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 database management system. It falls under the category of non-relational or NoSQL databases. Non-relational databases differ from traditional SQL databases by providing flexible data models and horizontal scalability.

Non-relational databases like MongoDB organize and store data in a way that is different from tables, rows, and columns typically found in SQL databases. Instead, they use various data models, such as key-value pairs, documents, graphs, or wide-column stores. MongoDB specifically uses a document model, where data is stored in flexible JSON-like documents called BSON (Binary JSON).

In MongoDB, data is structured in collections, which are similar to tables in SQL databases. Each document within a collection can have a different structure, allowing for dynamic schemas. This flexibility makes it easier to handle evolving data structures and accommodate changes without altering the entire database schema.

MongoDB is preferred over SQL databases in certain scenarios, such as:

1. Flexibility and scalability: MongoDB can handle unstructured and rapidly changing data, making it suitable for agile development and fast-paced environments. It allows you to add or modify fields on the fly without affecting the entire database.
2. Big data and high write loads: MongoDB is designed to handle high volumes of data and write-intensive workloads. It scales horizontally by distributing data across multiple servers, allowing for increased storage and performance.
3. Rapid prototyping and agile development: MongoDB's flexible schema and document model make it a good choice for prototyping and iterative development. It enables developers to work quickly and adapt their data models as requirements evolve.
4. Real-time analytics and event-driven applications: MongoDB's support for real-time data processing and streaming makes it suitable for applications that require fast data ingestion, analysis, and event-driven architectures.
5. Use cases with complex relationships: While MongoDB is a document-oriented database, it also supports relationships between documents through references or embedding. This allows for handling complex relationships without the need for traditional joins found in SQL databases.

It's important to note that the choice between MongoDB and SQL databases depends on specific project requirements, data structures, and the need for transactions, ACID compliance, or complex querying capabilities. Each type of database has its strengths and should be chosen based on the particular use case.

Q2. State and Explain the features of MongoDB.

MongoDB is a popular NoSQL database management system that offers several features that make it flexible and scalable for handling large amounts of unstructured data. Here are some of the key features of MongoDB:

1. Document-oriented: MongoDB is a document-oriented database, which means it stores data in the form of flexible, JSON-like documents instead of using traditional tables and rows. This makes it easy to represent complex hierarchical relationships and store varying structures of data within a single collection.
2. Schemaless: Unlike relational databases, MongoDB does not enforce a predefined schema on the data. Each document within a collection can have its own unique structure, allowing for dynamic and agile development. This flexibility makes it well-suited for applications where the data schema evolves over time.
3. High scalability and performance: MongoDB is designed to scale horizontally across multiple servers, allowing it to handle large amounts of data and high traffic loads. It supports automatic sharding, which distributes data across different servers, ensuring efficient use of resources and enabling linear scalability.
4. High availability: MongoDB provides built-in support for replication, allowing you to create replicas of your data across multiple servers. This ensures high availability and fault tolerance by automatically promoting a replica to primary status if the primary node fails. It also supports automatic failover and data synchronization between replicas.
5. Flexible querying: MongoDB provides a powerful and expressive query language that supports a wide range of query operations. It supports rich queries with support for filtering, sorting, projection, and aggregation. It also provides a flexible indexing system that allows you to create indexes on any field or combination of fields to optimize query performance.
6. Geospatial capabilities: MongoDB has built-in support for geospatial indexing and querying, making it well-suited for applications that deal with location-based data. It provides efficient spatial indexing and a rich set of geospatial operators for performing operations like proximity searches, finding nearby locations, and calculating distances.
7. Ad hoc queries: MongoDB allows you to perform ad hoc queries on your data without the need for a predefined schema or data model. This makes it easy to explore and analyze the data during development and allows for iterative and agile development processes.
8. Flexible data model: MongoDB supports a wide variety of data types, including strings, numbers, dates, arrays, and even nested documents. It also allows you to create arrays and arrays of documents as fields within a document, enabling you to represent complex relationships and hierarchical structures.
9. Horizontal scaling and load balancing: MongoDB's sharding feature allows you to distribute data across multiple servers, enabling horizontal scaling. This ensures that your database can handle growing amounts of data and traffic by adding more servers to the cluster. MongoDB's built-in load balancing ensures that the data is evenly distributed across the cluster.

Overall, MongoDB's features make it well-suited for building modern, scalable, and flexible applications that require handling large volumes of unstructured or semi-structured data. It provides developers with the tools and capabilities to efficiently store, retrieve, and manipulate data in a highly scalable and performant manner.

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 library, which is the official MongoDB driver for Python. Here's an example code that demonstrates how to connect to MongoDB, create a database, and a collection:

from pymongo import MongoClient

# Connect to MongoDB

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

# Create a database

my\_database = client['my\_database']

# Create a collection

my\_collection = my\_database['my\_collection']

# Insert a document into the collection

document = {

'name': 'John Doe',

'age': 30,

'email': 'johndoe@example.com'

}

my\_collection.insert\_one(document)

# Close the connection

client.close()

In this example, we first import the MongoClient class from the pymongo module. Then, we create a client object and specify the connection URL, which includes the host and port where MongoDB is running. You may need to modify the connection URL according to your MongoDB setup.

Next, we create a my\_database object using client['my\_database'], which creates a database named "my\_database" if it doesn't exist. Similarly, we create a my\_collection object using my\_database['my\_collection'], which creates a collection named "my\_collection" if it doesn't exist.

After that, create a document (a Python dictionary) with some sample data and insert it into the my\_collection using the insert\_one() method.

Finally, we close the MongoDB connection using client.close().

Make sure you have the PyMongo library installed before running this code. You can install it using pip:

pip install pymongo

to replace the connection URL, database name, collection name, and document data according to your specific 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.

1. Import the necessary modules:

from pymongo import MongoClient

1. Establish a connection to the MongoDB server:

client = MongoClient('<connection\_string>')

Replace <connection\_string> with the appropriate connection string for your MongoDB server.

1. Access the desired database and collection:

db = client['your\_database']

collection = db['your\_collection']

Replace 'your\_database' and 'your\_collection' with the names of your database and collection, respectively.

1. Insert one record using the insert\_one() method:

record\_one = {'key': 'value'}

collection.insert\_one(record\_one)

Replace 'key' and 'value' with the appropriate field names and values for your record.

1. Insert many records using the insert\_many() method:

records\_many = [

{'key1': 'value1'},

{'key2': 'value2'},

{'key3': 'value3'}

]

collection.insert\_many(records\_many)

Replace 'key1', 'value1', 'key2', 'value2', 'key3', and 'value3' with the appropriate field names and values for your records.

1. Use the find\_one() method to print the inserted record:

result = collection.find\_one({'key': 'value'})

print(result)

Replace 'key' and 'value' with the appropriate field names and values for your inserted record.

1. Use the find() method to print all inserted records:

results = collection.find({})

for result in results:

print(result)

to adjust the field names and values according to your specific use case.

Here's the complete code:

from pymongo import MongoClient

# Establish connection to MongoDB server

client = MongoClient('<connection\_string>')

# Access the database and collection

db = client['your\_database']

collection = db['your\_collection']

# Insert one record

record\_one = {'key': 'value'}

collection.insert\_one(record\_one)

# Insert many records

records\_many = [

{'key1': 'value1'},

{'key2': 'value2'},

{'key3': 'value3'}

]

collection.insert\_many(records\_many)

# Print the inserted record using find\_one()

result = collection.find\_one({'key': 'value'})

print(result)

# Print all inserted records using find()

results = collection.find({})

for result in results:

print(result)

to replace <connection\_string>, 'your\_database', and 'your\_collection' with your specific values.

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

In MongoDB, the find() method is used to query the database and retrieve documents that match certain criteria. It allows you to specify the conditions that the documents must meet in order to be included in the result set. Here's how you can use the find() method in MongoDB:

1. Connect to the MongoDB database using a MongoDB driver for your preferred programming language.
2. Choose the collection you want to query. A collection is a grouping of MongoDB documents, similar to a table in a relational database.
3. Use the find() method on the collection object and pass a query document as a parameter to specify the filtering conditions.

The query document consists of field-value pairs. The fields represent the document fields you want to query, and the values represent the conditions those fields must meet. MongoDB uses a flexible query language that supports a wide range of operators to specify conditions. Some common operators include $eq for equality, $gt for greater than, $lt for less than, $in for matching any value in an array, and $regex for regular expression matching.

Here's a simple example code snippet in Python demonstrating the usage of the find() method:

from pymongo import MongoClient

# Connect to the MongoDB server

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

# Choose the database and collection

db = client['mydatabase']

collection = db['mycollection']

# Query the collection

query = { 'age': { '$gt': 25 } } # Find documents where 'age' is greater than 25

result = collection.find(query)

# Iterate over the result set and print the documents

for document in result:

print(document)

# Close the MongoDB connection

client.close()

In this example, connect to the local MongoDB server, choose the database named 'mydatabase', and the collection named 'mycollection'. We define a query document to find documents where the 'age' field is greater than 25. We pass this query document to the find() method, which returns a cursor object. We can then iterate over the cursor to access the matching documents and print them.

to modify the connection URL, database name, collection name, and query conditions according to your specific setup and requirements.

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

In MongoDB, the sort() method is used to sort the results of a query in ascending or descending order based on one or more fields. It allows you to specify the sorting criteria and the order in which you want the documents to be sorted.

The syntax of the sort() method in MongoDB is as follows:

db.collection.find().sort({field1: order1, field2: order2, ...})

‘db.collection.find() is the query that selects the documents to be sorted, and sort({field1: order1, field2: order2, ...}) specifies the sorting criteria and the order in which the documents should be sorted. The sorting criteria are defined as key-value pairs, where the key is the field name, and the value is either 1 for ascending order or -1 for descending order.

Now let's consider an example to demonstrate sorting in MongoDB. Suppose we have a collection called "books" with documents representing different books. Each book document has fields like "title", "author", and "publication\_year". We want to retrieve the books sorted by their publication years in descending order.

Here's an example query using the sort() method:

db.books.find().sort({publication\_year: -1})

This query selects all documents from the "books" collection and sorts them based on the "publication\_year" field in descending order (-1). As a result, the books will be returned in the order of their publication years from the newest to the oldest.

Note that you can sort by multiple fields by including additional key-value pairs in the sort() method. For example, if you want to sort by "publication\_year" in descending order and then by "author" in ascending order, you can modify the query as follows:

db.books.find().sort({publication\_year: -1, author: 1})

This query will sort the books first by their publication years in descending order and then by the authors' names in ascending order.

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

In the context of databases and collections in MongoDB, the methods delete\_one(), delete\_many(), and drop() are used to remove data from a collection or even delete an entire collection. Here's an explanation of each method:

1. delete\_one(): The delete\_one() method is used to delete a single document that matches a specified filter. It takes a filter as a parameter, which defines the criteria for selecting the document to be deleted. If multiple documents match the filter, only the first one encountered will be deleted. This method is useful when you want to remove a specific document from a collection.
2. delete\_many(): The delete\_many() method is used to delete multiple documents that match a specified filter. It works similarly to delete\_one() but deletes all the matching documents instead of just one. Like delete\_one(), it also takes a filter as a parameter to define the criteria for selecting the documents to be deleted. This method is useful when you need to remove multiple documents that meet certain conditions.
3. drop(): The drop() method is used to delete an entire collection from a database. It doesn't take any parameters but is called on a specific collection. When you invoke this method, it permanently removes the entire collection, including all the documents and indexes associated with it. This method is useful when you want to completely eliminate a collection and its data from the database.

It's important to exercise caution while using these methods, especially drop(), as they permanently remove data. Always double-check the filters and confirm the consequences before executing these operations, as the deleted data cannot be easily recovered.