Practice Questions with Answer

Question 1: What is MongoDB?

Description: Explain what MongoDB is and how it differs from traditional relational databases.

Answer:

Explanation:
MongoDB is a NoSQL (Not Only SQL) database that stores data in flexible, JSON-like documents rather than in tables with fixed schemas. Unlike relational databases, which use structured data and require a predefined schema with rows and columns, MongoDB allows for schema flexibility, making it easier to store unstructured or semi-structured data.

Key differences:

- MongoDB stores data in **documents** (BSON format) rather than rows.
- It offers **schema flexibility**, allowing documents within a collection to have different structures.
- **Scalable and distributed** by design.

Question 2: How do you connect to a MongoDB database using the Mongo shell?

Description: Provide the command to connect to a MongoDB instance using the Mongo shell.

Answer:

To connect to a MongoDB instance using the Mongo shell, use the following command:

```
mongo
```

If you want to connect to a specific database, use:

```
mongo <database_name>
```

If MongoDB is running on a remote server or a specific port:

```
mongo --host <host_name> --port <port_number>
```

By default, MongoDB runs on `localhost` and port `27017`.

Question 3: What is the MongoDB command to insert a new document into a collection?

Description: Write the MongoDB shell command to insert a document into a collection named `users` with fields `name` and `age`.

Answer:

To insert a new document into the `users` collection, use the `insertOne()` method:

```
js

db.users.insertOne({ name: "John Doe", age: 30 });
```

This inserts a single document into the `users` collection with the specified fields.

Question 4: How do you retrieve all documents from a MongoDB collection?

Description: Write the MongoDB command to retrieve all documents from the `users` collection.

Answer:

To retrieve all documents from the `users` collection, use the `find()` method:

```
js

db.users.find();
```

This will return all documents within the `users` collection.

Question 5: What is the purpose of the `_id` field in MongoDB?

Description: Explain what the `_id` field is and its significance in MongoDB.

Answer:

Explanation:

- The `_id` field is a **special, unique identifier** for every document in a MongoDB collection.
- Every document in MongoDB automatically gets an `_id` field, and if one is not provided during insertion, MongoDB generates it automatically as an **ObjectId**.
- The `_id` field ensures that each document in a collection is **uniquely identifiable**.

Example of an auto-generated `_id`:

```
js
```

```
{ "_id": ObjectId("60c72b2f9af1d6e3d467b68a"), "name": "John Doe", "age":
30 }
```

The `_id` is indexed by default, allowing for fast querying and retrieval.

Practical Question

Question 1: MongoDB CRUD Operations

Description: Explain the difference between `insertOne()`, `insertMany()`, `find()`, and `deleteOne()` MongoDB methods. Provide an example for each method.

Answer:

- **`insertOne()`**: Inserts a single document into a collection.

  ```
  js

  db.users.insertOne({ name: "John", age: 30 });
  ```

- **`insertMany()`**: Inserts multiple documents into a collection.

  ```
  js

  db.users.insertMany([
    { name: "Jane", age: 25 },
    { name: "Tom", age: 35 }
  ]);
  ```

- **`find()`**: Retrieves documents from a collection. Returns a cursor.

  ```
  js

  db.users.find({ age: { $gt: 20 } });
  ```

- **`deleteOne()`**: Deletes the first document that matches a query.

  ```
  js

  db.users.deleteOne({ name: "John" });
  ```

Question 2: MongoDB Indexing

Description: What is an index in MongoDB, and why is it important? Write a command to create a descending index on the `age` field of the `users` collection.

Answer:

Explanation:
An index in MongoDB is a data structure that improves the speed of query operations. Without indexes, MongoDB has to scan every document in a collection, which can be inefficient for large datasets. Indexes are particularly useful for optimizing queries, such as sorting, filtering, and searching on specific fields.

Creating a descending index:

```
js

db.users.createIndex({ age: -1 });  // -1 for descending order
```

Question 3: Aggregation Framework

Description: Explain the MongoDB aggregation framework. Provide an example of an aggregation pipeline that groups documents by the `age` field and calculates the number of users in each age group.

Answer:

Explanation:
The aggregation framework in MongoDB allows you to process data and perform operations such as filtering, grouping, sorting, and projecting in a flexible and efficient manner. It uses a pipeline of stages, where each stage processes data in sequence.

Example Aggregation Pipeline:

```
js

db.users.aggregate([
  { $group: { _id: "$age", count: { $sum: 1 } } }  // Group by age and
count the number of users per age group
]);
```

This pipeline groups users by the `age` field and counts how many users exist in each group.

Question 4: Data Modeling

Description: Explain the difference between **embedding** and **referencing** in MongoDB data modeling. In which scenarios would you use each approach?

Answer:

Explanation:

- **Embedding**: This method stores related data within a single document. It's useful when the data is closely related and is frequently queried together. It reduces the number of read operations but can cause data duplication.

    - **Use case**: Storing comments for a blog post within the same document, as the comments are only relevant in the context of that post.
- **Referencing**: This method stores related data in separate documents and links them using references (e.g., ObjectIds). It is useful for maintaining relationships between collections, especially in cases of one-to-many or many-to-many relationships.
    - **Use case**: Storing user information in one collection and their orders in another, where multiple orders can be linked to a single user.

Question 5: MongoDB Schema Validation

Description: What is schema validation in MongoDB, and how can it be used to ensure data integrity? Write a MongoDB command to create a collection named `products` with a validation rule that ensures the `price` field is a positive number and the `name` field is required.

Answer:

Explanation: Schema validation in MongoDB ensures that documents inserted into a collection adhere to a specified structure. It helps maintain data integrity by enforcing constraints like data type, required fields, or ranges for field values.

Creating a collection with schema validation:

```
js

db.createCollection("products", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["name", "price"],
      properties: {
        name: {
          bsonType: "string",
          description: "Product name must be a string"
        },
        price: {
          bsonType: "double",
          minimum: 0,
          description: "Price must be a positive number"
        }
      }
    }
  }
});
```

This validation ensures that every `product` document has a `name` field (string) and a `price` field (double), and that the `price` is greater than or equal to 0.