***MongoDB***

**Q1. What is MongoDB? Explain non-relational databases in short. In which scenarios it is preferred to use**

**MongoDB over SQL databases?**

MongoDB is a popular document-oriented, NoSQL database management system that provides high performance, scalability, and flexibility. It stores data in flexible, semi-structured JSON-like documents that allow for dynamic schema designs and easier integration with modern web and mobile applications.

Non-relational databases, or NoSQL databases, are database management systems that do not use a traditional table-based relational schema to store data. Instead, they use different data models like key-value pairs, documents, and graphs. Non-relational databases are generally more flexible and scalable than traditional relational databases, making them ideal for handling large volumes of unstructured and semi-structured data.

MongoDB is preferred over SQL databases in scenarios where you need to store large amounts of unstructured data, such as log files, sensor data, or social media feeds. MongoDB's document-based approach allows you to store data in a way that's more natural for your application's data model, which can lead to faster development cycles and better performance. It also allows for more flexible schema design, which can be especially useful for agile development processes. Additionally, MongoDB's built-in replication and sharding capabilities make it easier to scale horizontally, which can help ensure high availability and performance as your application grows.

**Q2. State and Explain the features of MongoDB.**

MongoDB is a popular NoSQL database management system that offers a wide range of features that make it a popular choice for modern web and mobile applications. Here are some of the key features of MongoDB:

1. Document-based storage: MongoDB stores data in flexible, semi-structured JSON-like documents, which makes it easier to represent complex data structures and dynamic schemas.
2. High performance: MongoDB is designed for high performance, with support for indexing, caching, and other features that can help improve query performance.
3. Flexible querying: MongoDB supports a powerful querying language that allows you to filter and sort data based on specific criteria.
4. Horizontal scaling: MongoDB supports horizontal scaling, which allows you to distribute your data across multiple servers to improve performance and availability.
5. Automatic sharding: MongoDB includes built-in support for automatic sharding, which makes it easy to scale horizontally as your application grows.
6. Replication: MongoDB supports automatic replication, which allows you to create multiple copies of your data to improve availability and ensure data durability.
7. Rich data model: MongoDB supports a rich data model that includes embedded documents, arrays, and other data structures, which makes it easier to represent complex data relationships.
8. Dynamic schema: MongoDB supports a dynamic schema, which means that you can add or remove fields from your documents without having to modify your schema.
9. Ad hoc queries: MongoDB allows you to perform ad hoc queries on your data, which makes it easier to explore and analyze your data in real-time.
10. Integration with other technologies: MongoDB supports integration with a wide range of other technologies, including Hadoop, Spark, and other big data platforms, which makes it a popular choice for building modern data-driven applications.

Overall, MongoDB's features make it an ideal choice for modern web and mobile applications that need to handle large volumes of unstructured data, scale horizontally, and support real-time querying and analysis.

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

To connect MongoDB to Python, you can use the pymongo package. Here is an example code that shows how to connect to MongoDB, create a database, and create a collection:

**import pymongo**

**# Establish a connection to the MongoDB server**

**client = pymongo.MongoClient('mongodb://localhost:27017/')**

**# Create a new database**

**mydb = client['mydatabase']**

**# Create a new collection**

**mycol = mydb['customers']**

**# Insert a document into the collection**

**mydict = { "name": "John", "address": "Highway 37" }**

**x = mycol.insert\_one(mydict)**

**# Print the inserted document's ID**

**print(x.inserted\_id)**

In this code, we first establish a connection to the MongoDB server by creating a MongoClient object with the URL of the server. We then create a new database called mydatabase and a new collection called customers within that database. Finally, we insert a new document into the customers collection and print its ID.

Note that you will need to have MongoDB installed on your local machine and running in order to run this code. Additionally, you will need to install the pymongo package using pip or another package manager.

**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.**

Sure, here is the code to insert one record and insert many records into the customers collection in the mydatabase database created in the previous question, and then use the find() and find\_one() methods to print the inserted record(s):

**import pymongo**

**# Establish a connection to the MongoDB server**

**client = pymongo.MongoClient('mongodb://localhost:27017/')**

**# Select the database and collection**

**mydb = client['mydatabase']**

**mycol = mydb['customers']**

**# Insert one record**

**mydict = { "name": "Jane", "address": "Highway 38" }**

**x = mycol.insert\_one(mydict)**

**print("Inserted record ID:", x.inserted\_id)**

**# Insert many records**

**mylist = [**

**{ "name": "Mike", "address": "123 Main St." },**

**{ "name": "Sarah", "address": "456 Elm St." },**

**{ "name": "David", "address": "789 Oak St." }**

**]**

**x = mycol.insert\_many(mylist)**

**print("Inserted record IDs:", x.inserted\_ids)**

**# Find one record**

**result = mycol.find\_one({ "name": "Jane" })**

**print("Found one record:", result)**

**# Find all records**

**results = mycol.find()**

**print("Found", results.count(), "records:")**

**for result in results:**

**print(result)**

In this code, we first establish a connection to the MongoDB server, select the mydatabase database and the customers collection, and then insert one record and many records using the insert\_one() and insert\_many() methods, respectively. We then print the IDs of the inserted records.

Next, we use the find\_one() method to retrieve one record from the collection with the name "Jane", and print the result. We also use the find() method to retrieve all records in the collection, and print each record in the results. Note that we use the count() method to get the total number of records found by the find() method.

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

**demonstrate this.**

The find() method in MongoDB is used to query the database and retrieve documents that match a specified set of criteria. The method returns a cursor object, which can be used to iterate over the documents returned by the query. Here is an example code that demonstrates how to use the find() method to query a collection in a MongoDB database

**import pymongo**

**# Establish a connection to the MongoDB server**

**client = pymongo.MongoClient('mongodb://localhost:27017/')**

**# Select the database and collection**

**mydb = client['mydatabase']**

**mycol = mydb['customers']**

**# Query the collection using the find() method**

**query = { "address": "Highway 37" }**

**results = mycol.find(query)**

**# Print the documents returned by the query**

**for result in results:**

**print(result)**

In this code, we first establish a connection to the MongoDB server, select the mydatabase database and the customers collection. We then define a query using a dictionary with the key "address" and the value "Highway 37". We pass this query to the find() method of the collection, which returns a cursor object containing all documents in the collection that match the query.

We then iterate over the cursor object using a for loop, and print each document returned by the query. Note that the documents are printed in the order that they were inserted into the collection, which may not be the order in which they were returned by the query.

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

The sort() method in MongoDB is used to sort the documents in a collection based on one or more fields. It takes one parameter, which is a dictionary that specifies the fields to sort on and the sort order for each field.

The keys of the dictionary correspond to the fields to sort on, and the values specify the sort order for each field. A value of 1 specifies ascending order, while a value of -1 specifies descending order. If multiple fields are specified, the sort order is applied in the order that the fields appear in the dictionary.

Here is an example code that demonstrates how to use the sort() method to sort the documents in a collection:

**import pymongo**

**# Establish a connection to the MongoDB server**

**client = pymongo.MongoClient('mongodb://localhost:27017/')**

**# Select the database and collection**

**mydb = client['mydatabase']**

**mycol = mydb['customers']**

**# Insert some sample data into the collection**

**mycol.insert\_many([**

**{ "name": "John", "age": 35 },**

**{ "name": "Jane", "age": 30 },**

**{ "name": "Mike", "age": 40 }**

**])**

**# Sort the documents in the collection by age in ascending order**

**results = mycol.find().sort("age", 1)**

**# Print the sorted documents**

**for result in results:**

**print(result)**

In this code, we first establish a connection to the MongoDB server, select the mydatabase database and the customers collection, and insert some sample data into the collection.

We then use the find() method to retrieve all documents in the collection, and pass the result to the sort() method with the argument "age" and a value of 1, which specifies that the documents should be sorted by age in ascending order.

Finally, we iterate over the cursor object returned by the sort() method using a for loop, and print each document in the sorted order.

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

In MongoDB, the delete\_one() and delete\_many() methods are used to remove one or multiple documents from a collection, respectively. The drop() method is used to remove an entire collection from the database.

The delete\_one() method takes a single parameter, which is a dictionary that specifies the criteria for matching the document to be deleted. If multiple documents match the criteria, only the first document that is encountered will be deleted.

The delete\_many() method takes the same parameter as delete\_one(), but it deletes all documents that match the criteria.

Both delete\_one() and delete\_many() return a DeleteResult object that contains information about the number of documents that were deleted.

The drop() method takes no parameters and removes the entire collection from the database. This method is useful when you want to delete a large number of documents or when you want to start fresh with a new collection.

In summary, delete\_one() and delete\_many() are used to remove one or multiple documents from a collection based on specified criteria, while drop() is used to remove an entire collection from the database.