Q1. What is MongoDB? Explain non-relational databases in short. In which scenarios it is preferred to use MongoDB over SQL databases? Ans: MongoDB, the most popular NoSQL database, is an open-source document-oriented database. 'non-relational' means that MongoDB isn’t based on the table-like relational database structure but provides an altogether different mechanism for storage and retrieval of data. This format of storage is called BSON ( similar to JSON format). NoSQL databases are more scalable and provide superior performance. MongoDB is such a NoSQL database that scales by adding more and more servers and increases productivity with its flexible document model.

A simple MongoDB document Structure:

{ title: 'PW Skills', by: 'Harshit Gupta', type: 'NoSQL' }

MongoDB is preferred over RDBMS in the following scenarios:

Big Data: If you have huge amount of data to be stored in tables, think of MongoDB before RDBMS databases. MongoDB has built-in solution for partitioning and sharding your database. Unstable Schema: Adding a new column in RDBMS is hard whereas MongoDB is schema-less. Adding a new field does not effect old documents and will be very easy. Distributed data Since multiple copies of data are stored across different servers, recovery of data is instant and safe even if there is a hardware failure.

Q2. State and Explain the features of MongoDB.

Ans: Features of MongoDB: Document Oriented: MongoDB stores the main subject in the minimal number of documents and not by breaking it up into multiple relational structures like RDBMS. For example, it stores all the information of a computer in a single document called Computer and not in distinct relational structures like CPU, RAM, Hard disk, etc. Indexing: Without indexing, a database would have to scan every document of a collection to select those that match the query which would be inefficient. So, for efficient searching Indexing is a must and MongoDB uses it to process huge volumes of data in very less time. Scalability: MongoDB scales horizontally using sharding (partitioning data across various servers). Data is partitioned into data chunks using the shard key, and these data chunks are evenly distributed across shards that reside across many physical servers. Also, new machines can be added to a running database. Replication and High Availability: MongoDB increases the data availability with multiple copies of data on different servers. By providing redundancy, it protects the database from hardware failures. If one server goes down, the data can be retrieved easily from other active servers which also had the data stored on them. Aggregation: Aggregation operations process data records and return the computed results. It is similar to the GROUPBY clause in SQL. A few aggregation expressions are sum, avg, min, max, etc

Q3. Write a code to connect MongoDB to Python. Also, create a database and a collection in MongoDB.

Ans: Code To Connect MongoDB

​

# importing module

from pymongo import MongoClient

# creation of MongoClient

client=MongoClient()

# Connect with the portnumber and host

client = MongoClient(“mongodb://localhost:27017/”)

# Access database

mydatabase = client[‘name\_of\_the\_database’]

# Access collection of the database

mycollection=mydatabase[‘myTable’]

# dictionary to be added in the database

rec={

title: 'MongoDB and Python',

description: 'MongoDB is no SQL database',

tags: ['mongodb', 'database', 'NoSQL'],

viewers: 104

}

# inserting the data in the database

rec = mydatabase.myTable.insert(record)

​

Q4. Using the database and the collection created in question number 3, write a code to insert one record,

and insert many records. Use the find() and find\_one() methods to print the inserted record.

Ans:

# importing Mongoclient from pymongo

from pymongo import MongoClient

​

​

# Making Connection

myclient = MongoClient("mongodb://localhost:27017/")

​

# database

db = myclient["GFG"]

​

# Created or Switched to collection

# names: GeeksForGeeks

collection = db["Student"]

​

# Creating Dictionary of records to be

# inserted

single\_record = {

"record1": { "\_id": 5,

"name": "Mohit",

"Roll No": "1005",

"Branch": "CSE"},

}

collection.insert\_one(single\_record)

​

Multiple\_records = {

"record1": { "\_id": 6,

"name": "Anshul",

"Roll No": "1006",

"Branch": "CSE"},

​

"record2": { "\_id": 7,

"name": "Abhinav",

"Roll No": "1007",

"Branch": "ME"}

}

collection.insert\_many(record)

​

#use find\_one method

​

collection.findOne() #With empty query specification it returns the first document in the collection:

collection.findOne({"id":"6"})

Q5: Explain how you can use the find() method to query the MongoDB database. Write a simple code to

demonstrate this.

Ans:

In MongoDB, **find()** method is used to select documents in a collection and return a cursor to the selected documents. Cursor means a pointer that points to a document, when we use find() method it returns a pointer on the selected documents and returns one by one. If we want to return pointer on all documents then use empty() parameter that returns all documents one by one. It takes only some optional parameters. The first optional parameter is the selection criteria on which we want to return a cursor. To return all documents in a collection use empty document({}). Using this method you can also replace [embedded documents](https://www.geeksforgeeks.org/mongodb-embedded-documents/). You can also use this method in multi-document transactions. If you use this method in the mongo shell, then the shell will automatically iterate the cursor to display up to 20 documents in the collection, if you want to continue then type*it*or you can manually iterate the result of the find() method by assigning the returned cursor to a variable with the var keyword. You can also modify the behavior of this method using cursor methods.

**Syntax:**

*db.Collection\_name.find(selection\_criteria, projection,options)*

**Optional parameters:**

* **selection\_criteria:** It specifies selection criteria. To return all documents in a collection use empty document({}). The type of this parameter is document.
* **projection:** It specifies the fields to return in the documents that match the selection criteria. To return all fields in the matching documents, remove this parameter. It is of the document type.
* **options:** It specifies some additional options for the selection\_criteria parameter. It modifies the behavior of selection\_criteria and also affects the results that will be returned.

This document takes:

{ field1: <value1>, field2: <value2> ... }

Here if the value of the field is *1/true* then it specifies the inclusion of the field, or if the value of the field is *0/false* then it specifies the exclusion of the field.

**Return:**

It returns a cursor to the documents that match the selection criteria. When the find() method returns documents, the method is actually returning a cursor to the documents.

**Examples:**

In the following example, we are working with:

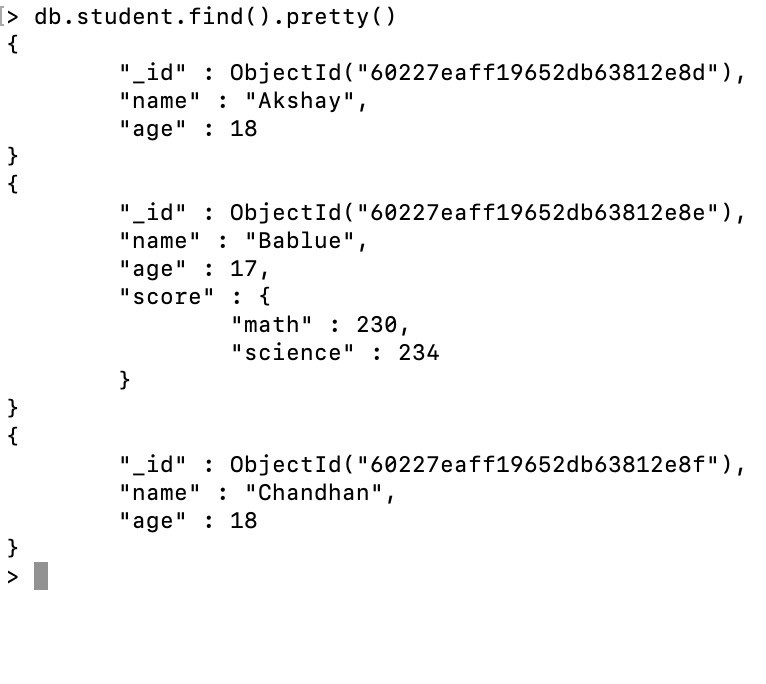
***Database:****gfg*

***Collections:****student*

***Document:****Three documents contains the details of the students*

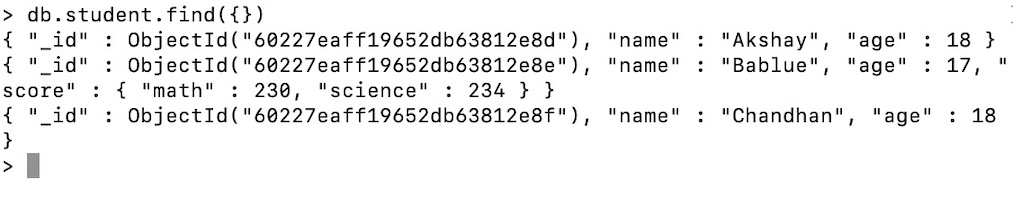
* **Find all the documents present in the collection:**

db.student.find()



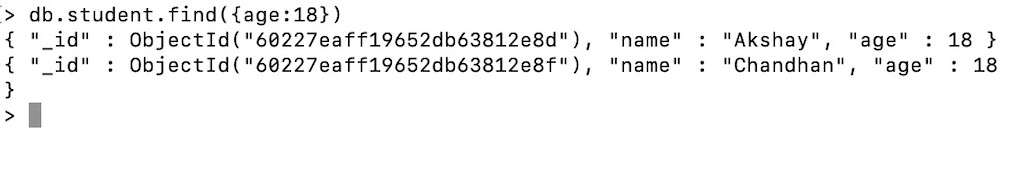
* **Find all the documents present in the collection by passing empty document:**

db.student.find({})



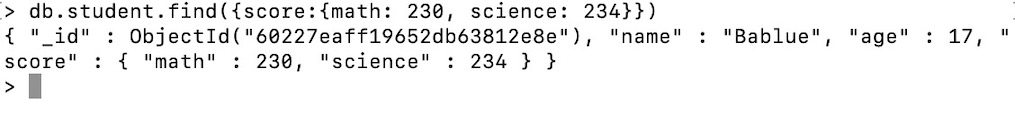
* **Find all the document that matches the given filter query(i.e., age:18):**

db.student.find({age:18})



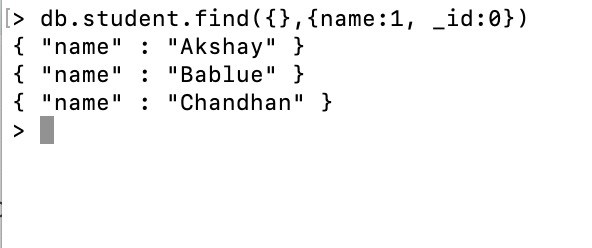
* **Find the embedded document that matches the given filter query:**

db.student.find({score:{math: 230, science: 234}})



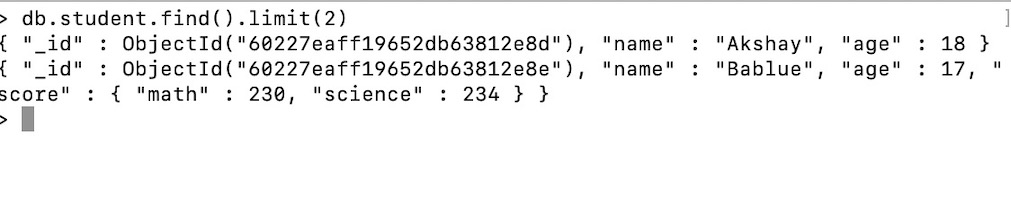
* **Display only the specified fields(Using Projection):**

db.student.find({},{name:1, \_id:0})



* **Display only two documents using the limit() method:**

db.student.find().limit(2)



Q6. Explain the sort() method. Give an example to demonstrate sorting in MongoDB.

Ans: The sort() method specifies the order in which the query returns the matching documents from the given collection. You must apply this method to the cursor before retrieving any documents from the database. It takes a document as a parameter that contains a field: value pair that defines the sort order of the result set. The value is 1 or -1 specifying an ascending or descending sort respectively.

If a sort returns the same result every time we perform on same data, then such type of sort is known as a stable sort.

If a sort returns a different result every time we perform on same data, then such type of sort is known as unstable sort.

MongoDB generally performs a stable sort unless sorting on a field that holds duplicate values.

We can use limit() method with sort() method, it will return first m documents, where m is the given limit.

MongoDB can find the result of the sort operation using indexes.

If MongoDB does not find sort order using index scanning, then it uses top-k sort algorithm.

Syntax:

db.Collection\_Name.sort({field\_name:1 or -1})

Parameter:

The parameter contains a field: value pair that defines the sort order of the result set. The value is 1 or -1 that specifies an ascending or descending sort respectively. The type of parameter is a document.

Return:

It returns the documents in sorted order.

Examples:

In the following examples, we are working with:

Database: gfg

Collections: student

Document: Four documents contains name and age of the students.

Return all the documents in ascending order of the age:

db.student.find().sort({age:1})

Return all the documents in descending order of the age:

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

Return all the documents in the ascending order of the age:

db.student.find().sort({name:1})

Return all the documents in the descending order of the age:

db.student.find().sort({name:-1})

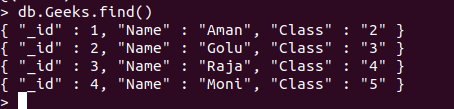
Q7. Explain why delete\_one(), delete\_many(), and drop() is used.

**Deleting document from Collection or Database**

In MongoDB, a single document can be deleted by the method delete\_one(). The first parameter of the method would be a query object which defines the document to be deleted. If there are multiple documents matching the filter query, only the first appeared document would be deleted.

**Note:** Deleting a document is the same as deleting a record in the case of SQL.

Consider the sample database:



**Examples:**

**Python**

|  |
| --- |
| # Python program to demonstrate  # delete\_one      **import** pymongo      # creating Mongoclient object to  # create database with the specified  # connection URL  students **=** pymongo.MongoClient('localhost', 27017)    # connecting to a database with  # name GFG  Db **=** students["GFG"]    # connecting to a collection with  # name Geeks  coll **=** Db["Geeks"]    # creating query object  myQuery **=**{'Class':'2'}  coll.delete\_one(myQuery)    # print collection after deletion:  **for** x **in** coll.find():      print(x) |

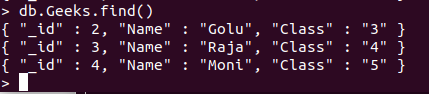
**Output :**

'\_id': 2.0, 'Name': 'Golu', 'Class': '3'}

{'\_id': 3.0, 'Name': 'Raja', 'Class': '4'}

{'\_id': 4.0, 'Name': 'Moni', 'Class': '5'}

**MongoDB Shell:**



[**MongoDB**](https://www.geeksforgeeks.org/mongodb-and-python/) is a general-purpose, document-based, distributed database built for modern application developers and the cloud. It is a document database, which means it stores data in JSON-like documents. This is an efficient way to think about data and is more expressive and powerful than the traditional table model.

**Delete\_many()**

Delete\_many() is used when one needs to delete more than one document. A query object containing which document to be deleted is created and is passed as the first parameter to the delete\_many().

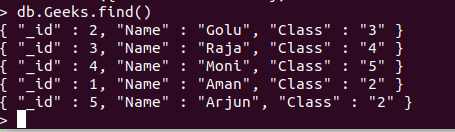
**Syntax:**

collection.delete\_many(filter, collation=None, hint=None, session=None)

***Parameters:***

* *‘****filter****’ : A query that matches the document to delete.*
* *‘****collation****’ (optional) : An instance of class: ‘~pymongo.collation.Collation’. This option is only supported on MongoDB 3.4 and above.*
* *‘****hint****’ (optional) : An index to use to support the query predicate. This option is only supported on MongoDB 3.11 and above.*
* *‘****session****’ (optional) : a class:’~pymongo.client\_session.ClientSession’.*

**Sample Database:**



**Example 1:** Deleting all the documents where the name starts with ‘A’.

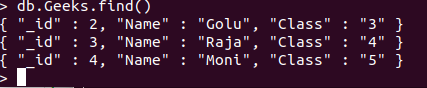
* Python3

|  |
| --- |
| **import** pymongo      client **=** pymongo.MongoClient("mongodb:**//**localhost:27017**/**")    # Connecting to the database  mydb **=** client["GFG"]    # Connecting the to collection  col **=** mydb["Geeks"]    query **=** {"Name": {"$regex": "^A"}}  d **=** col.delete\_many(query)    print(d.deleted\_count, " documents deleted !!") |

**Output:**

2 documents deleted !!

**MongoDB Shell:**



**Example 2:**

* Python3

|  |
| --- |
| **import** pymongo      client **=** pymongo.MongoClient("mongodb:**//**localhost:27017**/**")    # Connecting to the database  mydb **=** client["GFG"]    # Connecting the to collection  col **=** mydb["Geeks"]    query **=** {"Class": '3'}  d **=** col.delete\_many(query)    print(d.deleted\_count, " documents deleted !!") |

**Output:**

1 documents deleted !!

**MongoDB Shell:**

