Python assignment-4

1.Explain the difference between Cassandra and typical databases?

Ans. **Cassandra:** Cassandra is a high-performance and highly scalable distributed NoSQL database management system. Cassandra deals with unstructured data and handles a high volume of incoming data velocity. In Cassandra data is written in many locations also data come from many locations this row represents a unit of replication and the column represents a unit of storage.

TYPICAL DATA BASE: Relational Database Management System (TYPICAL DATA BASE) is a Database management system or software that is designed for relational databases and uses Structured Query Language (SQL) for querying and maintaining the database. It deals with structured data and handles moderate incoming data velocity. In TYPICAL DATA BASE mainly data is written in one location also data come from one/few locations and a row represents a single record column that represents an attribute.

Difference between Cassandra and TYPICAL DATA BASE:

S.No.	CASSANDRA	TYPICAL DATA BASE
1.	Cassandra is a high performance and highly scalable distributed NoSQL database management system.	TYPICAL DATA BASE is a Database management system or software which is designed for relational databases.
2.	Cassandra is a NoSQL database.	TYPICAL DATA BASE uses SQL for querying and maintaining the database.
3.	It deals with unstructured data.	It deals with structured data.
4.	It has a flexible schema.	It has fixed schema.
5.	Cassandra has peer-to-peer architecture with no single point of failure.	TYPICAL DATA BASE has master-slave core architecture means a single point of failure.
6.	Cassandra handles high volume incoming data velocity.	TYPICAL DATA BASE handles moderate incoming data velocity.

S.No.	CASSANDRA	TYPICAL DATA BASE
7.	In TYPICAL DATA BASE there is limited data source means data come from many locations.	In Cassandra there are various data source means data come from one/few location.
8.	It supports simple transactions.	It supports complex and nested transactions.
9.	In Cassandra the outermost container is Keyspace.	In TYPICAL DATA BASE the outermost container is database.
10.	Cassandra follows decentralized deployments.	TYPICAL DATA BASE follows centralized deployments.
11.	In Cassandra data written in many locations.	In TYPICAL DATA BASE mainly data are written in one location.
12.	In Cassandra row represents a unit of replication.	In TYPICAL DATA BASE row represents a single record.
13.	In Cassandra column represents a unit of storage.	In TYPICAL DATA BASE column represents an attribute.
14.	In Cassandra, relationships are represented using collections.	In TYPICAL DATA BASE relationships are represented using keys and join etc.

2.What exactly is CQLSH?

Ans. the Cassandra query language shell and explains how to use its commands.

By default, Cassandra provides a prompt Cassandra query language shell **(cqlsh)** that allows users to communicate with it. Using this shell, you can execute **Cassandra Query Language (CQL)**.

Using cqlsh, you can

- define a schema,
- insert data, and
- execute a query.

Starting cqlsh

Start cqlsh using the command **cqlsh** as shown below. It gives the Cassandra cqlsh prompt as output.

```
[hadoop@linux bin]$ cqlsh
```

```
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 2.1.2 | CQL spec 3.2.0 | Native protocol v3]
Use HELP for help.
cqlsh>
```

Cqlsh − As discussed above, this command is used to start the cqlsh prompt. In addition, it supports a few more options as well. The following table explains all the options of **cqlsh** and their usage.

Options	Usage
cqlsh --help	Shows help topics about the options of **cqlsh** commands.
cqlsh --version	Provides the version of the cqlsh you are using.
cqlsh --color	Directs the shell to use colored output.
cqlsh --debug	Shows additional debugging information.
cqlsh --execute cql_statement	Directs the shell to accept and execute a CQL command.
cqlsh --file= **"file name"**	If you use this option, Cassandra executes the command in the given file and exits.
cqlsh --no-color	Directs Cassandra not to use colored output.
cqlsh -u **"user name"**	Using this option, you can authenticate a user. The default user name is: cassandra.
cqlsh-p **"pass word"**	Using this option, you can authenticate a user with a password. The default password is: cassandra.

Cqlsh Commands

Cqlsh has a few commands that allow users to interact with it. The commands are listed below.

Documented Shell Commands

Given below are the Cqlsh documented shell commands. These are the commands used to perform tasks such as displaying help topics, exit from cqlsh, describe,etc.

- **HELP** − Displays help topics for all cqlsh commands.
- **CAPTURE** − Captures the output of a command and adds it to a file.

- **CONSISTENCY** − Shows the current consistency level, or sets a new consistency level.
- **COPY** − Copies data to and from Cassandra.
- **DESCRIBE** − Describes the current cluster of Cassandra and its objects.
- **EXPAND** − Expands the output of a query vertically.
- **EXIT** − Using this command, you can terminate cqlsh.
- **PAGING** − Enables or disables query paging.
- **SHOW** − Displays the details of current cqlsh session such as Cassandra version, host, or data type assumptions.
- **SOURCE** − Executes a file that contains CQL statements.
- **TRACING** − Enables or disables request tracing.

CQL Data Definition Commands

- **CREATE KEYSPACE** − Creates a KeySpace in Cassandra.
- **USE** − Connects to a created KeySpace.
- **ALTER KEYSPACE** − Changes the properties of a KeySpace.
- **DROP KEYSPACE** − Removes a KeySpace
- **CREATE TABLE** − Creates a table in a KeySpace.
- **ALTER TABLE** − Modifies the column properties of a table.
- **DROP TABLE** − Removes a table.
- **TRUNCATE** − Removes all the data from a table.
- **CREATE INDEX** − Defines a new index on a single column of a table.
- **DROP INDEX** − Deletes a named index.

CQL Data Manipulation Commands

- **INSERT** − Adds columns for a row in a table.
- **UPDATE** − Updates a column of a row.
- **DELETE** − Deletes data from a table.
- **BATCH** − Executes multiple DML statements at once.

CQL Clauses

- **SELECT** − This clause reads data from a table
- **WHERE** − The where clause is used along with select to read a specific data.
- **ORDERBY** − The orderby clause is used along with select to read a specific data in a specific order.

3.Explain the Cassandra cluster idea?

Ans._ In this **Cassandra tutorial**, we will go through one of the main parts of the Cassandra database i.e. Cassandra Cluster. Moreover, we will see the meaning of Cluster and different layers in Cluster.
The cluster is a collection of nodes that represents a single system. A cluster in Cassandra is one of the shells in the whole Cassandra database. Many Cassandra Clusters combine together to form the database in Cassandra.
A Cluster is basically the outermost shell or storage unit in a database. The Cassandra Cluster contains many different layers of storage units. Each layer contains the other.

Let's Explore Cassandra Applications

The following are different layers in a Cassandra

a. Node Cluster

Node is the second layer in a cluster. This layer basically comprises of systems or computers or storage units. Each cluster may contain many nodes or systems. These systems or nodes are connected together.
They collectively share data through the replication in Cassandra and independently as well. The replication factor in the nodes allows the user to have a redundancy for the data stored.

b. Keyspace

The keyspace is the next layer of the storage. In a node, there are many keyspaces. These keyspaces are basically the outermost storage unit in a system. They contain the main data. The data distributed according to their properties or areas.

Let's revise Cassandra API

c. Column Families

The next layer is the column families. The keyspace is further divided into column families. These column families have different areas or headings under which the data is distributed. In a keyspace, these column families are categorized into different headings or titles.
These titles further contain different layers of storage units. These column families can also be characterized by tables. The column families differ from the tables through their APIs.

d. Rows

The next layer in the database is of the Rows as according to column families. The Rows are basically the classification under which the column family is divided. These classifi

4.Give an example to demonstrate the class room?

Ans._ A class is the basis of all data in Python, everything is an object in Python, and a class is how an object is defined. They are the foundation of object-oriented

programming and represent real-world things you want to model in your programs. You use a class to instantiate objects, which are specific instances of a class. If you had a House class, you might create a colonial object, a contemporary object, or a log cabin object from that base class. A class defines the general behavior that an entire category of objects may have as well as the information that can be associated with those objects. Classes can inherit from each other which means you can create a class that extends the functionality of an existing class. This is a very common practice in **Object Oriented Python**.

How To Create A Class

A class is usually modeled after a Noun. Classes are things. So we might make a class that represents a Person, House, Vehicle, or Animal. These are all common examples to use when learning about classes. For us, we will create a Vehicle class. What information would we associate with a vehicle, and what behavior would it have? A vehicle may have a type, brand, model and so on. This type of information is stored in **python variables** called attributes. These things also have behaviors. A Vehicle can drive, stop, honk its horn, and so on. Behaviors are contained in functions and a function that is part of a class is called a **method**.

A Vehicle class

```
class Vehicle:
    def __init__(self, brand, model, type):
        self.brand = brand
        self.model = model
        self.type = type
        self.gas_tank_size = 14
        self.fuel_level = 0

    def fuel_up(self):
        self.fuel_level = self.gas_tank_size
        print('Gas tank is now full.')

    def drive(self):
        print(f'The {self.model} is now driving.')
```

Python
Copy
Notice that the gas_tank_size holds a **Python Number** in this case.

Constructing an object

An **instance** of a class is called an object. It's created by calling the class itself as if it were a function. The code below passes in three **python strings** to create a new vehicle object.

```
vehicle_object = Vehicle('Honda', 'Ridgeline', 'Truck')
```

Python
Copy
Accessing attribute values

```
print(vehicle_object.brand)
print(vehicle_object.model)
print(vehicle_object.type)
```

Python
Copy
Honda
Ridgeline
Truck
Calling methods

```
vehicle_object.fuel_up()
vehicle_object.drive()
```

Python
Copy
Gas tank is now full.
The Ridgeline is now driving.
Creating multiple objects

```
vehicle_object = Vehicle('Honda', 'Ridgeline', 'Truck')
a_subaru = Vehicle('Subaru', 'Forester', 'Crossover')
an_suv = Vehicle('Ford', 'Explorer', 'SUV')
```

5. use an example to explain the object?

Ans._ A python is an object-oriented programming language. Almost everything in Python is considered as an object. An object has its own properties(attributes) and behavior(methods).

A class is a blueprint of the objects or can be termed as object constructor for creating objects.

One class can have many objects and value of properties for different objects can be different.

Example of properties and behavior of an object

Let's take the example of car as an object. Its properties will include its color, company name, year of manufacture , price , mileage etc. The behavior of the car will include the functions it can perform, this will include increase speed, decrease speed, apply brakes etc. Object basically related everything with real life objects. Everything we find around us in real life has some properties and some functions.

Example of class and object

Different objects belonging to same class can have different properties. For example, Person(Human) can be treated as a class which has properties such as name, age,gender etc. Every individual can be treated as an object of the class human or Person. Each individual will have different values of the properties of class Person.Everyone will have different names, age and gender.

What is instantiation?

An object is also called an instance of a class. Thus, the process of creating object of a class is known as instantiation.

Defining class in Python

As the function in Python is defined using the keyword 'def'. The keyword 'class' is used to define a class in Python. Since the class is a blueprint of the object, all the common attributes and methods will be declared and defined in the class. Different objects which are created from the class can access those properties and functions. Different objects can hold their own values for properties defined inside the class.

Creating object in Python

Creating object of a class is simple. The name of the class must be known and object can be created as follows −

Object_name= class_name()

Example

Live Demo

```
class Person:
  name=""
  age=0
  city=""
  def display(self):
    print("Name : ",self.name)
    print("Age : ",self.age)
    print("City : ",self.city)
```

```
p1=Person()
p1.name="Rahul"
p1.age=20
p1.city="Kolkata"
p1.display()

print()

p2=Person()
p2.name="Karan"
p2.age=22
p2.city="Bangalore"
p2.display()

print()
p1.display()
```

In the above implementation, p1=Person() is the object instantiation. p1 is the name of the object . We accessed the properties of the class through object p1 and gave them different values and later called the display function to display values of this object.Later,we do the same for second object p2 and display properties of p2.

At the end, we again call display() for object p1 to show that each object holds its own value of properties and those are independent of the other objects.

Output

```
Name : Rahul
Age : 20
City : Kolkata
Name : Karan
Age : 22
City : Bangalore
Name : Rahul
Age : 20
City : Kolkata
```