Assignment

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

MongoDB is a popular open-source NoSQL (non-relational) database management system that stores data in flexible, JSON-like documents called BSON (Binary JSON) instead of traditional tables with rows and columns. It is designed to be scalable, flexible, and efficient, making it suitable for handling large volumes of data and diverse data models. MongoDB is part of the NoSQL movement, which emphasizes non-tabular, distributed, and flexible data storage.

Non-Relational Databases:

Non-relational databases, often referred to as NoSQL databases, deviate from the traditional relational database model. They are designed to handle various types of data and are particularly well-suited for scenarios where the data structure is dynamic and evolving. Non-relational databases offer advantages such as flexibility, scalability, and the ability to handle large volumes of data with varying structures.

Key characteristics of non-relational databases include:

Schema-less: Non-relational databases are schema-less, allowing you to insert data without a predefined schema. This flexibility is beneficial in scenarios where the data structure is subject to frequent changes.
NoSQL (Not Only SQL): Non-relational databases support various data models, including key-value pairs, document stores, column-family stores, and graph databases. The choice of the data model depends on the specific requirements of the application.
Scalability: Non-relational databases are designed to be horizontally scalable, allowing them to handle increased load by adding more servers to a distributed architecture.
Performance: Non-relational databases often provide high performance for certain types of operations, such as read and write-intensive tasks, due to their optimized storage and retrieval mechanisms.

Scenarios for MongoDB over SQL Databases:

MongoDB is often preferred over traditional SQL databases in the following scenarios:

Flexible Schema: When the data structure is not fixed and evolves over time, MongoDB's schema-less design allows for dynamic changes without requiring alterations to the database schema.
Document-Oriented Data: MongoDB is well-suited for document-oriented data, where each record can have a complex and nested structure. This makes it suitable for scenarios like content management systems, catalogs, and product information.
Big Data and Real-time Applications: MongoDB is designed to handle large volumes of data and provides horizontal scalability, making it suitable for big data applications and real-time data processing.
Agile Development: In agile development environments, where requirements change frequently, MongoDB's flexibility allows developers to adapt to changing needs without significant overhead.
Geospatial Data: MongoDB has built-in support for geospatial indexing and queries, making it a good choice for applications dealing with location-based data, such as geolocation services.

It's important to note that the choice between MongoDB and traditional SQL databases depends on the specific requirements and characteristics of the application. Each type of database has its strengths and weaknesses, and the decision should be based on factors such as data structure, scalability needs, and development preferences.

Q2. State and Explain the features of MongoDB.
Ans:MongoDB is a NoSQL, document-oriented database management system that provides a flexible and scalable solution for handling diverse data types. Here are some key features of MongoDB:

Document-Oriented:
- MongoDB stores data in BSON (Binary JSON) documents, which are JSON-like, binary-encoded documents. Each document can have a different structure, allowing for flexible and dynamic data models.

Dynamic Schema:
- MongoDB has a dynamic schema, meaning that documents in the same collection can have different fields. Fields can be added or removed from documents without affecting other documents in the collection.

Scalability:
- MongoDB is designed to scale horizontally, allowing for the distribution of data across multiple servers or clusters. This horizontal scaling approach helps handle large amounts of data and high traffic volumes.

Indexing:

- MongoDB supports various types of indexes, including compound indexes and geospatial indexes. Indexing helps improve query performance by facilitating faster data retrieval.

Query Language:

- MongoDB supports a rich query language that includes a wide range of operators and expressions. Queries can be based on field values, ranges, and even regular expressions, providing powerful search capabilities.

Aggregation Framework:

- MongoDB includes a powerful aggregation framework for performing data transformations and computations. It allows users to filter, group, and project data, making it well-suited for complex analytics and reporting.

High Performance:

- MongoDB provides high performance for both read and write operations. It uses memory-mapped files for data storage and supports in-memory computing for certain operations, contributing to its overall speed.

Automatic Sharding:

- MongoDB supports automatic sharding, which involves partitioning data across multiple servers or clusters. This helps distribute the workload and ensures efficient use of resources.

Geospatial Indexing and Queries:

- MongoDB has built-in support for geospatial data, allowing the storage and retrieval of location-based information. Geospatial indexes enable efficient queries based on geographic coordinates.

Replication:

- MongoDB supports replication, allowing the creation of multiple copies (replica sets) of data across different servers. Replication enhances fault tolerance, data availability, and read scalability.

JSON/BSON Format:

- Data in MongoDB is represented in a JSON-like format (BSON), making it easy to work with and integrate into applications that use JSON.

Security Features:

- MongoDB includes security features such as authentication, role-based access control, and encryption, ensuring the protection of sensitive data.

Schema Validation:

- MongoDB allows the definition of JSON-like schema validation rules at the collection level, enforcing data integrity and consistency.

Agile Development:

- MongoDB's flexible schema and dynamic nature make it well-suited for agile development environments, where requirements may change frequently.

MongoDB's combination of flexibility, scalability, and performance makes it a popular choice for a variety of applications, including content management systems, e-commerce platforms, mobile applications, and big data analytics.

Q3. Write a code to connect MongoDB to Python. Also, create a database and a collection in MongoDB.
Ans:To connect MongoDB to Python, you can use the `pymongo` library, which is the official Python driver for MongoDB. If you haven't installed it yet, you can do so by running:

bash

Copy code

Now, let's write a Python script to connect to MongoDB, create a database, and a collection:

python

Copy code

```
import


            "mongodb+srv://username:password@cluster.mongodb.net/test"





                     "my_database"


                            "my_collection"


            "name"   "John"   "age"   30   "city"   "New York"




for          in
 print
```

Replace `"mongodb+srv://username:password@cluster.mongodb.net/test"` with your MongoDB connection string, which can be obtained from your MongoDB Atlas dashboard or your local MongoDB instance.

Explanation:

Connecting to MongoDB:

python

Copy code

The `MongoClient` is used to connect to the MongoDB server using the connection string.
Creating or Getting a Database:

python

Copy code

```
                 "my_database"
```

The script creates a database named "my_database" or retrieves an existing one.
Creating or Getting a Collection:

python

Copy code

```
                         "my_collection"
```

The script creates a collection named "my_collection" within the database or retrieves an existing one.
Inserting a Document:

python

Copy code

```
        "name"  "John"  "age"  30  "city"  "New York"
```

A document is inserted into the collection.
Querying and Printing Documents:

python

Copy code

```
for          in
 print
```

The script queries and prints all documents in the collection.
Closing the Connection:

python

Copy code

It's important to close the connection when you're done interacting with MongoDB.

Make sure to replace the connection string, and you can customize the database name,

collection name, and document contents according to your requirements.

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:Here's a Python script that uses the MongoDB database and collection created in question number 3 to insert one record and insert many records. It also demonstrates the use of `find()` and `find_one()` methods to print the inserted records:
python

Copy code

```
import


            "mongodb+srv://username:password@cluster.mongodb.net/test"
```

```
                     "my_database"


                            "my_collection"


              "name"   "Alice"   "age"   25   "city"   "London"




  "name"   "Bob"   "age"   30   "city"   "Paris"
  "name"   "Charlie"   "age"   22   "city"   "Berlin"
  "name"   "David"   "age"   35   "city"   "Tokyo"






print "All records:"

for          in
 print


print "\nOne record:"
                                     "name"   "Alice"
print
```

Replace `"mongodb+srv://username:password@cluster.mongodb.net/test"` with your MongoDB connection string.

Explanation:

Insert One Record:

python

Copy code

```
            "name"   "Alice"   "age"   25   "city"   "London"
```

Inserts a single record into the collection.
Insert Many Records:

python

Copy code

```
  "name"   "Bob"   "age"   30   "city"   "Paris"
  "name"   "Charlie"   "age"   22   "city"   "Berlin"
  "name"   "David"   "age"   35   "city"   "Tokyo"
```

Inserts multiple records into the collection.
Print All Records Using `find()`:

python

Copy code

```
print "All records:"

for           in
 print
```

Retrieves and prints all records in the collection using the `find()` method.
Print One Record Using `find_one()`:

python

Copy code

```
print "\nOne record:"
                                    "name"   "Alice"
print
```

Retrieves and prints one record in the collection using the `find_one()` method.

Remember to customize the record data and search criteria based on your specific use case.

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

Ans:The `find()` method in MongoDB is used to query documents in a collection. It allows you to specify conditions or criteria based on which documents are retrieved. The `find()` method returns a cursor, and you can iterate over the cursor to access the documents that match the specified criteria.

Here's a simple example demonstrating how to use the `find()` method to query a MongoDB database:

python

Copy code

```
import

             "mongodb+srv://username:password@cluster.mongodb.net/test"



                      "my_database"

                             "my_collection"


  "name"   "Alice"   "age"  25  "city"   "London"
  "name"   "Bob"   "age"  30  "city"   "Paris"
  "name"   "Charlie"   "age"  22  "city"   "Berlin"
  "name"   "David"   "age"  35  "city"   "Tokyo"



                   "age"   "$gt"  25
```

```
print "Matching documents:"
for          in
 print
```

Explanation:

Inserting Sample Data:

python

Copy code

```
  "name"   "Alice"   "age"  25   "city"   "London"
  "name"   "Bob"   "age"  30   "city"   "Paris"
  "name"   "Charlie"   "age"  22   "city"   "Berlin"
  "name"   "David"   "age"  35   "city"   "Tokyo"
```

Inserts some sample data into the MongoDB collection for demonstration purposes.
Querying with `find()`:

python

Copy code

```
                "age"    "$gt"  25
```

Defines a query criteria to retrieve documents where the "age" field is greater than 25.
The `$gt` operator stands for "greater than."
Printing Matching Documents:

python

Copy code

```
print "Matching documents:"
for          in
```

```
print
```

Prints the documents that match the specified criteria.

Adjust the query criteria based on your specific requirements. The `find()` method supports a wide range of query operators and expressions to filter documents based on various conditions.

Q6. Explain the sort() method. Give an example to demonstrate sorting in MongoDB.
Ans:In MongoDB, the `sort()` method is used to sort the documents in a collection based on the specified criteria. The `sort()` method takes one parameter, which is a document that defines the sorting criteria. Each field in the document represents a field in the documents to sort, and the associated value indicates the sort order (ascending or descending).

The basic syntax of the `sort()` method is as follows:

python

Copy code

- `field1`, `field2`, etc.: The fields to sort by.
- `order1`, `order2`, etc.: The order of sorting for each field. Use `1` for ascending order and `-1` for descending order.

Here's an example to demonstrate sorting in MongoDB:

Let's assume we have a collection named "students" with documents representing students' information:

python

Copy code

```
  "name"  "Alice"  "age"  25  "score"  90
  "name"  "Bob"  "age"  30  "score"  85
  "name"  "Charlie"  "age"  22  "score"  95
  "name"  "David"  "age"  35  "score"  88
```

Now, let's use the `sort()` method to retrieve the documents sorted by the "score" field in descending order:

python

Copy code

```
import


            "mongodb+srv://username:password@cluster.mongodb.net/test"





                     "my_database"


                            "students"


                                              "score"   1


print "Sorted documents by score (descending):"
for          in
 print
```

In this example, we use the `sort("score", -1)` to sort the documents based on the "score" field in descending order (`-1`). The output would be:

python

Copy code

```

 '_id'           '...'   'name'  'Charlie'  'age'  22  'score'  95
 '_id'           '...'   'name'  'Alice'  'age'  25  'score'  90
 '_id'           '...'   'name'  'David'  'age'  35  'score'  88
 '_id'           '...'   'name'   'Bob'   'age'   30   'score'   85
```

Adjust the field and order according to your sorting requirements. The `sort()` method is useful for arranging documents in a specific order for retrieval or analysis.

Q7. Explain why delete_one(), delete_many(), and drop() is used.
Ans:In MongoDB, the `delete_one()`, `delete_many()`, and `drop()` methods are used for removing documents from a collection or dropping an entire collection. Here's an explanation of each method:

`delete_one(filter)` Method:

- The `delete_one()` method is used to delete a single document that matches the specified filter criteria.
- Syntax:
- python
- Copy code

```
                    filter
```

-
- Example:
- python
- Copy code

```
                   "name"   "Alice"
```

-
- This method removes the first document that matches the specified filter.

`delete_many(filter)` Method:

- The `delete_many()` method is used to delete multiple documents that match the specified filter criteria.
- Syntax:
- python
- Copy code

`filter`

-
- Example:
- python
- Copy code

`"age"` `"$gt"` `30`

-
- This method removes all documents that match the specified filter.

`drop()` Method:

- The `drop()` method is used to remove an entire collection from the database. It effectively deletes all documents and the collection itself.
- Syntax:
- python
- Copy code
-
- Example:
- python
- Copy code
-
- This method is irreversible and permanently removes the collection.

These methods are essential for managing the data in a MongoDB database. Use them carefully, especially when working with production data, as they permanently delete documents or collections.

Common use cases:

- `delete_one()` and `delete_many()`:
    - Removing specific documents based on filter criteria.
    - Cleaning up outdated or irrelevant data.
- `drop()`:
    - Dropping an entire collection when it is no longer needed.
    - Resetting a collection during development or testing.

It's important to note that the `delete_one()` and `delete_many()` methods are used to remove documents, whereas the `drop()` method is used to delete entire collections. Always exercise caution when performing delete operations, especially in production environments, to avoid unintentional data loss. Consider taking backups before executing critical delete operations.