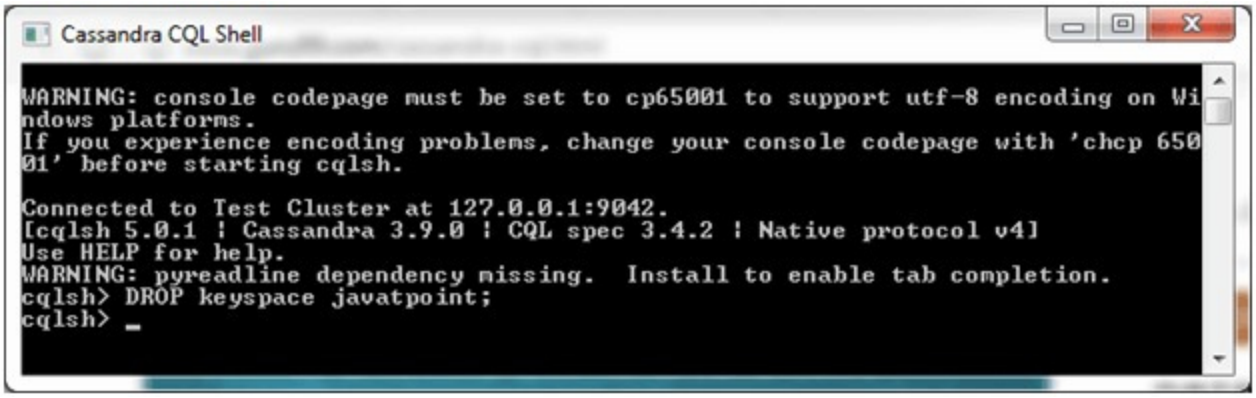
Syntax:

DROP keyspace KeyspaceName ;

Example:

Let's take an example to drop the keyspace named "javatpoint".

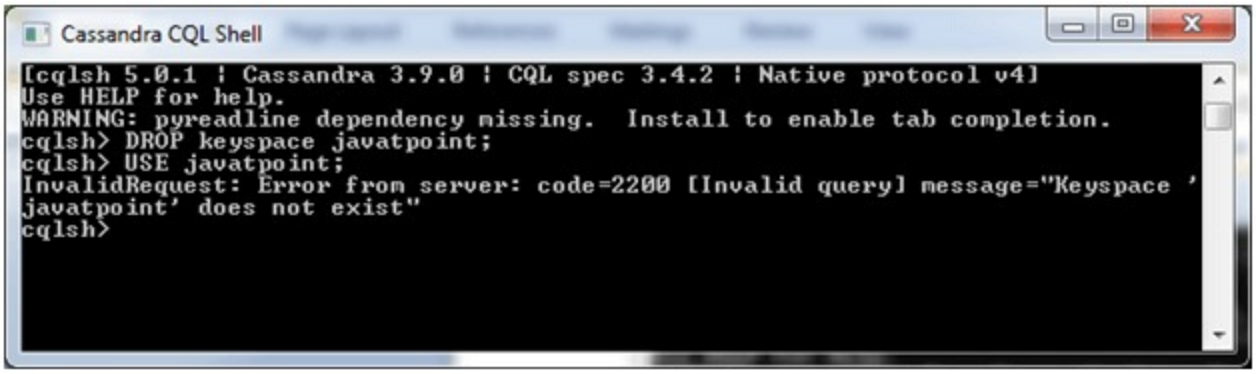
DROP keyspace javatpoint;



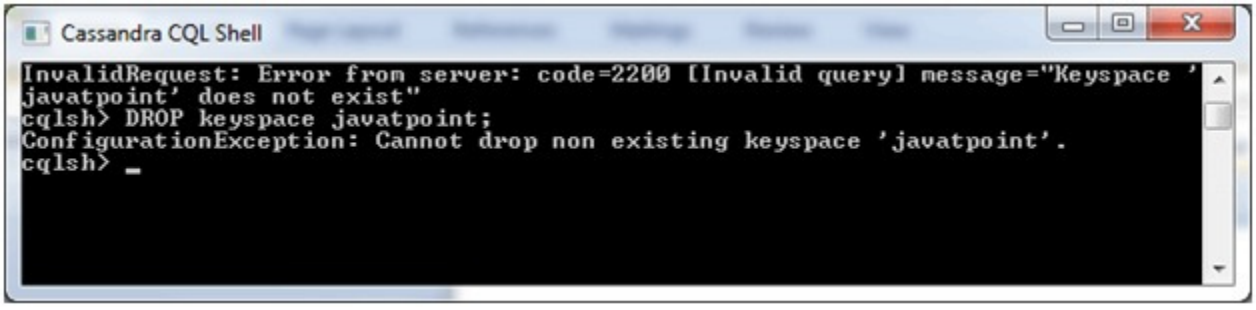
Verification:

After the execution of the above command the keyspace "javatpoint" is dropped from Cassandra with all the data and schema.

You can verify it by using the "USE" command.



Now you can see that "javatpoint" keyspace is dropped. If you use "DROP" command again, you will get the following message.



Cassandra Create Table

In Cassandra, CREATE TABLE command is used to create a table. Here, column family is used to store data just like table in RDBMS.

So, you can say that CREATE TABLE command is used to create a column family in Cassandra.

Syntax:

CREATE (TABLE | COLUMNFAMILY) <tablename>
('<column-definition>' , '<column-definition>')
(WITH <option> AND <option>)

Or

For declaring a primary key:

CREATE TABLE tablename(
column1 name datatype PRIMARYKEY,
column2 name data type,
column3 name data type.
)

You can also define a primary key by using the following syntax:

Create table TableName
(
ColumnName DataType,
ColumnName DataType,
ColumnName DataType
.
.
.
Primary key(ColumnName)
) with PropertyName=PropertyValue;

There are two types of primary keys:

Single primary key: Use the following syntax for single primary key.

1. Primary key (ColumnName)

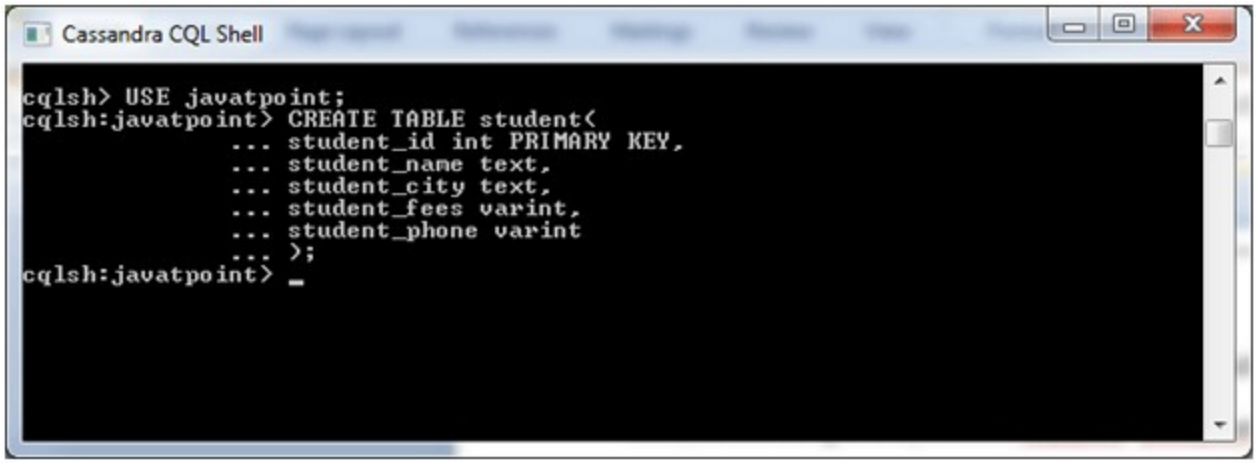
Compound primary key: Use the following syntax for single primary key.

1. Primary key(ColumnName1,ColumnName2 . . .)

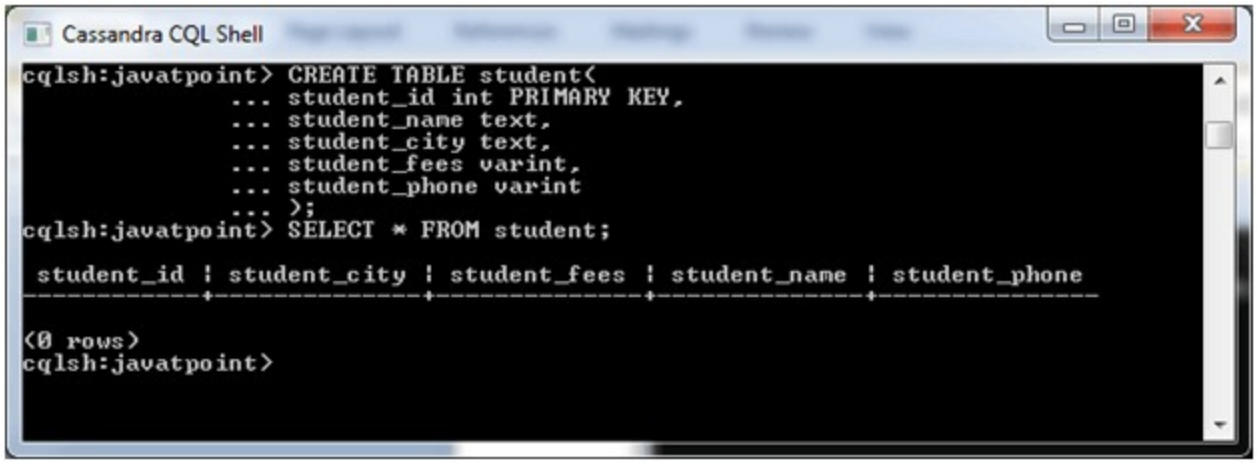
Example:

Let's take an example to demonstrate the CREATE TABLE command.
Here, we are using already created Keyspace "javatpoint".

CREATE TABLE student(
student_id int PRIMARY KEY,
student_name text,
student_city text,
student_fees varint,
student_phone varint
);



The table is created now. You can check it by using the following command.
SELECT * FROM student;



Cassandra Alter Table

ALTER TABLE command is used to alter the table after creating it. You can use the ALTER command to perform two types of operations:

o Add a column
o Drop a column

Syntax:

ALTER (TABLE | COLUMNFAMILY) <tablename> <instruction>

Adding a Column

You can add a column in the table by using the ALTER command. While adding column, you have to aware that the column name is not conflicting with the existing column names and that the table is not defined with compact storage option.

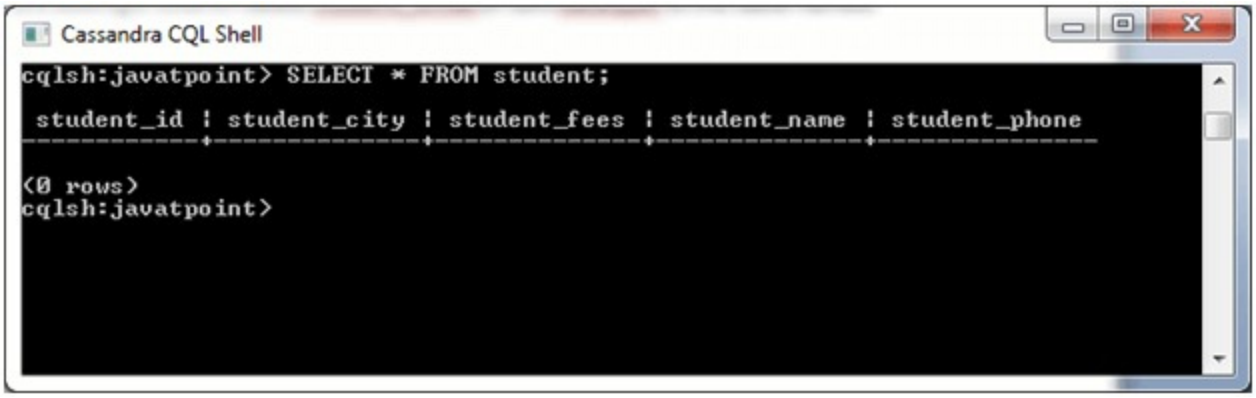
Syntax:

ALTER TABLE table name
ADD new column datatype;

Example:

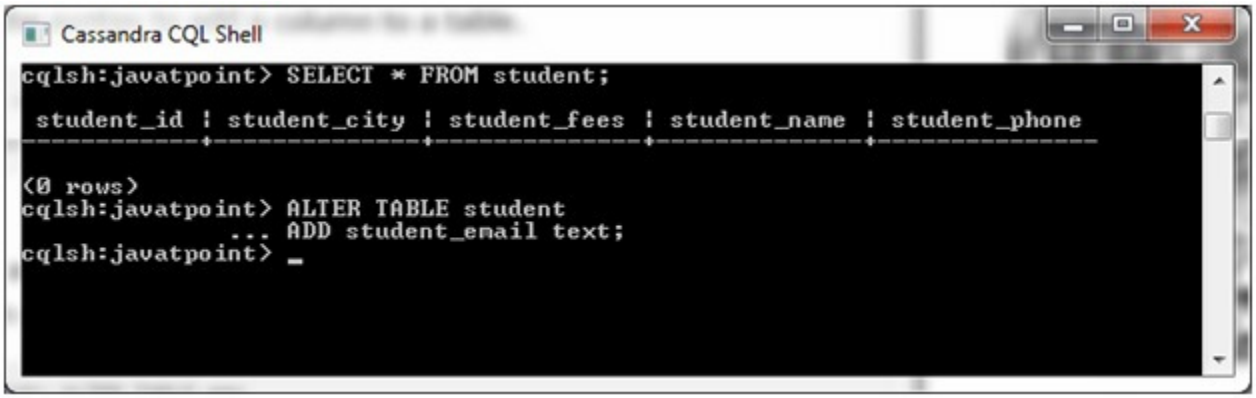
Let's take an example to demonstrate the ALTER command on the already created table named "student". Here we are adding a column called student_email of text datatype to the table named student.

Prior table:

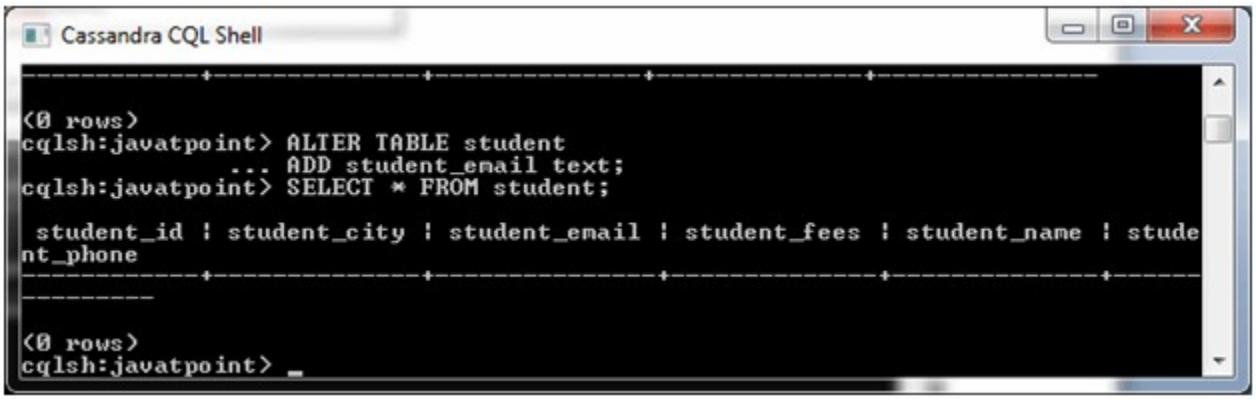


After using the following command:

ALTER TABLE student
ADD student_email text;



A new column is added. You can check it by using the SELECT command.



Dropping a Column

You can also drop an existing column from a table by using ALTER command. You should check that the table is not defined with compact storage option before dropping a column from a table.

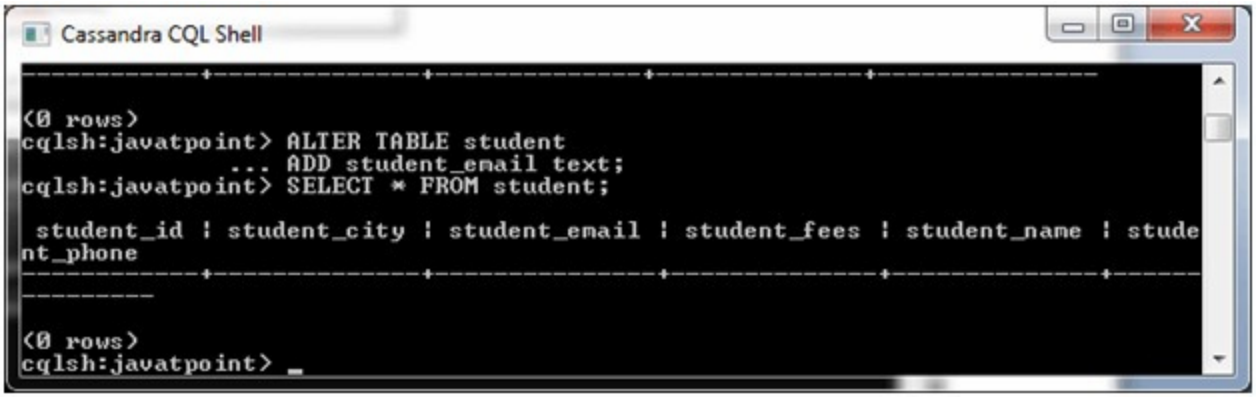
Syntax:

ALTER table name
DROP column name;

Example:

Let's take an example to drop a column named student_email from a table named student.

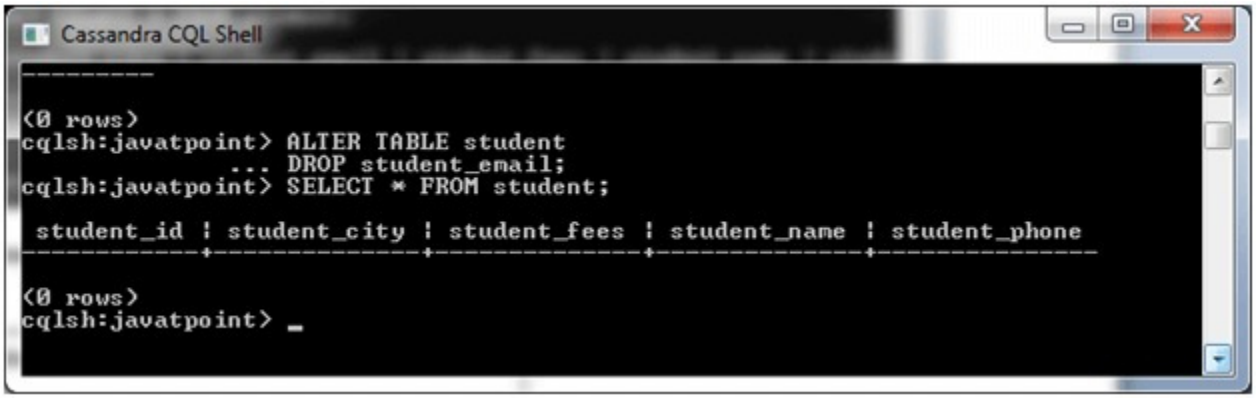
Prior table:



After using the following command:

ALTER TABLE student

DROP student_email;



Now you can see that a column named "student_email" is dropped now.

If you want to drop the multiple columns, separate the columns name by ",".

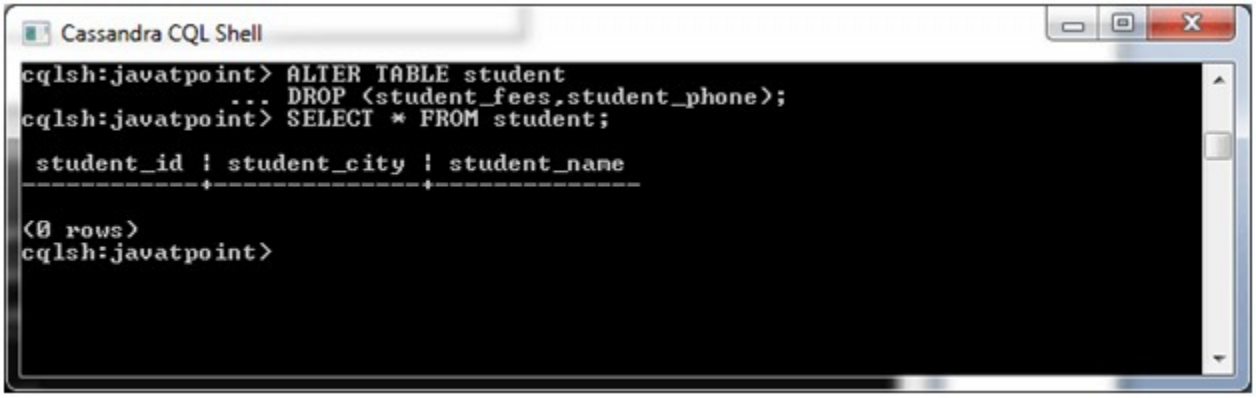
See this example:

Here we will drop two columns student_fees and student_phone.

ALTER TABLE student

DROP (student_fees, student_phone);

Output:



Cassandra DROP table

DROP TABLE command is used to drop a table.

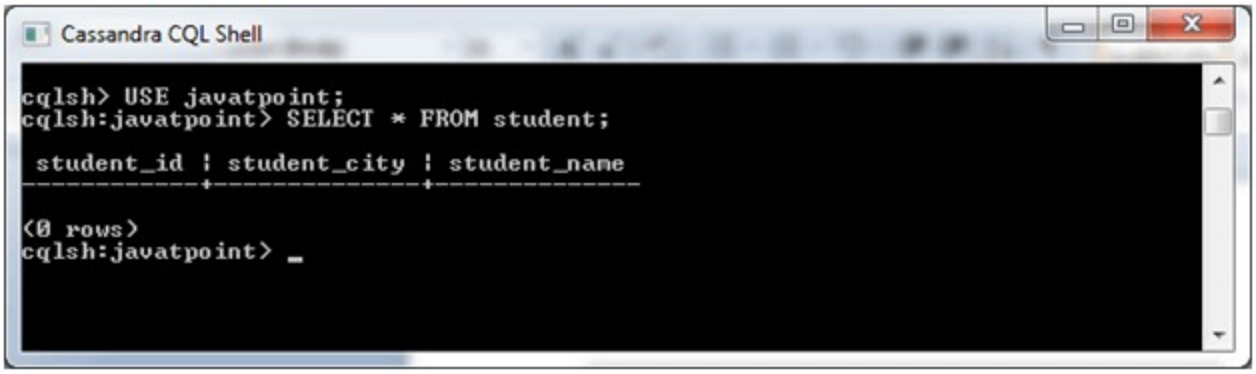
Syntax:

DROP TABLE <tablename>

Example:

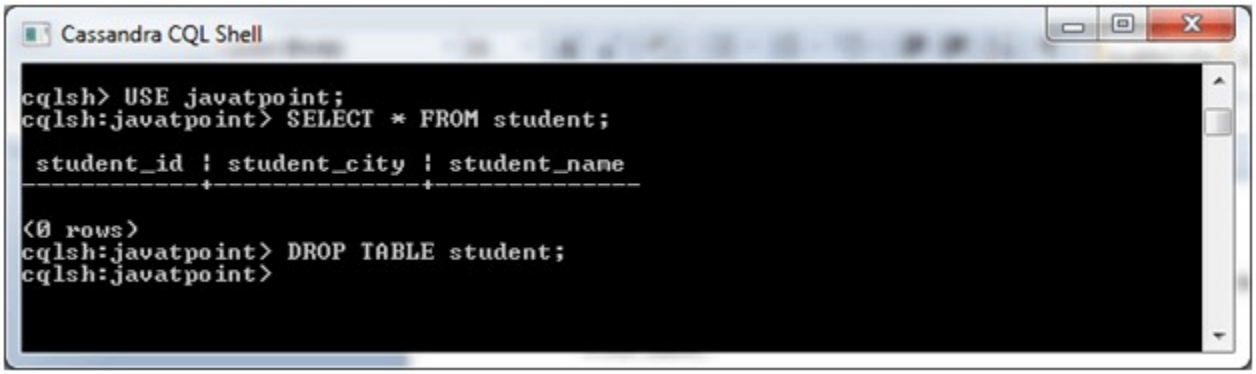
Let's take an example to demonstrate how to drop a table. Here, we drop the student table.

Prior table:



After using the following command:

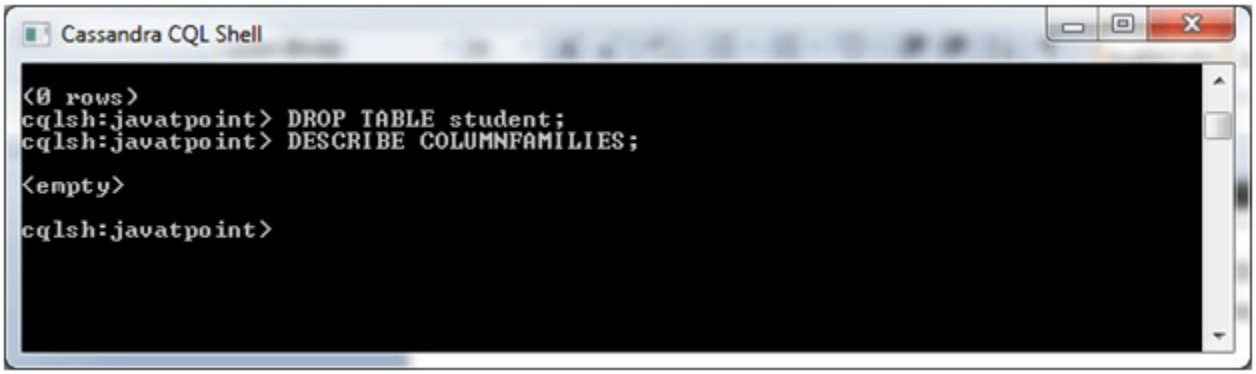
DROP TABLE student;



The table named "student" is dropped now. You can use DESCRIBE command to verify if the table is deleted or not. Here the student table has been deleted; you will not find it in the column families list.

DESCRIBE COLUMNFAMILIES;

Output:



Cassandra Truncate Table

TRUNCATE command is used to truncate a table. If you truncate a table, all the rows of the table are deleted permanently.

Syntax:

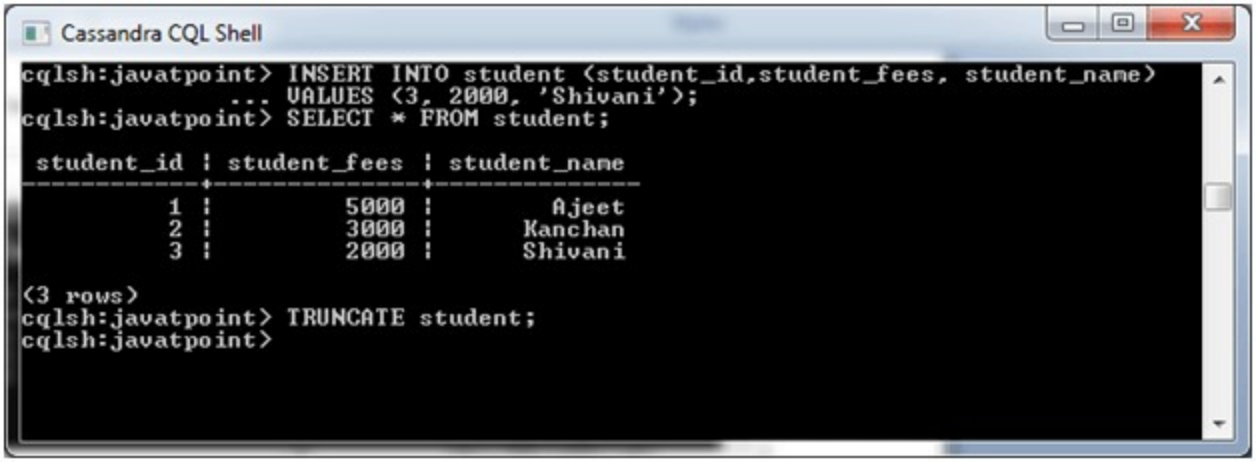
TRUNCATE <tablename>

Example:

We have a table named "student" having the following data:

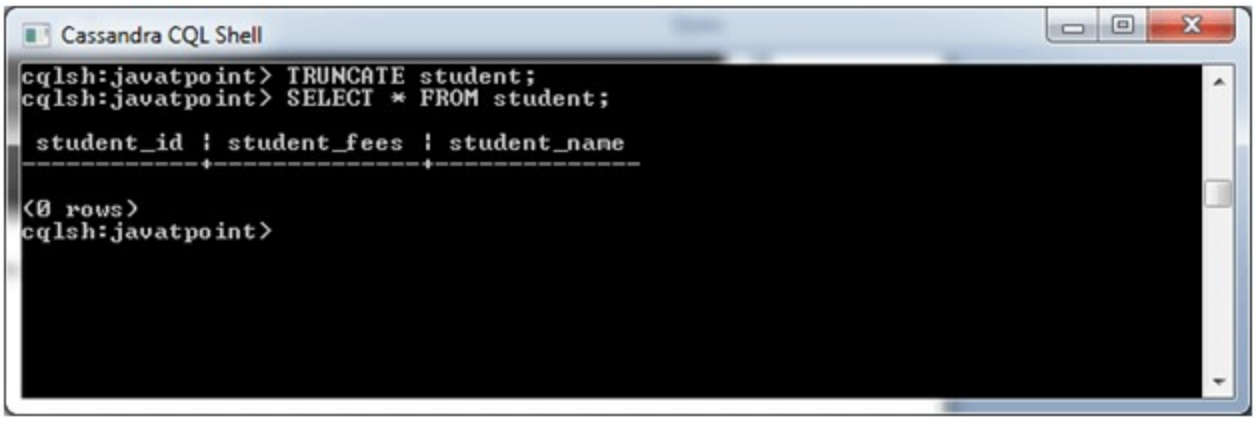
Now, we use TRUNCATE command:

TRUNCATE student;



Now, the table is truncated. You can verify it by using SELECT command.

SELECT * FROM student;



You can see that table is truncated now.

Cassandra Create Index

CREATE INDEX command is used to create an index on the column specified by the user. If the data already exists for the column which you choose to index, Cassandra creates indexes on the data during the 'create index' statement execution.

Syntax:

CREATE INDEX <identifier> ON <tablename>

Rules for creating Index

o The index cannot be created on primary key as a primary key is already indexed.

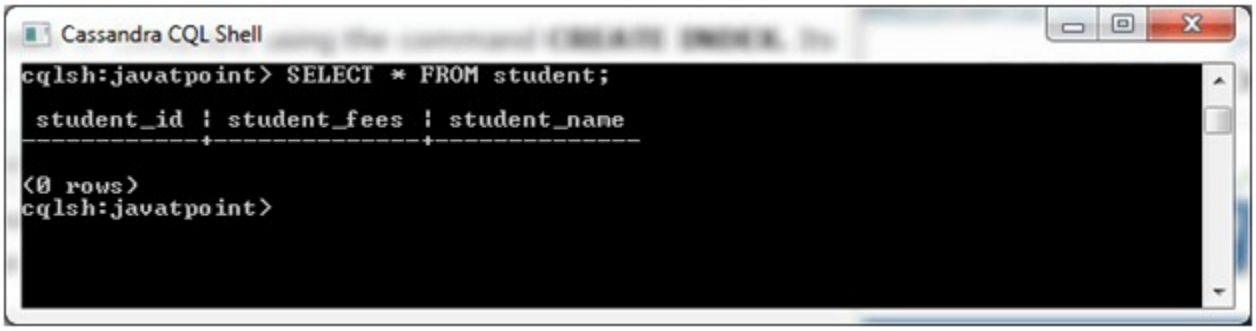
o In Cassandra, Indexes on collections are not supported.

o Without indexing on the column, Cassandra can't filter that column unless it is a primary key.

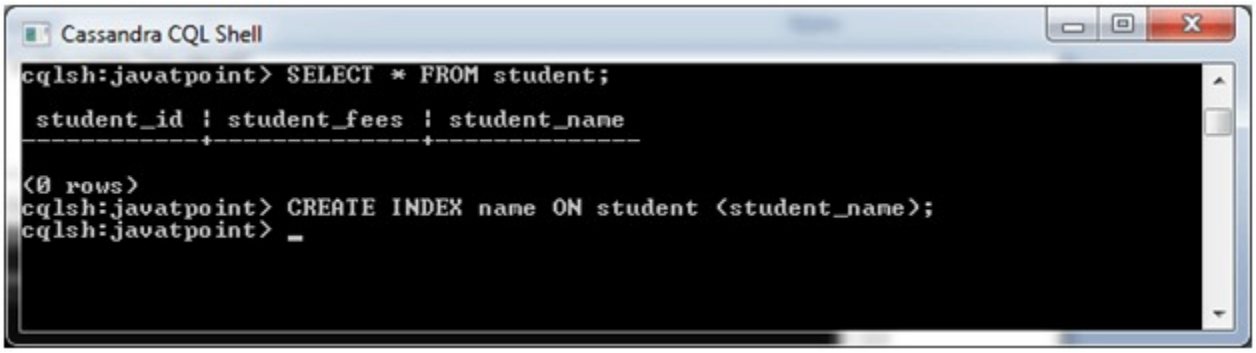
Example:

Let's take an example to demonstrate how to create index on a column. Here we create an index to a column "student_name" in the table "student".

Prior table:



CREATE INDEX name ON student (student_name);



Index is created on the column "student_name".

Cassandra DROP Index

DROP INDEX command is used to drop a specified index. If the index name was not specified during index creation, then index name is TableName_ColumnName_idx.

Syntax:

DROP INDEX <identifier>

Or

Drop index IF EXISTS KeyspaceName.IndexName

Rules for dropping an Index

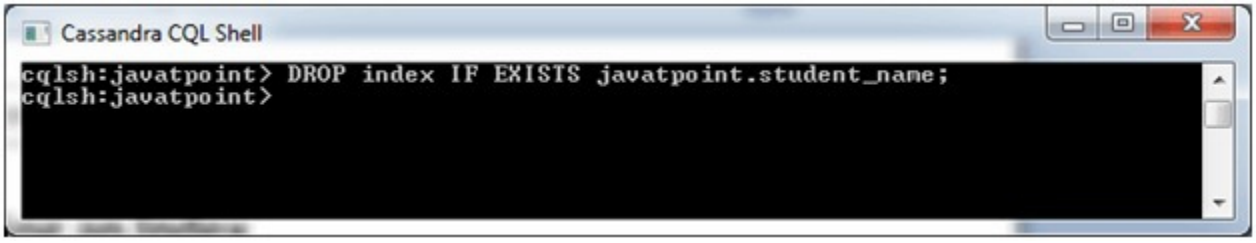
o If the index does not exist, it will return an error unless you use IF EXISTS which returns no operation.

o During index creation, you have to specify keyspace name with the index name otherwise index will be dropped from the current keyspace.

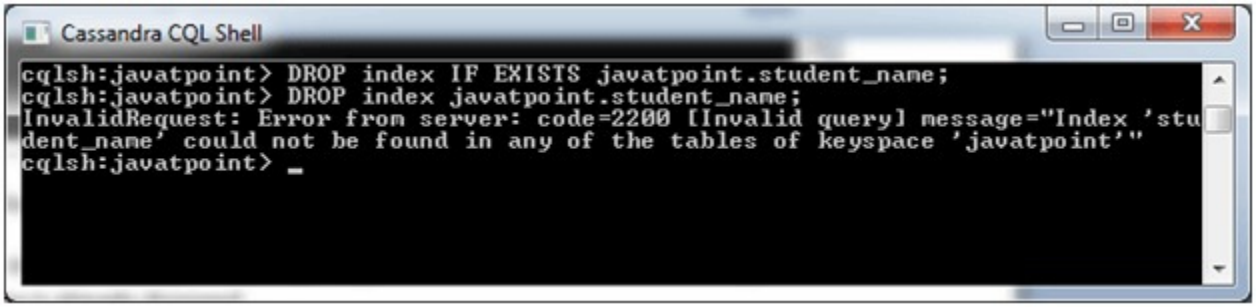
Example:

Let's take an example to demonstrate how to drop index on a column. Here we drop the created index to a column "student_name" in the table "student". The name of the keyspace is "javatpoint".

Drop index IF EXISTS javatpoint.student_name



Index is dropped from the column "student_name".



Ex. No: 9 Date:	Exercise based on Cassandra Query Language i.e. selecting records, select records with specific conditions

Aim:

To perform query based on Cassandra Query Language i.e. selecting records, select records with specific conditions.

Cassandra Create Data

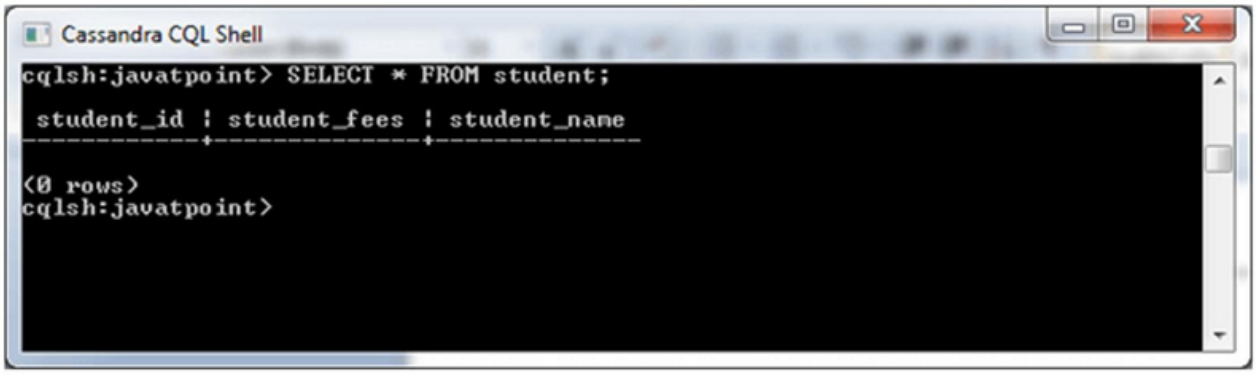
INSERT command is used to insert data into the columns of the table.

Syntax:

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

Example:

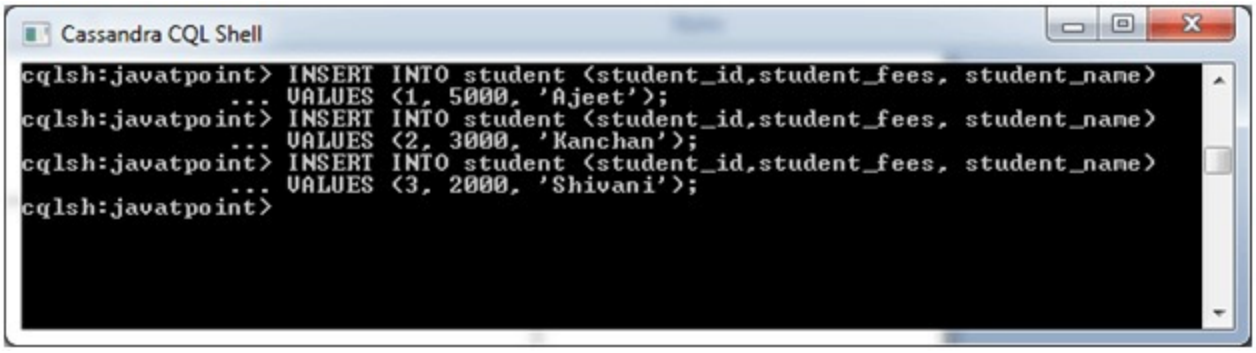
We have a table named "student" with columns (student_id, student_fees student_name,) and need to insert some data in student table.



student_id	student_fees	student_name
1	5000	ajeet
2	3000	kanchan
3	2000	shivani

Let's see the code to insert data in student table.

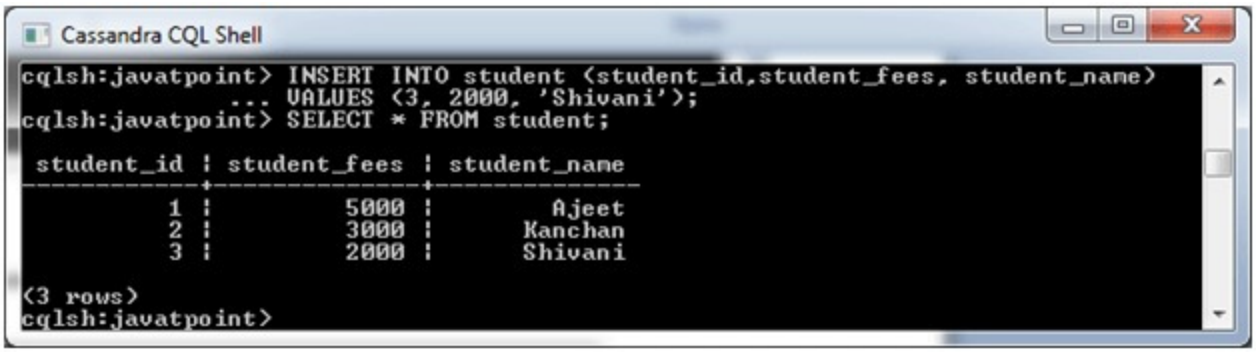
```
INSERT INTO student (student_id, student_fees, student_name) VALUES(1,5000, 'Ajeet');
 INSERT INTO student (student_id, student_fees, student_name) VALUES(2,3000, 'Kanchan');
  INSERT INTO student (student_id, student_fees, student_name) VALUES(3, 2000, 'Shivani');
```



Now the data is inserted. You can use SELECT command to verify whether data is inserted or not.

SELECT * FROM student;

Output:



Cassandra READ Data

SELECT command is used to read data from Cassandra table. You can use this command to read a whole table, a single column, a particular cell etc.

Syntax:

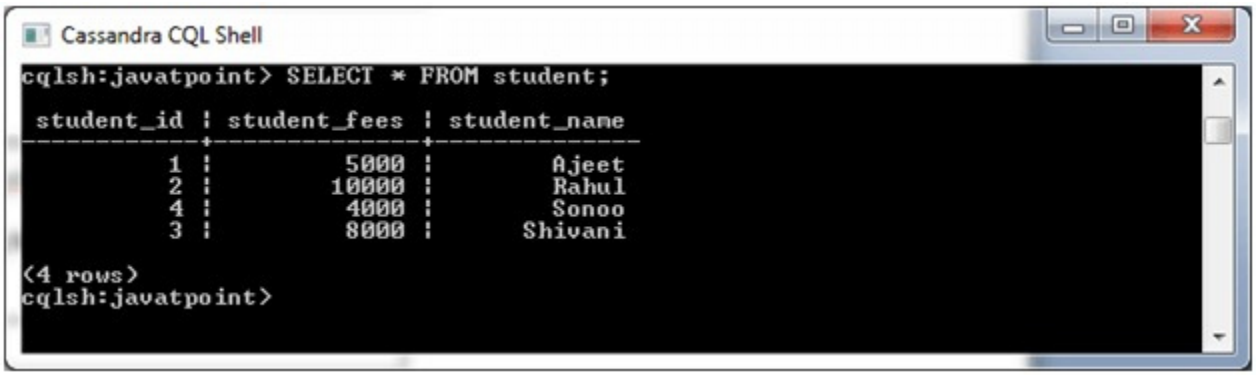
SELECT FROM <tablename>

Example:

Let's take an example to demonstrate how to read data from Cassandra table. We have a table named "student" with columns (student_id, student_fees student_name)

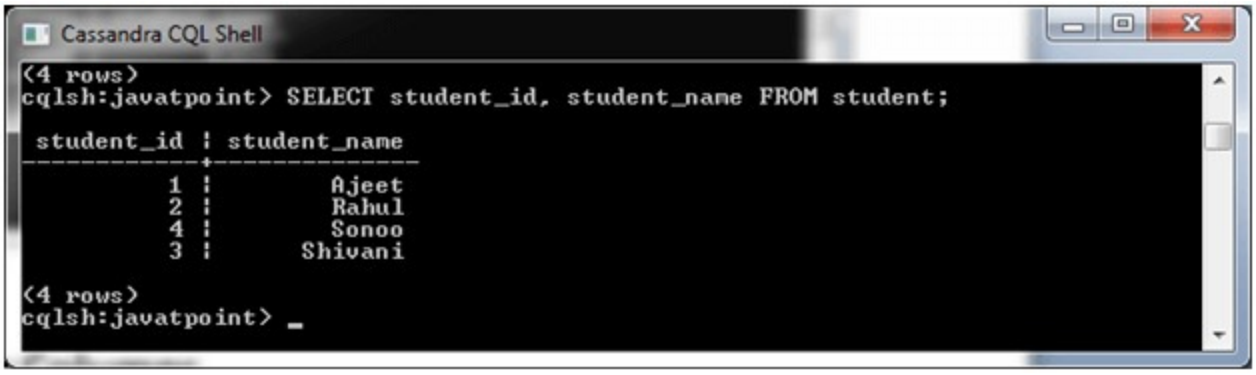
Read the whole table using SELECT command

SELECT * FROM student;



Read Particular Columns
This example will read only student_name and student_id from the student table.

SELECT student_id, student_name FROM student;



Use of WHERE Clause

WHERE clause is used with SELECT command to specify the exact location from where we have to fetch data.

Syntax:

SELECT FROM <table name> WHERE <condition>;

Example:

SELECT * FROM student WHERE student_id=2;