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

MongoDB: MongoDB is a popular open-source NoSQL database that uses a document-oriented data model. It stores data in flexible, semi-structured BSON (Binary JSON) format. MongoDB is known for its scalability, flexibility, and the ability to handle unstructured or semi-structured data efficiently. It is widely used in modern web and mobile applications for its ease of development and scalability.

Non-relational databases: Non-relational databases, also known as NoSQL databases, are a category of databases that do not follow the traditional relational model of SQL databases. They are designed to handle large volumes of unstructured or semi-structured data and offer more flexibility. NoSQL databases can be document-oriented, key-value stores, column-family stores, or graph databases.

Scenarios to prefer MongoDB over SQL databases:

When you need to store and query unstructured or semi-structured data, such as JSON documents. In situations that require horizontal scalability, where you can add more servers to handle increased data loads. Rapid application development and prototyping due to its flexible schema. When you need to work with large amounts of data or big data. Real-time analytics and IoT (Internet of Things) applications that require fast data ingestion and querying.

Q2. State and Explain the features of MongoDB.

MongoDB has several key features, including:

1.Flexible Schema: MongoDB does not require a fixed schema, allowing you to work with data that doesn't fit neatly into tables. This flexibility is well-suited for dynamic and evolving data.

2.Highly Scalable: MongoDB can scale horizontally by distributing data across multiple servers, making it suitable for handling large amounts of data and high traffic.

3.Document-Oriented: MongoDB stores data in BSON (Binary JSON) documents, making it easy to work with JSON-like data structures.

4.Rich Query Language: MongoDB supports a powerful query language for filtering and searching data, including geospatial queries.

5.Indexes: Indexes can be created to improve query performance. MongoDB automatically creates a unique index on the _id field for fast lookups.

6.Aggregation Framework: MongoDB provides an expressive and efficient aggregation framework for complex data transformation and analysis.

7.Geospatial Data Support: MongoDB offers geospatial indexing and queries for location-based data.

8.Replication and High Availability: It supports replica sets for data redundancy and automatic failover to ensure high availability.

9.Horizontal Partitioning: Sharding allows data distribution across multiple servers to support growth and scalability.

10.Security: MongoDB provides authentication, authorization, and encryption features to secure your data.

11.ACID Transactions: MongoDB supports multi-document transactions, ensuring data consistency in complex operations.

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

To connect MongoDB to Python and create a database and a collection, you can use the pymongo library. First, make sure you have pymongo installed. You can install it using pip:

In [ ]:
```
pip install pymongo
```

Here's a Python code to connect to MongoDB and create a database and collection:

In [ ]:
```
import pymongo

# Create a connection to the MongoDB server (by default, it connects to localhos
client = pymongo.MongoClient("mongodb://localhost:27017/")

# Create or access a database
mydb = client["mydatabase"]

# Create or access a collection
mycollection = mydb["mycollection"]
```

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.

In [ ]:
```
# Insert one record
record_one = {"name": "John", "age": 30}
inserted_record = mycollection.insert_one(record_one)
print(f"Inserted record ID: {inserted_record.inserted_id}")

# Insert many records
records_many = [
    {"name": "Alice", "age": 25},
    {"name": "Bob", "age": 35},
    {"name": "Carol", "age": 28},
]
inserted_records = mycollection.insert_many(records_many)
```

```python
print(f"Inserted {len(inserted_records.inserted_ids)} records")

# Find and print one record
one_record = mycollection.find_one({"name": "John"})
print("One Record:")
print(one_record)

# Find and print all records
all_records = mycollection.find()
print("All Records:")
for record in all_records:
    print(record)
```

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 and retrieve documents from a collection based on specified criteria. You can pass a query document as an argument to find() to filter the documents that match the query criteria.

Here's a simple code example to demonstrate the use of find():

In [ ]:
```python
# Query for documents where age is greater than 30
query = {"age": {"$gt": 30}}
results = mycollection.find(query)

print("Documents with age > 30:")
for doc in results:
    print(doc)
```

In this code, we query the "mycollection" collection for documents where the "age" field is greater than 30. The results will include all documents that match this criteria.

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

The sort() method in MongoDB is used to sort the results of a query in ascending or descending order based on one or more fields. It allows you to control the order in which documents are returned.

Here's an example to demonstrate sorting in MongoDB:

In [ ]:
```python
# Sort documents by age in ascending order
sorted_results = mycollection.find().sort("age")

print("Sorted by age (ascending):")
for doc in sorted_results:
    print(doc)

# Sort documents by name in descending order
sorted_results_desc = mycollection.find().sort("name", pymongo.DESCENDING)

print("Sorted by name (descending):")
for doc in sorted_results_desc:
    print(doc)
```

In this example, we use sort() to sort the documents in the "mycollection" collection by the "age" field in ascending order and by the "name" field in descending order.

Q7. Explain why delete_one(), delete_many(), and drop() are used.

delete_one(): This method is used to delete a single document that matches the specified criteria from a collection. It is useful when you want to remove one specific document without affecting other documents.

delete_many(): This method is used to delete multiple documents that match the specified criteria from a collection. It allows you to delete