MSc. Computer Science Semester-IV

Name: Aniket Yadav

Roll no. :5734

INDEX

Practical No:01 and 04

AIM: Setting up and Exploring MongoDB

a) Install MongoDB on your local machine or lab server.

b) Create a new MongoDB database and collection.

c) Insert sample data into the collection.

d) Retrieve and display data from the collection using MongoDB queries

e) Write and execute MongoDB queries to retrieve specific data from a collection.

THEORY:

What is MongoDB?

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document. Any relational database has a typical schema design that shows number of tables and the relationship between these tables. While in MongoDB, there is no concept of relationship.

Advantages of MongoDB over RDBMS

- ✓ Schema less − MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
- ✓ Structure of a single object is clear.
- ✓ No complex joins.
- ✓ Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
- ✓ Tuning.
- ✓ Ease of scale-out − MongoDB is easy to scale.
- ✓ Conversion/mapping of application objects to database objects not needed.
- ✓ Uses internal memory for storing the (windowed) working set, enabling faster access of data.

Why Use MongoDB?

- ✓ Document Oriented Storage − Data is stored in the form of JSON style documents.
- ✓ Index on any attribute
- ✓ Replication and high availability
- ✓ Auto-Sharding
- ✓ Rich queries
- ✓ Fast in-place updates
- ✓ Professional support by MongoDB

Where to Use MongoDB?

- ✓ Big Data
- ✓ Content Management and Delivery
- ✓ Mobile and Social Infrastructure
- ✓ User Data Management
- ✓ Data Hub

Create Database in
MongoDB. test> use salma
switched to db salma

To display the name of current database:
salma> db
salma

To list down all the databases, use the command show dbs. This command lists down all the databases and their size on the disk.
salma> show dbs
admin 40.00 KiB
config 60.00 KiB
local 40.00 KiB
since our database has no collections inside it. Our database is not been listed.

Create a collection in the database. salma>
db.createCollection("Student")
{ ok: 1 }

List all the databases again. Now, you will see that our database is also been shown here.
salma> show dbs
admin 40.00 KiB
config 92.00 KiB
local 40.00 KiB
salma 8.00 KiB

Insert data to the collection Student
salma> db.student.insertOne(
... {
... "rollno":1,
... "name":"salma",
... "email":"salma@gmail.com",
... }
{
acknowledged: true,
insertedId: ObjectId("636e6e071ec07139204cfcc2")
}

Find the data in Student
salma> db.student.find() [
{
_id: ObjectId("636e6e071ec07139204cfcc2"), rollno:

```
    1,
    name: 'salma',
    email: 'salma@gmail.com'
  }
]
```

Another way of printing the data of collection.

```
salma> db.student.find().pretty()
[
  {
    _id: ObjectId("636e6e071ec07139204cfcc2"), rollno:
    1,
    name: 'salma',
    email: 'salma@gmail.com'
  }
]
```

Insert data in the collection with insetMany() method.

```
salma>db.student.insertMany([{"rollno":5,"name":"sneha"},
    {"rollno":6,"name":"suman}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("636e6fa51ec07139204cfcc6"),
    '1': ObjectId("636e6fa51ec07139204cfcc7"

    }
}
```

Display the inserted data in the collection.

```
salma> db.student.find().pretty()
[
  {
    _id: ObjectId("636e6e071ec07139204cfcc2"), rollno:
    1,
    name: 'salma',
    email: 'salma@gmail.com'
  },
  {
    _id: ObjectId("636e6fa51ec07139204cfcc6"),
    rollno: 5,
    name: 'sneha'
  },
  {
    _id: ObjectId("636e6fa51ec07139204cfcc7"),
```

```
  rollno: 6,
  name: 'suman'
 }
]
```

Update the collection students.

```
salma> db.student.updateOne({rollno:6},{$set:{"name":"suhana"}})
{
 acknowledged: true,
 insertedId: null,
 matchedCount: 1,
 modifiedCount: 1,
 upsertedCount: 0
}
```

Print the data in the collection to check if the data has been updated or not.

```
salma> db.student.find().pretty()
[
 {
  _id: ObjectId("636e6e071ec07139204cfcc2"), rollno:
  1,
  name: 'salma',
  email: 'salma@gmail.com'},
 {
  _id: ObjectId("636e6fa51ec07139204cfcc6"),
  rollno: 5,
  name: 'sneha'
 },
 {
  _id: ObjectId("636e6fa51ec07139204cfcc7"),
  rollno: 6,
  name: 'suhana'
 }
]
```

You can also add the extra data in the existing document with the same updateOne() method.

```
salma> db.student.updateOne({rollno:6},{$set:{"mobile":"8958926489"}})
{
 acknowledged: true,
 insertedId: null,
 matchedCount: 1,
 modifiedCount: 1,
 upsertedCount: 0
}
```

Check if the data has been added or not.

```
salma> db.student.find().pretty()
[
  {
    _id: ObjectId("636e6e071ec07139204cfcc2"), rollno:
    1,
    name: 'salma',
    email: 'salma@gmail.com'
  },
  {
    _id: ObjectId("636e6fa51ec07139204cfcc6"),
    rollno: 5,
    name: 'sneha'
  },
  {
    _id: ObjectId("636e6fa51ec07139204cfcc7"),
    rollno: 6,
    name: 'suhana',
    mobile: '8958926489'
  }]
```

To drop the collection in the database.

```
salma> db.student.drop()
true
```

Check if collection has been dropped successfully.

```
salma> show collections
Student
```

Drop the database.

```
salma> db.dropDatabase()
{ ok: 1, dropped: 'salma' }
```

Check if database has been dropped successfully.

```
salma> show dbs
admin    40.00 KiB
config  108.00 KiB
local    40.00 KiB
```

PRACTICAL-02

Aim: ouchDB Operations

a) Install and configure Apache Cassandra in a lab environment.

b) Create a keyspace and define a table schema.

c) Insert data into the table.

d) Perform CRUD operations and query data

Description:

What is CouchDB?

CouchDB is an open source database developed by Apache software foundation. The focus is on the ease of use, embracing the web. It is a NoSQL document store database.
It uses JSON, to store data (documents), java script as its query language to transform the documents, http protocol for api to access the documents, query the indices with the web browser. It is a multi master application released in 2005 and it became an apache project in 2008.

Why CouchDB?

CouchDB have an HTTP-based REST API, which helps to communicate with the database easily. And the simple structure of HTTP resources and methods (GET, PUT, DELETE) are easy to understand and use.

As we store data in the flexible document-based structure, there is no need to worry about the structure of the data.

Users are provided with powerful data mapping, which allows querying, combining, and filtering the information.

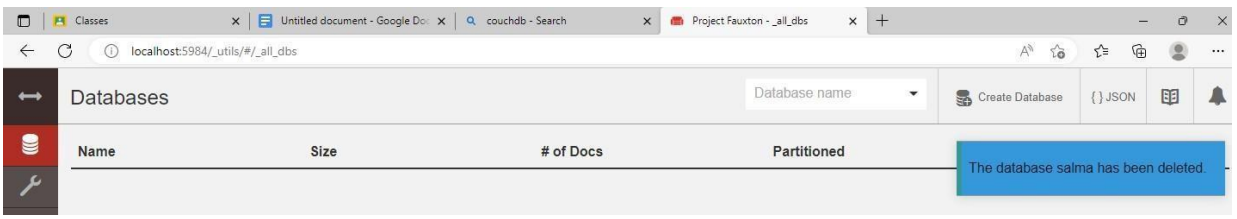
CouchDB provides easy-to-use replication, using which you can copy, share, and synchronize the data between databases and machines.

Data Model

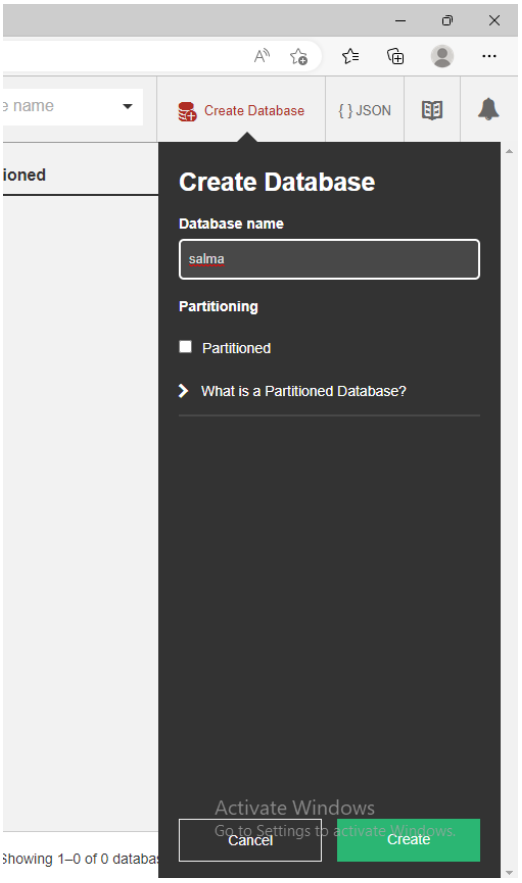
- ✓ Database is the outermost data structure/container in CouchDB.
- ✓ Each database is a collection of independent documents.
- ✓ Each document maintains its own data and self-contained schema.
- ✓ Document metadata contains revision information, which makes it possible to merge the differences occurred while the databases were disconnected.
- ✓ CouchDB implements multi version concurrency control, to avoid the need to lock the database field during writes.

PERFORMING CRUD IN COUCHDB

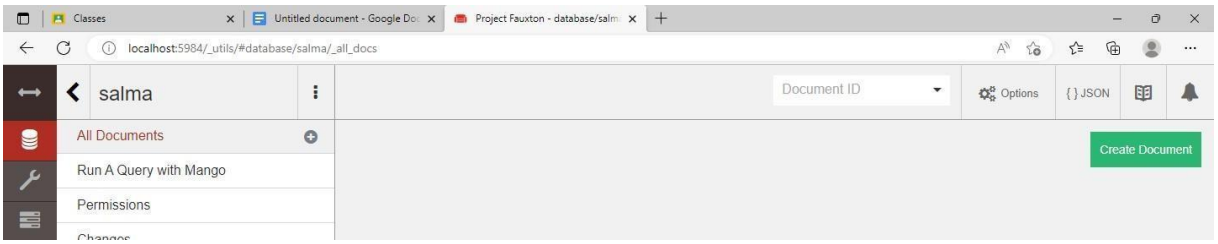
- Install the CouchDB
- Go to the browser and type: **http://localhost:5984**
- Now type: **http://localhost:5984/_utils**
- This will open the UI of CouchD as shown below.



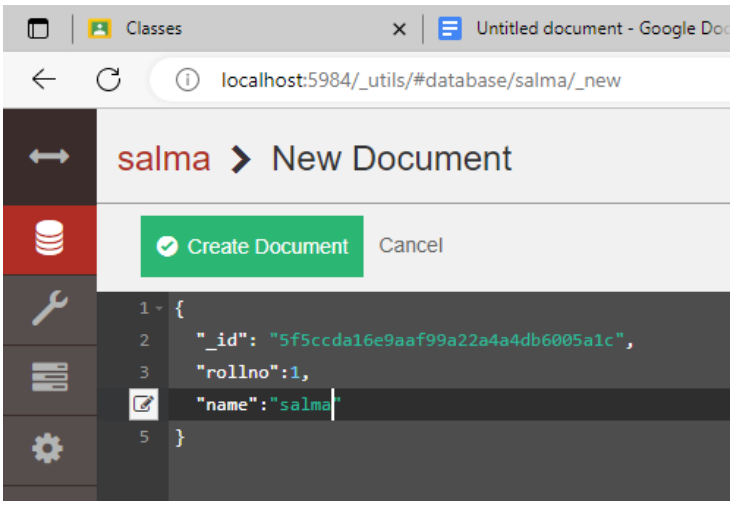
At the top right corner there is an option of create database.
Click on the button and it will show the database creation option as given below. Type the name of database you want and then click create button.



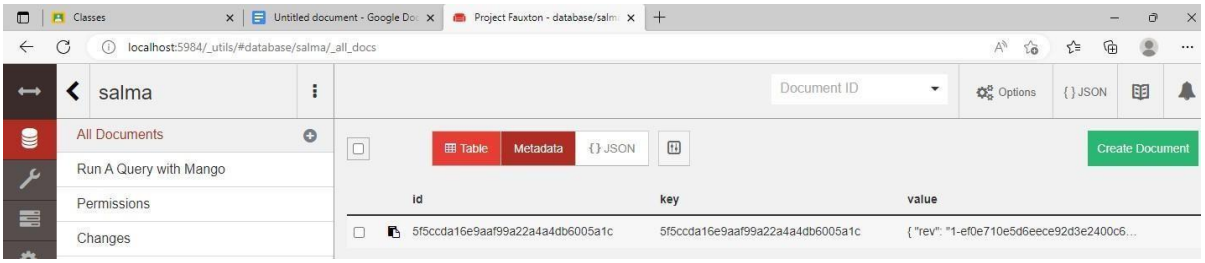
After the creation of the database the screen Will look like this. At the top right corner there is an option of create document.



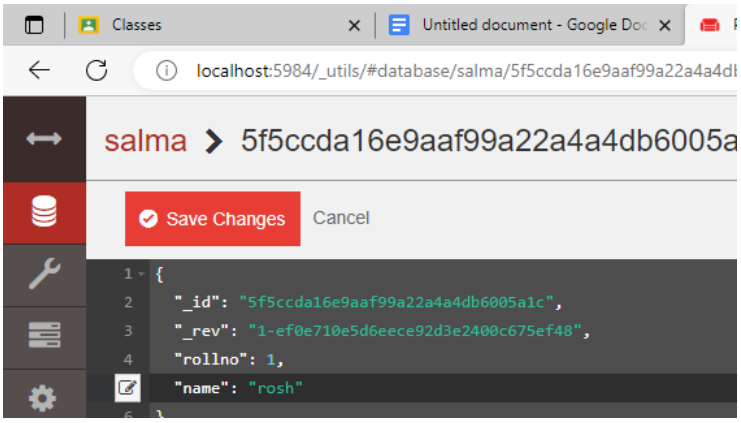
Click on the create document button. And, the screen will appear as shown below. Enter the data of your choice and click on create document button again.



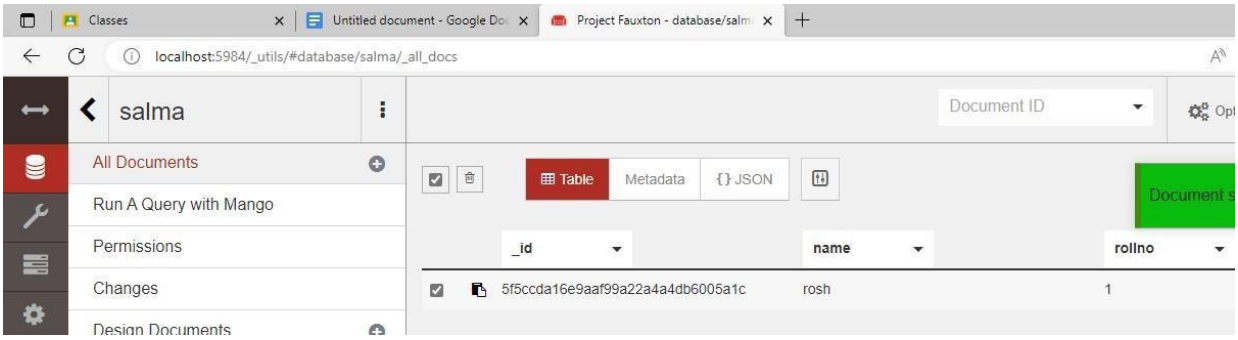
Once the data is been created, you will see your inserted data as shown below.



You can update the data by clicking on the data and click on the save changes button on the top left corner and your data will be updated.

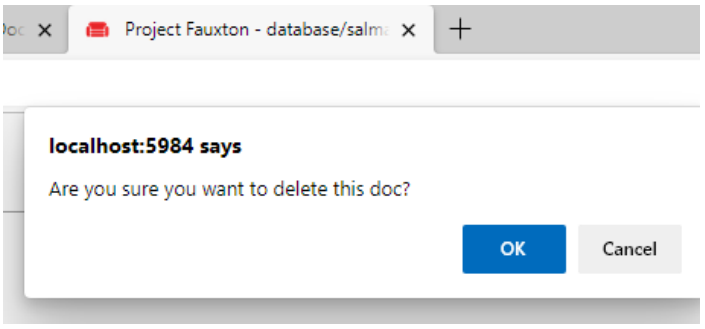


After updating you can view your data properly by click on the table view option given on top middle part.

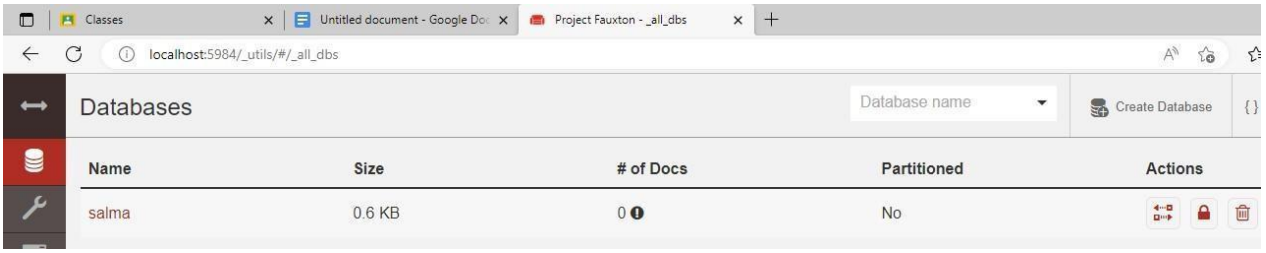


After checking the check box of the row you can delete the data by clicking on the delete icon.

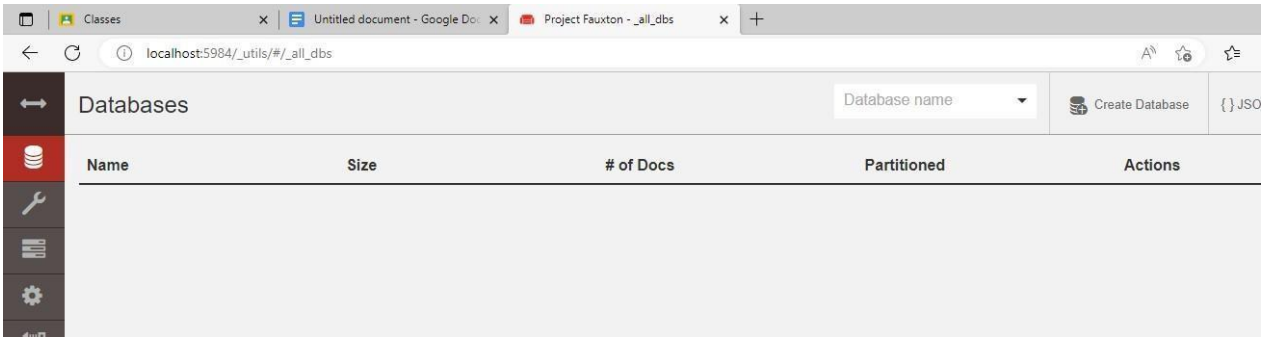
When you click on delete icon. It will show you a prompt asking for confirmation. Once you confirm it your data will be deleted.



Now this below picture shows the name of your database. You can delete your database by clicking on the delete icon.



After clicking on the delete icon, you will see the prompt asking for the name of your database, once you type your database name and click on delete button, your database will be deleted.



Above picture shows that there is no database in your databases list

PRACTICAL-03 and 5

Aim: Interacting with Redis

a) Install Redis on your lab server or local machine.

b) Store and retrieve data in Redis using various data structures like strings, lists, and sets.

c) Implement basic Redis commands for data manipulation and retrieval.

d) Use Redis commands to manipulate and modify data stored in different data structures.

e) Retrieve specific data using Redis query operations.

What is Redis database?

Redis is an open source (BSD licensed), in-memory data structure store used as a database, cache, message broker, and streaming engine.
Redis provides data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes, and streams.
Redis has built-in replication, Lua scripting, LRU eviction, transactions, and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

You can run atomic operations on these types, like appending to a string; incrementing the value in a hash; pushing an element to a list; computing set intersection, union and difference; or getting the member with highest ranking in a sorted set.

To achieve top performance, Redis works with an in-memory dataset.
Depending on your use case, Redis can persist your data either by periodically dumping the dataset to disk or by appending each command to a disk-based log.

PERFORMING CRUD IN REDIS

Two ways to open the redis cli

- Go to the C drive> program files>redis>redis-cli
- Open cmd >type redis-cli>press enter

To store string data in redis

```
127.0.0.1:6379> set name salma
OK
127.0.0.1:6379> set rollno 2
OK
127.0.0.1:6379> set add chembur_mumbai_89
OK
127.0.0.1:6379> set mail salmashaikh98677@gmail.com OK
```

To retrieve the stored string data in redis

```
127.0.0.1:6379> get name
"salma"
127.0.0.1:6379> get name
"salma"
127.0.0.1:6379> get add
```

"chembur_mumbai_89"
127.0.0.1:6379> get mail
"salmashaikh98677@gmail.com"

To list all the keys
127.0.0.1:6379> keys *
1) "name"
2) "mail"
3) "add"
4) "rollno"

To check the whether the key exists or not (1 means exists, 0 mean not exists)
127.0.0.1:6379> exists name
(integer) 1
127.0.0.1:6379> exists mobile
(integer) 0

To delete the key from the database
127.0.0.1:6379> del rollno
(integer) 1

After deleting the key if try to get it the output will be nil and existence value will be 0
127.0.0.1:6379> get rollno
(nil)
127.0.0.1:6379> exists rollno
(integer) 0

To check the time to live of key
127.0.0.1:6379> ttl name
(integer) -1

To set the expiration time of key
127.0.0.1:6379> expire add 15
(integer) 1

After setting the expiration time you can check the time left for the key to get expire
127.0.0.1:6379> ttl add
(integer) 13
127.0.0.1:6379> ttl add
(integer) 7
127.0.0.1:6379> ttl add
(integer) 2
127.0.0.1:6379> ttl add
(integer) 1
127.0.0.1:6379> ttl add
(integer) 0
127.0.0.1:6379> ttl add
(integer) -2

Set the expiration time of the key

127.0.0.1:6379> expire add 200
(integer) 1
127.0.0.1:6379> ttl add
(integer) 177

Now. After setting the expiration time of the key , you can again edit the time of expiration of the key
127.0.0.1:6379> setex add 30 vadala
OK
127.0.0.1:6379> ttl add
(integer) 28

To create list data in redis
127.0.0.1:6379> lpush friends teju rosh salma kavita priyanka (integer) 5

To retrieve list data on redis
127.0.0.1:6379> lrange friends 0 -1
1) "priyanka"
2) "kavita"
3) "salma"
4) "rosh"
5) "teju"

Lpush insert data at the start
127.0.0.1:6379> lpush friends sunita
(integer) 6

Output of lpush
127.0.0.1:6379> lrange friends 0 -1
1) "sunita"
2) "priyanka"
3) "kavita"
4) "salma"
5) "rosh"
6) "teju"

Rpush insert data at the end
127.0.0.1:6379> rpush friends cscornersunita
(integer) 7

Output of rpush
127.0.0.1:6379> lrange friends 0 -1
1) "sunita"
2) "priyanka"
3) "kavita"
4) "salma"
5) "rosh"
6) "teju"
7) "cscornersunita"

To create set data in redis

```
127.0.0.1:6379> sadd fruits mango apple banana papaya
(integer) 4
```

To retrieve set data from redis

```
127.0.0.1:6379> smembers fruits
1) "banana"
2) "mango"
3) "apple"
4) "papaya"
```

To create ordered set in redis

```
127.0.0.1:6379> zadd os 1 windows 2 linux 3 unix
(integer) 3
```

To retrieve ordered set in redis

```
127.0.0.1:6379> zrange os 0 -1
1) "windows"
2) "linux"
3) "unix"
```

PRACTICAL-06

Aim: Implementing Indexing in MongoDB
a) Create an index on a specific field in a MongoDB collection.
b) Measure the impact of indexing on query performance

Description:

The createIndex() method has the following form:

```
db.collection.createIndex( <keys>, <options>, <commitQuorum>)
```

The createIndex() method takes the following parameters:

Parameter	Type	Description
keys	document	A document that contains the field and value pairs where the field is the index key and the value describes the type of index for that field. For an ascending index on a field, specify a value of 1. For descending index, specify a value of -1. An asterisk (*) is not a valid index name. MongoDB supports several different index types, including: text geospatial hashed indexes See index types for more information. Wildcard indexes support workloads where users query against custom fields or a larg e variety of fields in a collection. You can create a wildcard index on a specific field and its subpaths or on all of the fields in a document.
options	document	Optional. A document that contains a set of options that controls the creation of the index. See Options for details.
commitQuorum	integer or string	Optional. The minimum number of data-bearing voting replica set members (i.e. commit quorum), including the primary, that must Zreport a successful index build before the primary marks the indexes as ready. A "voting" member is any replica set member where members[n].votes is greater than 0. Supports the following values: "votingMembers" - all data-bearing voting replica set members (*Default*).

Parameter	Type	Description
		"majority" - a simple majority of data-bearing voting replica set members. <int> - a specific number of data-bearing voting replica set members. 0 - Disables quorum-voting behavior. Members start the index build simultaneously but do *not* vote or wait for quorum before completing the index build. If you start an index build with a commit quorum of 0, you cannot later modify the commit quorum using setIndexCommitQuorum.

Options

The options document contains a set of options that controls the creation of the index. Different index types can have additional options specific for that type.

Multiple index options can be specified in the same document. However, if you specify multiple option documents the db.collection.createIndex() operation will fail.

Consider the following db.collection.createIndex() operation:

```
db.collection.createIndex(
{
"a": 1
},
{
unique: true,
sparse: true,
expireAfterSeconds: 3600
}
)
```

If the options specification had been split into multiple documents like this: { unique: true }, { sparse: true, expireAfterSeconds: 3600 } the index creation operation would have failed.

Options for All Index Types

The following options are available for all index types unless otherwise specified:

Parameter	Type	Description
unique	boolean	Optional. Creates a unique index so that the collection will not accept insertion or update of documents where the index key value matches an existing value in the index.

Parameter	Type	Description
		Specify `true` to create a unique index. The default value is `false`. The option is *unavailable* for hashed indexes.
`name`	string	Optional. The name of the index. If unspecified, MongoDB generates an index name by concatenating the names of the indexed fields and the sort order.
`partialFilterExpression`	document	Optional. If specified, the index only references documents that match the filter expression. See Partial Indexes for more information. A filter expression can include: equality expressions (i.e. `field: value` or using the `$eq` operator), `$exists: true` expression, `$gt`, `$gte`, `$lt`, `$lte` expressions, `$type` expressions, `$and` operator, `$or` operator, `$in` operator You can specify a `partialFilterExpression` option for all MongoDB index types.
`sparse`	boolean	Optional. If `true`, the index only references documents with the specified field. These indexes use less space but behave differently in some situations (particularly sorts). The default value is `false`. See Sparse Indexes for more information. For a compound index that includes `2dsphere` index key(s) along with keys of other types, only the `2dsphere` index fields determine whether the index references a document.
`expireAfterSeconds`	integer	Optional. Specifies a value, in seconds, as a time to live (TTL) to control how long MongoDB retains documents in this collection. This option only applies to TTL indexes. See Expire Data from Collections by Setting TTL for more information. If you use TTL indexes created before MongoDB 5.0, or if you want to sync data created in MongDB 5.0 with a pre-5.0 installation, see Indexes Configured Using NaN to avoid misconfiguration issues. The TTL index `expireAfterSeconds` value must be within `0` and `2147483647` inclusive.

Parameter	Type	Description
hidden	boolean	Optional. A flag that determines whether the index is hidden from the query planner. A hidden index is not evaluated as part of the query plan selection. Default is false.
storageEngine	document	Optional. Allows users to configure the storage engine on a per-index basis when creating an index. The storageEngine option should take the following form: storageEngine: { <storage-engine-name>: <options> } Storage engine configuration options specified when creating indexes are validated and logged to the oplog during replication to support replica sets with members that use different storage engines.

Option for Collation

Parameter	Type	Description

collation	document	Optional. Specifies the collation for the index.
		
		Collation allows users to specify language-specific rules for string comparison, such as rules for lettercase and accent marks.
		
		If you have specified a collation at the collection level, then:
		
		If you do not specify a collation when creating the index, MongoDB creates the index with the collection's default collation.
		
		If you do specify a collation when creating the index, MongoDB creates the index with the specified collation.
		
		The collation option has the following syntax:
		
		collation: {
		locale: <string>,
		caseLevel: <boolean>,
		caseFirst: <string>,
		strength: <int>,
		numericOrdering: <boolean>,
		alternate: <string>,
		maxVariable: <string>,
		backwards: <boolean>
		}
		When specifying collation, the locale field is mandatory; all other collation fields are optional. For descriptions of the fields, see Collation Document.

The collection myColl has an index on a string field category with the collation locale "fr".

```
db.myColl.createIndex( { category: 1 }, { collation: { locale: "fr" } } )
```

The following query operation, which specifies the same collation as the index, can use the index:

```
db.myColl.find( { category: "cafe" } ).collation( { locale: "fr" }
```

The following query operation, which by default uses the "simple" binary collator, cannot use the index:

```
db.myColl.find( { category: "cafe" } )
```

For a compound index where the index prefix keys are not strings, arrays, and embedded documents, an operation that specifies a different collation can still use the index to support comparisons on the index prefix keys.

For example, the collection myColl has a compound index on the numeric fields score and price and the string field category; the index is created with the collation locale "fr" for string comparisons:

```
db.myColl.createIndex(
{ score: 1, price: 1, category: 1 },
{ collation: { locale: "fr" } } )
```

The following operations, which use "simple" binary collation for string comparisons, can use the index:

```
db.myColl.find( { score: 5 } ).sort( { price: 1 } )
db.myColl.find( { score: 5, price: { $gt: NumberDecimal( "10" ) } } ).sort( { price: 1 } )
```

The following operation, which uses "simple" binary collation for string comparisons on the indexed category field, can use the index to fulfill only the score: 5 portion of the query:

```
db.myColl.find( { score: 5, category: "cafe" } )
```

Practical-07

Aim: Data Storage in Redis
a) Implement caching functionality using Redis as a cache store.
b) Store and retrieve data from Redis cache using appropriate commands.

Description:

Redis Cache:
caching refers to storing frequently used or dealt-up data in temporary high-speed storage to reduce the latency of a system. Now so do the same when happens inside a Redis cluster. Therefore, Redis Cache supercharges application performance by utilizing in-memory data caching. By storing frequently accessed data in memory, Redis Cache dramatically reduces response times and database load, resulting in faster and more scalable applications.

Caching in Redis:
Redis, often referred to as a "data structure server," is known for its exceptional performance and versatility. While Redis offers a wide range of features, one of its primary use cases is data caching. Redis caching leverages its in-memory storage capabilities, allowing applications to store and retrieve data with extremely low latency. It provides key-value storage, allowing developers to store and retrieve data using unique keys. With Redis, developers can set expiration times for cached data, ensuring that the cache remains up to date. Additionally, Redis offers advanced features like pub/sub messaging, which can be leveraged to implement cache invalidation mechanisms. By using Redis for caching, applications can significantly enhance their performance, reduce response times, and improve overall scalability.

Implementation of caching in Redis
In this section, we will explore the step-by-step implementation of Redis caching in an application. We will cover the following subtopics with code snippets and examples:

Establishing a Connection with Redis
To begin, we need to establish a connection with the Redis server. The following code snippet demonstrates how to connect to Redis and check if the connection is successful by sending a ping request and receiving the response as "True."

```pip install redis```

```
import redis
python
# Connect to Redis
r = redis.Redis(host='localhost', port=6379, db=0)
# Send a ping request and check the response
response = r.ping()
print("Response:", response)
```

Defining External Routes from the API:
Next, we define the external routes from our API that will be responsible for fetching data. These routes can be endpoints that retrieve data from a database, external APIs, or any other data source. Here is an example of defining an API route to fetch user data:

```
from flask import Flask, jsonify
app = Flask(__name__)
# API route to fetch user data
```
```

```
@app.route('/api/users/<user_id>', methods=['GET'])
def get_user(user_id):
    # Code to fetch user data from the database or external APIs
    user_data = fetch_user_data(user_id)
    return jsonify(user_data)

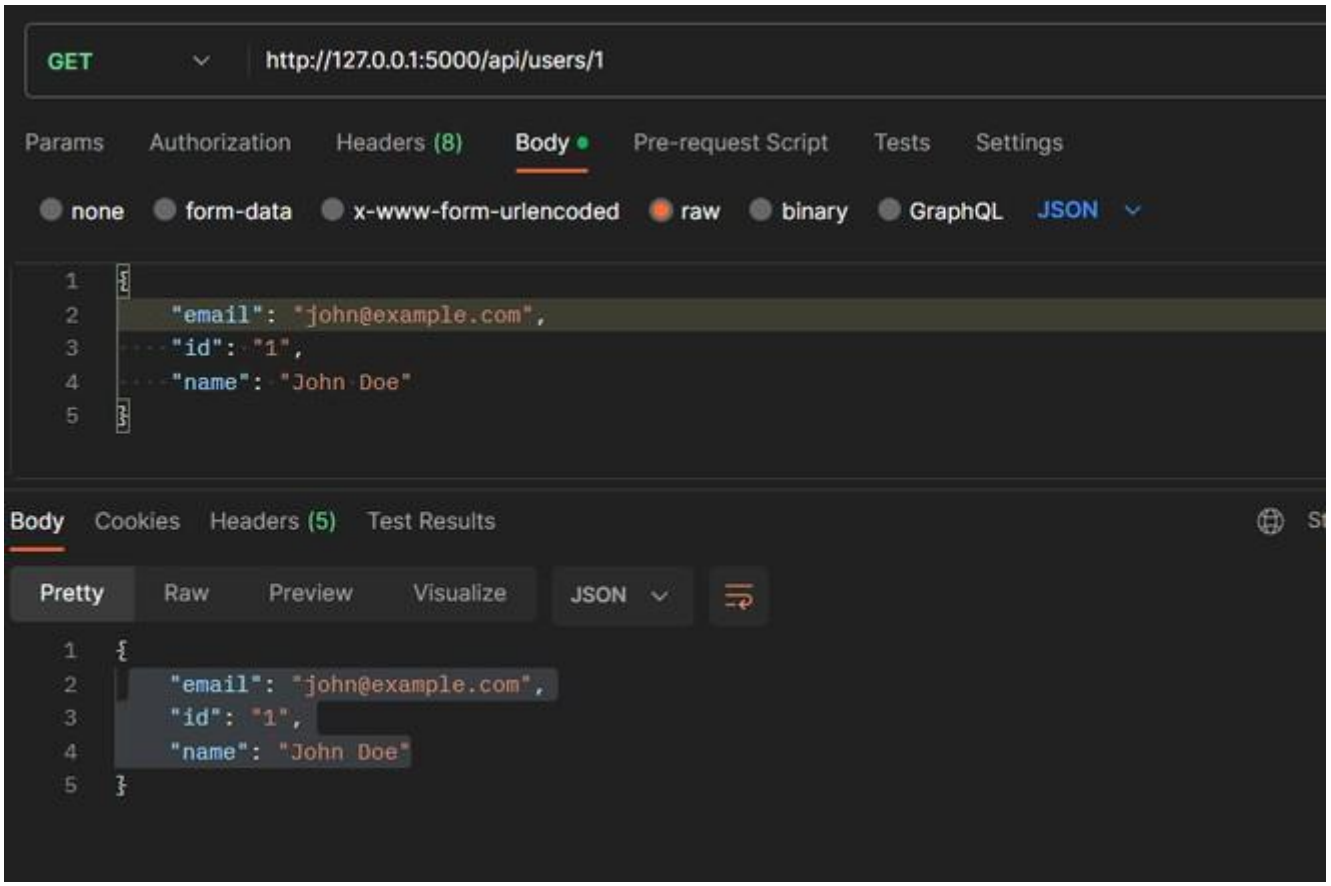
if __name__== '__main__':
    app.run()
```

Establishing Caching with Redis:

To implement caching, we can utilize Redis to store and retrieve data. The code snippet below demonstrates how to establish caching by checking if the requested data is available in the cache. If not, it fetches the data from the external route and stores it in Redis for future requests.

```
# Function to fetch user data either from the cache or the external route
def fetch_user_data(user_id):
    # Check if data is available in the cache
    user_data = r.get(user_id)
    if user_data is None:
        # Fetch data from the external route
        response = requests.get(f'http://localhost:5000/api/users/{user_id}')
        user_data = response.json()
        # Store the fetched the data in the cache for future requests
        r.set(user_id, json.dumps(user_data))

    return user_data
```



Practical-08

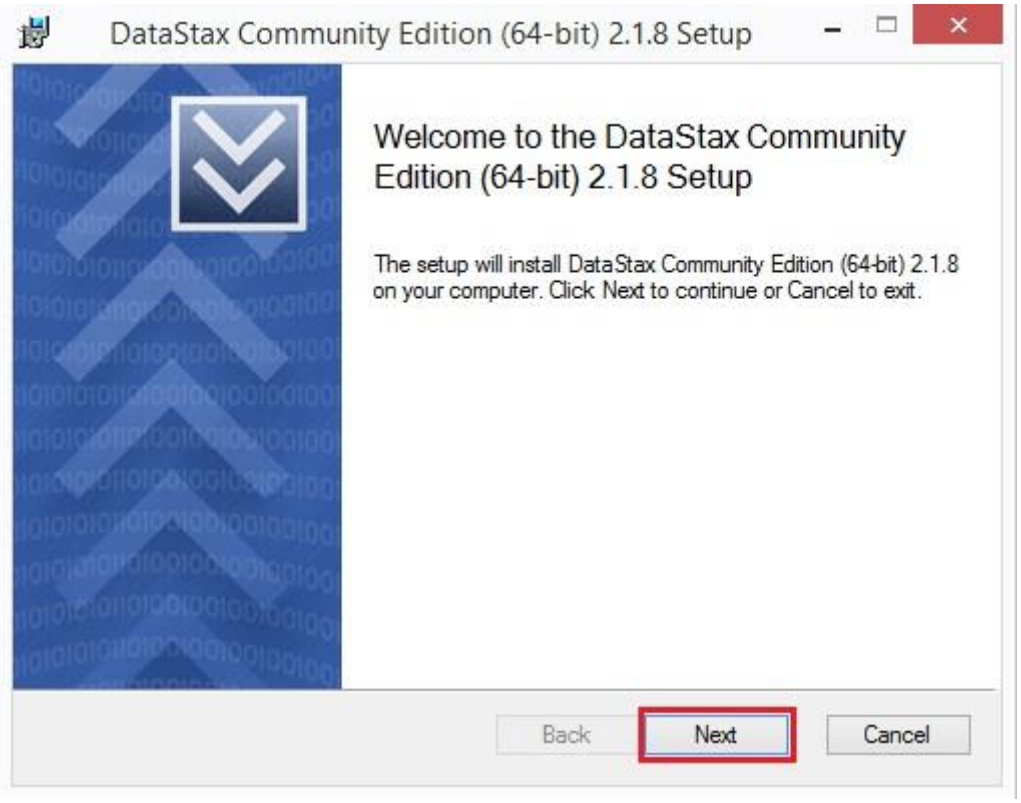
Aim: Apache Cassandra Operations
a) Install and configure Apache Cassandra in a lab environment.
b) Create a keyspace and define a table schema.
c) Insert data into the table.
d) Perform CRUD operations and query data from Apache Cassandra.

Installation

- You must have Datastax community edition. You can download Cassandra from Apache website.
- JDK must be installed
- Windows platform is required

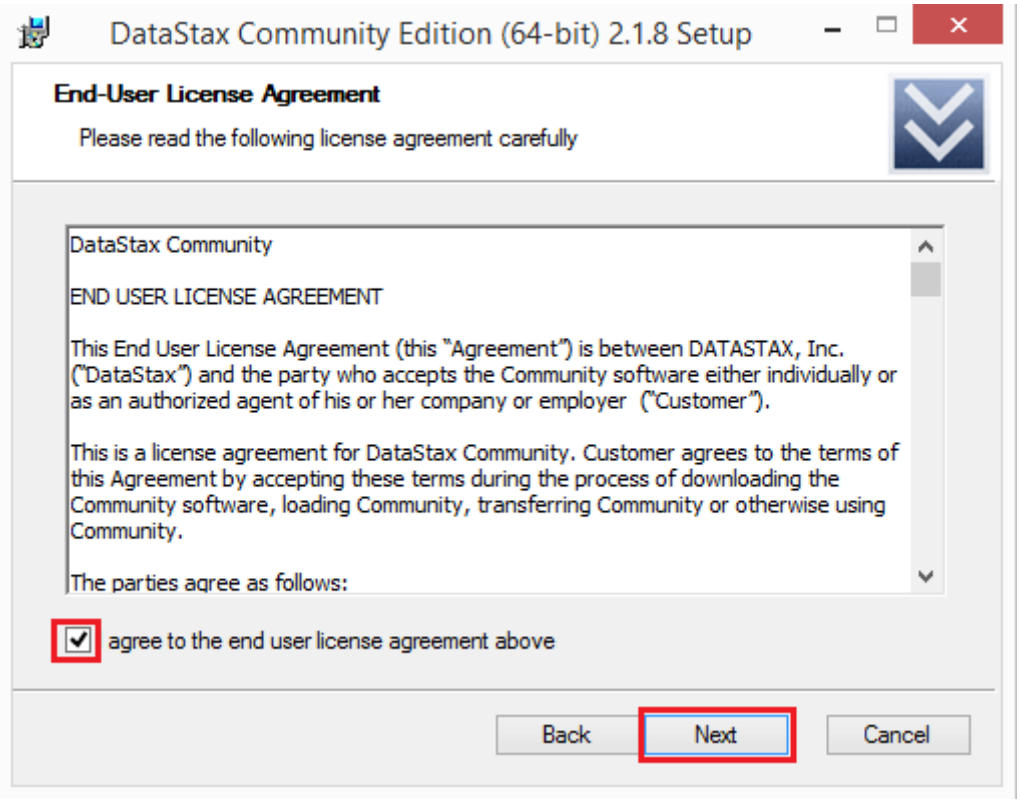
How to Download and Install Cassandra

Step 1) Run the Datastax community edition setup.
After running the Setup, following page will be displayed. Here in the screenshot 64 bit version is being installed. You can download 32 bit version as well according to your requirements. But I recommend 64 bit version to use.



This page gives you information about the Cassandra version you are going to install. Press the 'next' button.

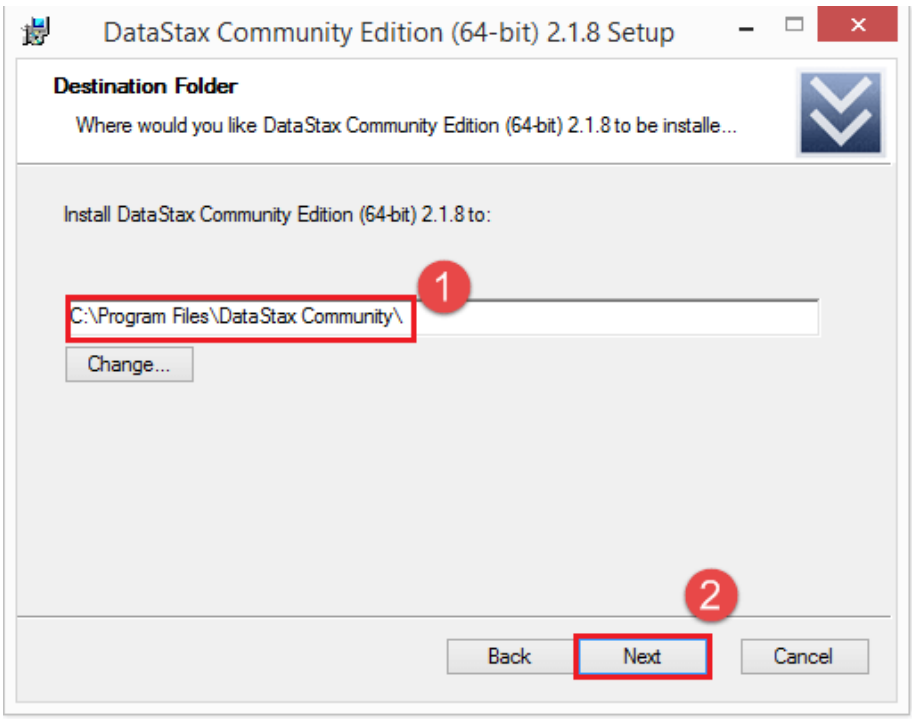
Step 2) press the 'next' button.
After pressing the 'next' button, following page will be displayed.



This page is about the license agreement. Mark the checkbox and press the next button.

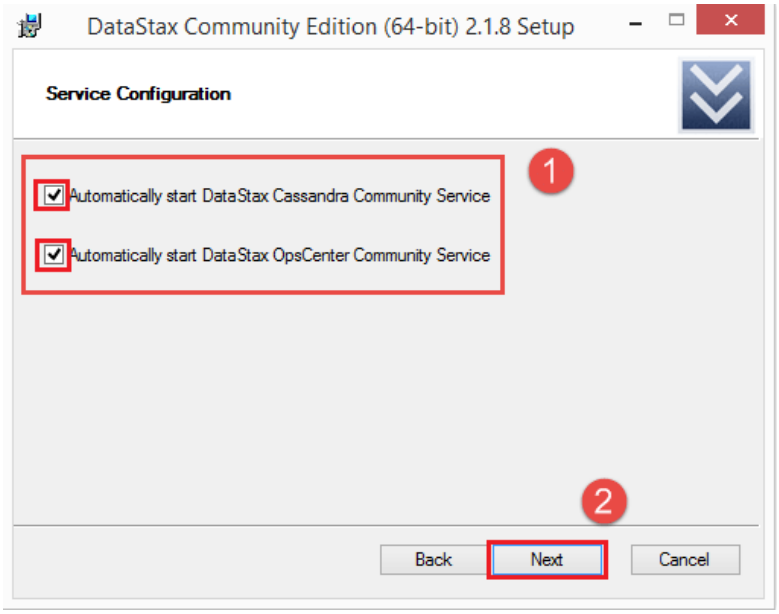
Step 3) Press the 'next' button.
The following page will be displayed asks about the installation location.

1. Default location is C:\Program Files. You can change installation location if you want to change. It is recommended not to change installation location.
2. After setting installation location, press the 'next' button

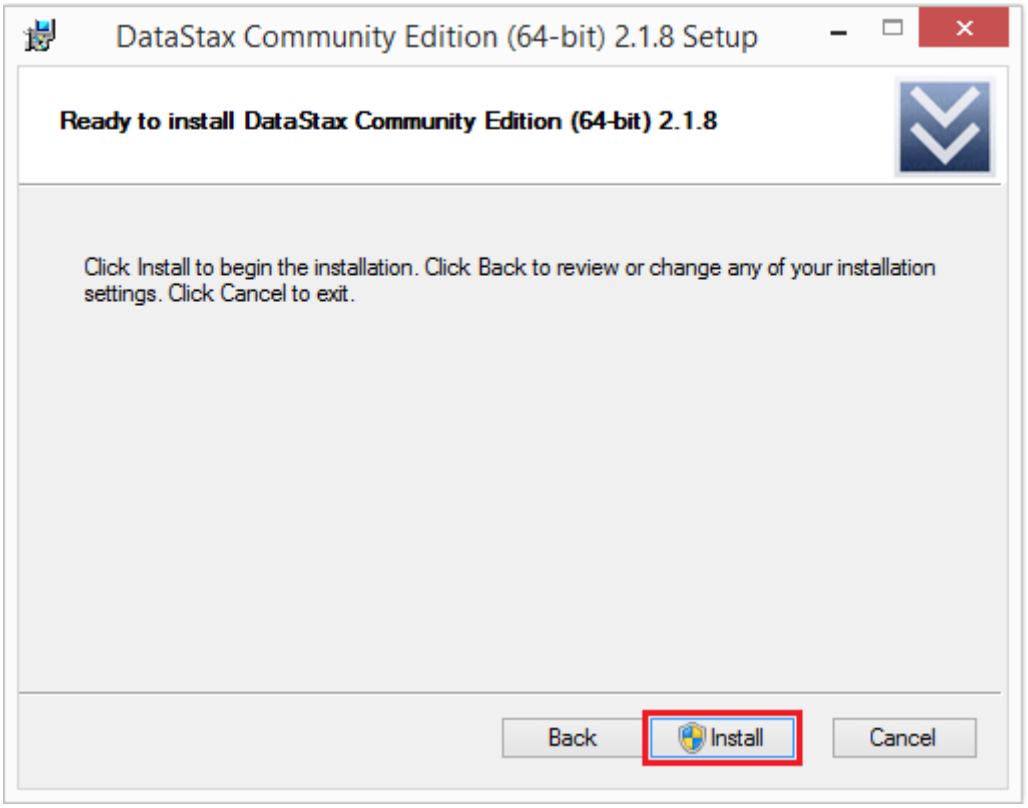


Step 4) start Cassandra and OpsCenter.
After pressing ‘next’ button in above step, the following page will be displayed. This page asks about whether you want to automatically start Cassandra and OpsCenter.

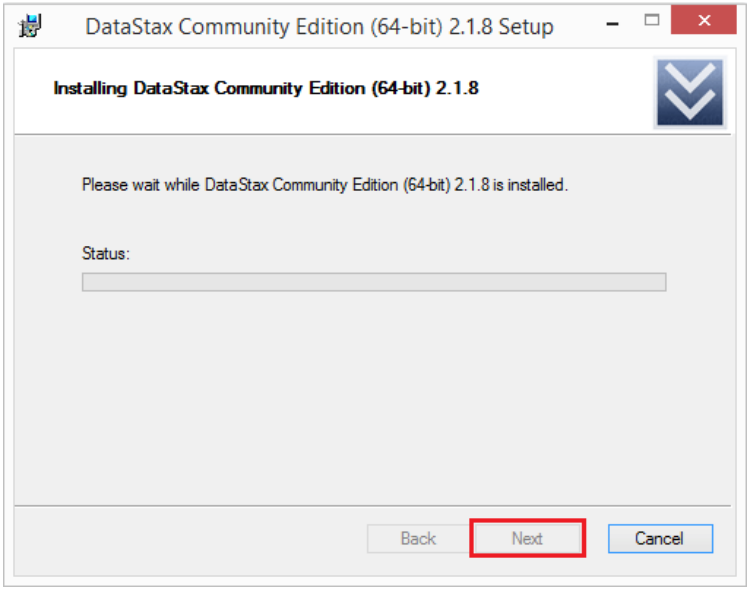
1. Mark the checkboxes if you want to automatically start Cassandra and opsCenter.
2. After providing this information, press the ‘next’ button.



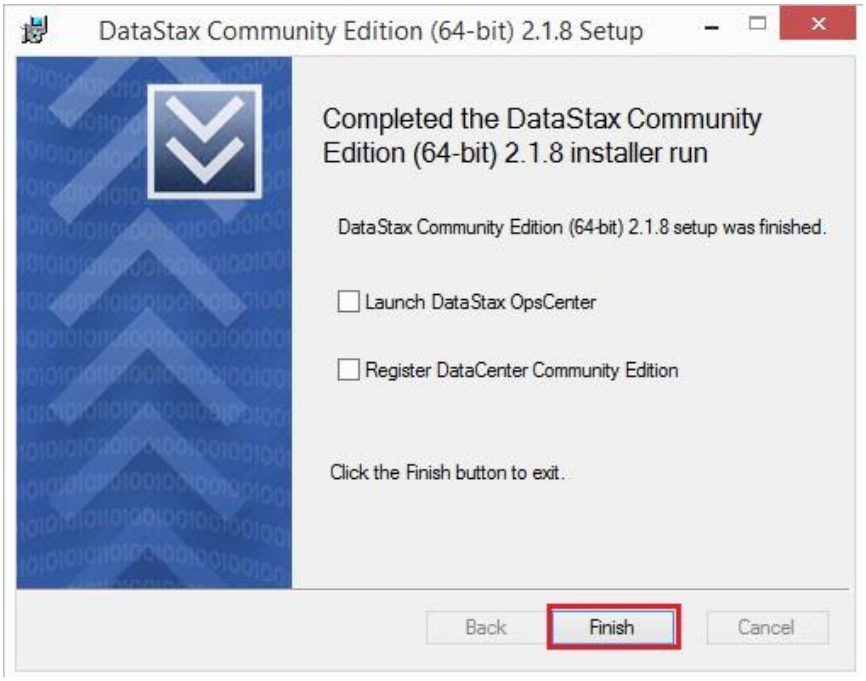
Step 5) Click on install button.
Setup has collected all the necessary information and now the setup is ready to install. Press install button.



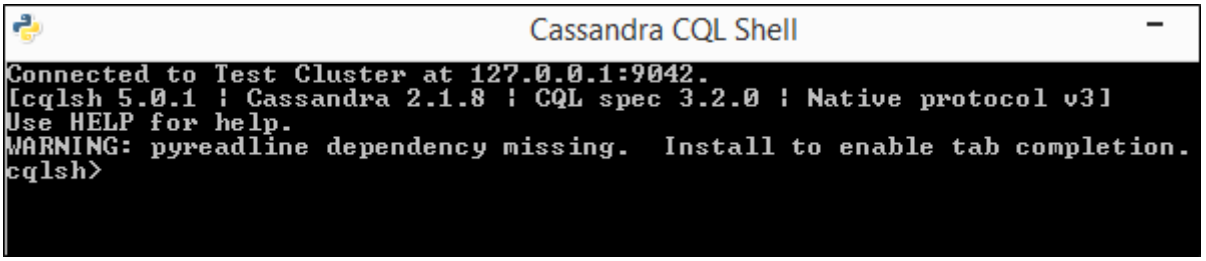
Step 6) Click on Next
After pressing ‘install’ button, following page will be displayed.



Datastax community edition is being installed. After installation is completed, click on next button. When setup is installed successfully, press the ‘Finish’ button.



Go to windows start programs, search Cassandra CQL Shell and run the Cassandra Shell. After running Cassandra shell, you will see the following command line



Now you can create a keyspace, tables, and write queries.

Command "**Create Keyspace**" is used to create keyspace in Cassandra.

Syntax

Create keyspace KeyspaceName with replication={'class':strategy name,
'replication_factor': No of replications on different nodes};

Various Components of Cassandra Keyspace

- **Strategy**: While declaring strategy name in Cassandra. There are two kinds of strategies declared in Cassandra Syntax.

1. **Simple Strategy**: Simple strategy is used when you have just one data center. In this strategy, the first replica is placed on the node selected by the partitioner. Remaining nodes are placed in the clockwise direction in the ring without considering rack or node location.
2. **Network Topology Strategy**: Network topology strategy is used when you have more than one data centers. In this strategy, you have to provide replication factor for each data center separately. Network topology strategy places replicas in nodes in the clockwise direction in the same data center. This strategy attempts to place replicas in different racks.

- **Replication Factor**: Replication factor is the number of replicas of data placed on different nodes. For no failure, 3 is good replication factor. More than two replication factor ensures no single point of failure. Sometimes, the server can be down, or network problem can occur, then other replicas provide service with no failure.

- Create keyspace University with replication={'class':SimpleStrategy,'replication_factor': 3};

Create table Syntax

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

Create table University.Student
(

```
    rollno  int,
    name text,
    dept text,
        primary key(roll_no)
);
```

Insert data Syntax

Insert into KeyspaceName.TableName(ColumnName1, ColumnName2, ColumnName3 .....)
values (Column1Value, Column2Value, Column3Value.....)

insert into University.Student(roll_no,name,dept)values(1,'vinita','cs');

Update Data

The Cassandra Update query is used to update the data in the Cassandra table. If no results are returned after updating data, it means data is successfully updated otherwise an error will be returned. Column values are changed in 'Set' clause while data is filtered with 'Where' clause.

Syntax

Update KeyspaceName.TableName
Set ColumnName1=new Column1Value,
    ColumnName2=new Column2Value,
    ColumnName3=new Column3Value,
    .
    .
    .
Where ColumnName=ColumnValue

update University.Student set name='john' where roll_no=1;

Delete Data

Command 'Delete' removes an entire row or some columns from the table Student. When data is deleted, it is not deleted from the table immediately. Instead deleted data is marked with a tombstone and are removed after compaction.

Syntax

Delete from KeyspaceName.TableName Where ColumnName1=ColumnValue

Delete from University.Student where rollno=1;

Practical 9

Aim: Python api to check square of a number

nine.py

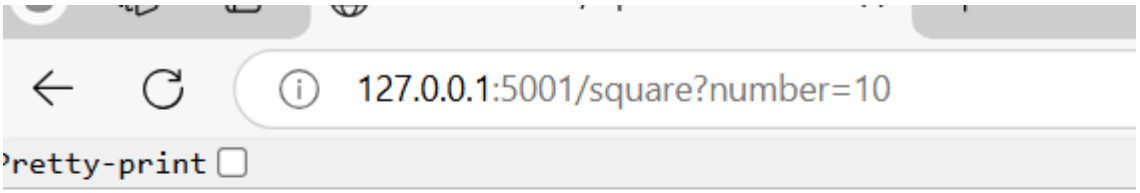
```python
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/square', methods=['GET'])  # Corrected here
def square():
    try:
        num = int(request.args.get('number'))
        return jsonify({
            "practical": 9,
            "number": num,
            "square": num ** 2
        })
    except (TypeError, ValueError):
        return jsonify({"error": "Invalid input, provide an integer"}), 400

if __name__ == '__main__':
    app.run(debug=True, port=5001)
```

Run




```
"number": 10,
"practical": 9,
"square": 100
```

Practical 10

Aim: Python api to check whether a number is Armstrong or not.

Code:

```
from flask import Flask, request, jsonify

app = Flask(__name__)

def is_armstrong(num):
    order = len(str(num))
    return num == sum(int(digit) ** order for digit in str(num))

@app.route('/armstrong', methods=['GET'])
def armstrong():
    try:
        num = int(request.args.get('number'))
        return jsonify({
            "practical": 10,
            "number": num,
            "is_armstrong": is_armstrong(num)
        })
    except (TypeError, ValueError):
        return jsonify({"error": "Invalid input, provide an integer"}), 400

if __name__ == '__main__':
    app.run(debug=True, port=5002)

# Request url: http://127.0.0.1:5002/armstrong?number=153
```


```
"is_armstrong": true,
"number": 153,
"practical": 10
```