**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 NoSQL, document-oriented database that stores data in JSON-like BSON (Binary JSON) format. It is designed to handle unstructured or semi-structured data, offering flexibility and scalability compared to traditional relational databases.

**Non-relational databases** are databases that do not follow the rigid table-based schema of relational databases. They store data in formats like key-value pairs, documents, wide-column stores, or graphs. Examples include MongoDB, Cassandra, and Redis.

**When to prefer MongoDB over SQL databases:**

1. **Dynamic Schema**: When you need flexibility in schema design, as MongoDB supports schema-less data storage.
2. **Big Data**: For applications requiring horizontal scalability and handling large amounts of unstructured data.
3. **Fast Iterations**: When rapid development cycles demand a database that adapts quickly to changes.
4. **Complex Data Types**: For hierarchical or nested data structures.
5. **Distributed Systems**: MongoDB is built for distributed systems, ensuring high availability and performance.

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

1. **Schema Flexibility**: MongoDB uses a flexible, schema-less structure, allowing varied fields and data types.
2. **Document-Oriented**: Stores data in BSON format, resembling JSON, making it intuitive for developers.
3. **Scalability**: Supports horizontal scaling through sharding.
4. **Indexing**: Enables faster query execution by indexing fields.
5. **Aggregation Framework**: Offers powerful aggregation capabilities for data transformation and analysis.
6. **High Availability**: Provides replication to ensure data redundancy and fault tolerance.
7. **Geospatial Indexing**: Supports location-based queries.
8. **Ad Hoc Queries**: Allows complex, on-the-fly queries without needing predefined schemas.

**Q3. Code to Connect MongoDB to Python and Create a Database and Collection**

from pymongo import MongoClient

# Connect to MongoDB server

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

# Create a database

db = client["my\_database"]

# Create a collection

collection = db["my\_collection"]

print("Database and Collection created successfully.")

**Q4. Insert Records and Retrieve Them Using find() and find\_one()**

# Insert one record

record = {"name": "Alice", "age": 25, "city": "New York"}

collection.insert\_one(record)

# Insert multiple records

records = [

{"name": "Bob", "age": 30, "city": "Chicago"},

{"name": "Charlie", "age": 35, "city": "San Francisco"}

]

collection.insert\_many(records)

# Find one record

print("Single Record:", collection.find\_one())

# Find all records

print("All Records:")

for rec in collection.find():

print(rec)

**Q5. Using find() to Query MongoDB**

The find() method retrieves documents matching the query criteria. If no criteria are specified, it fetches all documents.

# Query for records where age is greater than 25

query = {"age": {"$gt": 25}}

results = collection.find(query)

# Print the results

for rec in results:

print(rec)

**Q6. Explanation and Example of the sort() Method**

The sort() method orders the query results based on a specified field. By default, it sorts in ascending order. Use -1 for descending order.

# Sort records by age in ascending order

results = collection.find().sort("age", 1)

print("Sorted Results:")

for rec in results:

print(rec)

**Q7. Explanation of delete\_one(), delete\_many(), and drop()**

1. **delete\_one()**: Removes the first document that matches the query.
2. collection.delete\_one({"name": "Alice"})
3. **delete\_many()**: Deletes all documents matching the query.
4. collection.delete\_many({"age": {"$gt": 30}})
5. **drop()**: Deletes the entire collection, including all documents and metadata.
6. collection.drop()

These methods are used for removing data selectively or entirely, depending on the requirements.