assignment=13

July 27, 2023

1 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 that stores data in flexible, JSON-like documents. It is designed for high availability, scalability, and performance, and is commonly used for web, mobile, and IoT applications.

Non-relational databases, also known as NoSQL databases, are database management systems that do not use the traditional relational database model that SQL databases use. Instead, NoSQL databases use different data models, such as document-oriented, key-value, column-family, or graph.

NoSQL databases are preferred over SQL databases in scenarios where there is a need for high scalability, agility, and flexibility in handling large, unstructured or semi-structured data. This makes NoSQL databases a good choice for use cases such as web analytics, content management systems, real-time data processing, and IoT applications.

MongoDB is preferred over SQL databases in scenarios where there is a need for handling large volumes of semi-structured or unstructured data with high availability, performance, and scalability requirements. It is commonly used for use cases such as e-commerce, content management, mobile applications, and real-time analytics.

Some specific advantages of MongoDB over SQL databases include its ability to handle unstructured and semi-structured data, its flexible schema, its horizontal scalability, and its support for sharding and replication. Additionally, MongoDB's query language is more similar to modern programming languages, which makes it easier for developers to work with.

2 Q2. State and Explain the features of MongoDB.

MongoDB is a popular NoSQL database that offers a range of features that make it a popular choice for developers and organizations looking for a scalable and flexible database solution. Some of the key features of MongoDB include:

1. Document-oriented data model: MongoDB stores data in flexible, JSON-like documents that can vary in structure, making it a good choice for handling semi-structured or unstructured data.
2. High availability and scalability: MongoDB offers high availability and scalability through features such as automatic sharding, replication, and load balancing.

3. Rich query language: MongoDB provides a powerful query language that supports a wide range of queries, including ad hoc queries, range queries, and full-text search.
4. Indexing: MongoDB provides a range of indexing options, including single-field, compound, and multi-key indexes.
5. Aggregation: MongoDB offers a flexible aggregation framework that allows developers to perform advanced data processing and analysis, such as grouping, sorting, and filtering.
6. GridFS: MongoDB offers GridFS, a file storage system that allows developers to store and retrieve large files, such as images and videos, as part of the database.
7. Transactions: MongoDB supports multi-document transactions, allowing developers to perform complex operations that span multiple documents.
8. Security: MongoDB offers a range of security features, such as access control, authentication, and encryption, to ensure the security and integrity of the data stored in the database.
9. Flexibility: MongoDB's flexible data model and schemaless design make it a good choice for rapidly evolving applications and dynamic environments

Overall, MongoDB's rich feature set and flexibility make it a popular choice for a wide range of use cases, from mobile and web applications to IoT and real-time analytics.

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

[ ]:
```python
# Import the PyMongo library
import pymongo

# Connect to the MongoDB server
client = pymongo.MongoClient("mongodb://localhost:27017/")

# Create a new database
db = client["mydatabase"]

# Create a new collection in the database
collection = db["mycollection"]

# Insert a new document into the collection
document = {"name": "sonal", "age": 21, "city": "london"}
collection.insert_one(document)

# Find a document in the collection
query = {"name": "sonal"}
result = collection.find_one(query)
print(result)
```

4 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.

[ ]:
```python
# Import the PyMongo library
import pymongo

# Connect to the MongoDB server
client = pymongo.MongoClient("mongodb://localhost:27017/")

# Create a reference to the "mycollection" collection in the "mydatabase"␣
↪database
db = client["mydatabase"]
collection = db["mycollection"]

# Insert one record into the collection
record_one = {"name": "sonal", "age": 21, "city": "london"}
result_one = collection.insert_one(record_one)
print("Inserted record ID:", result_one.inserted_id)

# Insert many records into the collection
records_many = [
    {"name": "jadavbhai", "age": 47, "city": "New York"},
    {"name": "Charlie", "age": 35, "city": "San Francisco"},
    {"name": "asha", "age": 19, "city": "boston"}
]
result_many = collection.insert_many(records_many)
print("Inserted records IDs:", result_many.inserted_ids)

# Find one record in the collection
query = {"name": "sonal"}
result = collection.find_one(query)
print("Found record:", result)

# Find all records in the collection
results = collection.find()
print("All records:")
for result in results:
    print(result)
```

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

The find() method is used to query a MongoDB database for documents that match a specified set of criteria. This method returns a cursor object that can be used to iterate over the matching documents.

Here's an example code to demonstrate how to use the find() method to query a MongoDB database:

[ ]:
```
# Import the PyMongo library
import pymongo

# Connect to the MongoDB server
client = pymongo.MongoClient("mongodb://localhost:27017/")

# Create a reference to the "mycollection" collection in the "mydatabase"␣
 ↪database
db = client["mydatabase"]
collection = db["mycollection"]

# Find all documents in the collection
results = collection.find()

# Iterate over the results and print each document
for result in results:
    print(result)
```

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

The sort() method is used to sort the results of a MongoDB query in either ascending or descending order. This method takes one or more key-value pairs that specify the fields to sort by and the order in which to sort them.

Here's an example to demonstrate how to use the sort() method to sort the results of a MongoDB query:

[ ]:
```
# Import the PyMongo library
import pymongo

# Connect to the MongoDB server
client = pymongo.MongoClient("mongodb://localhost:27017/")

# Create a reference to the "mycollection" collection in the "mydatabase"␣
 ↪database
db = client["mydatabase"]
collection = db["mycollection"]
```

```python
# Insert some sample documents into the collection
collection.insert_many([
    {"name": "sonal", "age": 21},
    {"name": "jadavbhai", "age": 47},
    {"name": "Charlie", "age": 35}
])

# Sort the documents in the collection by age in ascending order
results = collection.find().sort("age", pymongo.ASCENDING)

# Print the sorted documents
for result in results:
    print(result)
```

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

In MongoDB, the delete_one() and delete_many() methods are used to remove documents from a collection that match a specified set of criteria. The drop() method is used to remove an entire collection from a database.

Here's a brief explanation of each method:

delete_one(filter): This method deletes the first document that matches the specified filter criteria. If there are multiple documents that match the criteria, only the first one is deleted. This method returns a DeleteResult object that contains information about the operation, such as the number of documents deleted.

delete_many(filter): This method deletes all documents that match the specified filter criteria. This method returns a DeleteResult object that contains information about the operation, such as the number of documents deleted.

drop(): This method removes an entire collection from the database. This method returns None.

These methods are used to remove documents from a MongoDB collection for various reasons, such as removing outdated or invalid data, cleaning up unused records, or removing duplicates. The drop() method is used when you want to completely remove a collection from the database and all of its associated documents.

It's important to note that when using any of these methods, you should exercise caution and double-check that you are deleting the correct documents or collections, as the deleted data cannot be recovered.

```
[ ]:
```