MongoDB Assessment - 12

Section : I (MCQ) (Each Question Carries 1 Mark) Total : 10 Marks

1. MongoDB is written in .....

   A. C++

   B. Go

   C. JavaScript

   D. Python

   E. All

2. Does mongodb support linux?*

   A. True

   B. False

3. Which of the following is wrong statement –

   A. MongoDB supports search by field, range queries, regular expression searches

   B. MongoDB can store the business subject in the minimal number of documents

   C. Secondary indices is not available in MongoDB

   D. All of the above

4. In which format MongoDB represents document structure?

   A. BSON

   B. .txt

   C. .Docx

   D. None of these

5. A collection in MongoDB is a group of .....

   A. Databases

   B. Schema

   C. Related documents

   D. Rows

6. Which field is always the first field in the document.?

A. _id

B. Ob_id

C. id

D. None of these

7. The application, that communicates with application MongoDB by way of a client library, is called

A. Parent

B. Driver

C. Rank

D. None of these

8. Consider that the posts collection contains an array called ratings which contains ratings given to the post by various users in the following format:

```
{
        _id: 1,
        post_text: "This is my first post",
        ratings: [5, 4, 2, 5],
        //other elements of document
}
```

Which of the following query will return all the documents where the rating array contains elements that in some combination satisfy the query conditions?

A. db.inventory.find({ ratings: { $elemMatch: { $gt: 3, $lt: 6 } } })

B. db.inventory.find({ ratings: { ratings: { $gt: 5, $lt: 9 } } })

C. db.inventory.find({ ratings: { ratings.$: { $gt: 5, $lt: 9 } } })

D. db.inventory.find({ ratings: { $elemMatch: { $gte: 3, $lte: 6 } } })

9. Which of the following methods can be used on a cursor object?

A. cursor.next()

B. cursor.hasNext()

C. cursor.forEach()

D. All of the above

10. In our posts collection, which command can be used to find all the posts whose author names begin to lie between "A" and "C" in dictionary order?

A. db.posts.find({ post_author : { $gte : "A" , $lte : "C" } });

B. db.posts.find({ post_author : { $gte : "C" , $lte : "A" } });

C. db.posts.find({ post_author : { $gt : "A" , $lt : "C" } });

D. This type of search is not supported by MongoDB. $lt and $gt operators are applicable only for numeric values.

Section : II (Answer Any 2 Questions from [1-3] and Question 4 is Mandatory (Carries 5 Mark)
Total : 15 Marks

1. What are NoSQL databases? What are the various kinds of NoSQL databases? What are the advantages and disadvantages of normalizing data in a MongoDB database? What distinguishes MongoDB as the best? On a high level, compare SQL databases and MongoDB

NoSQL databases, like MongoDB, are non-relational and designed to store large amounts of data with high performance and flexible data models. Unlike traditional SQL databases that organize data in tables with fixed schemas, NoSQL databases allow flexible or dynamic schemas that can handle unstructured or rapidly changing data. They scale horizontally by spreading data across many servers.

Types of NoSQL Databases
1) The major types of NoSQL databases are:
2) Key-value stores: Each item is stored as a key-value pair (e.g., Redis, DynamoDB). Document-oriented databases: Store data as JSON-like documents, making them well-suited for semi-structured data and dynamic schemas (e.g., MongoDB, Couchbase).
3) Wide-column stores: Organize data into rows and flexible columns, optimized for analytical workloads and high scalability (e.g., Cassandra, HBase).
4) Graph databases: Focus on managing relationships among data using nodes and edges
5) Multi-model databases: Combine multiple types into a single system.

Advantages of normalizing data in MongoDB include:

1) Reduces duplicate data.

2) Improves accuracy and consistency.

3) Makes data updates easier.

4) Prevents problems like inconsistencies when inserting, updating, or deleting data.

Disadvantages of normalization in MongoDB:

1) Queries become complicated because related data is in multiple places, needing extra lookups.

2) Can slow down performance, especially for frequent read operations.

3) Designing normalized schemas is more complex.

4) Goes against MongoDB's document-based style, which prefers embedding related data in one document for speed.

What makes MongoDB special?

It is flexible and schema-less, so data fields and types can change without stopping the database.

Highly scalable, using sharding to spread data across servers to handle large loads.

Stores data as JSON-like documents, which fits modern apps well.

Has strong tools to perform data transformations in the database.

Open source with a large community and good tools.

Built-in data replication and failover for reliability.

Differentiation between SQL databases and MongoDB:

1) Data Model

SQL: Uses a relational model with data stored in tables of rows and columns.

MongoDB: Uses a document-oriented model storing data as flexible JSON-like (BSON) documents.

2) Schema

SQL: Requires a fixed, predefined schema before data insertion.

MongoDB: Offers a dynamic, schema-less design allowing schema changes without downtime.

3) Query Language
SQL: Uses Structured Query Language (SQL) for data definition and manipulation.
MongoDB: Uses MongoDB Query Language (MQL), which is JSON-like and object-oriented.

4) Relationships and Joins
SQL: Supports complex joins across multiple tables.
MongoDB: Does not support traditional joins; uses embedding and referencing within documents for relationships.

5) Transactions and ACID Compliance
SQL: Strong ACID compliance with multi-operation transactions.
MongoDB: Supports multi-document ACID transactions since version 4.0 but traditionally leaned towards atomic operations on single documents.

6) Scaling
SQL: Vertical scaling (scaling up by increasing resources on a single server).
MongoDB: Horizontal scaling via sharding, distributing data across multiple servers.

7) Performance & Use Cases
SQL: Excels in structured data, complex transactions, and applications requiring data integrity (e.g., banking systems).
MongoDB: Suited for unstructured or semi-structured data, big data, rapid prototyping, content management, and real-time analytics.

8) Flexibility
SQL: Less flexible due to rigid schema requirements.
MongoDB: Highly flexible to accommodate evolving data models.

9) Data Storage Format
SQL: Stores data in tabular form.
MongoDB: Stores data in BSON format (similar to JSON).

10) Programming Language Support
SQL: Works well with languages supporting SQL libraries.

MongoDB: Supports many programming languages natively due to its JSON-like document structure.

2. Explain Sharding and Replica Set. What are the similarities and differences between sharding and replication in MongoDB?

3. Create MongoDB Statements Illustrating

A. Projection
B. Limiting Records
C. Sorting Records
D. User Management Methods
E. Role Management Methods
F. MongoDB Replication Methods

A. Projection: Projection restricts which fields are returned in query results.

```
db.users.find({}, { name: 1, age: 1, _id: 0 })
```

This returns only the name and age fields, excluding the default _id field.

B. Limiting Records: Limits the number of documents returned by a query.

```
db.orders.find().limit(5)
```

C. Sorting Records: Sort results in ascending (1) or descending (-1) order.

```
db.products.find().sort({ price: 1 })
```

Find all products and sort by price ascending

```
db.users.find().sort({ age: -1 })
```

Sort by age descending

D. User Management Methods: Creating users with roles and authentication.

Create a user with readWrite role on "shopDB"

```
db.createUser({
  user: "shopUser",
  pwd: "strongPassword",
  roles: [ { role: "readWrite", db: "shopDB" } ]
})
```

Authenticate as that user:

```
db.auth("shopUser", "strongPassword")
```

E. Role Management Methods: Creating, updating, and assigning roles.

Create a custom role

```
db.createRole({
  role: "reportViewer",
  privileges: [
    { resource: { db: "reportsDB", collection: "" }, actions: ["find"] }
  ],
  roles: []
})
```

Grant role to a user

```
db.grantRolesToUser("shopUser", [{ role: "reportViewer", db: "reportsDB" }])
```

Revoke role from a user

```
db.revokeRolesFromUser("shopUser", [{ role: "reportViewer", db: "reportsDB" }])
```

F. MongoDB Replication Methods: Commands to configure and manage replica sets.

Initiate a replica set with name "rs0"

```
rs.initiate()
```

Check replica set status

```
rs.status()
```

Add a member to replica set

```
rs.add("mongodb2.example.net:27017")
```

Remove a member from replica set

```
rs.remove("mongodb3.example.net:27017")
```

4. Write a Java Application to communicate with MongoDB to perform CRUD Operations on Offence
Collection.

A. Connect to Database

B. Create a Collection

C. Getting/Selecting a Collection

D. Insert a Document

E. Retrieve All Documents

F. Update Document

G. Delete a Document

H. Dropping a Collection

I. Listing All the Collections*

```
import com.mongodb.client.*;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Updates;
import org.bson.Document;
import java.util.Arrays;

public class MongoDBOffenceCRUD {

    // A. Connect to Database
    public static MongoClient connectToMongo() {
        // Connects to MongoDB running on localhost default port 27017
        return MongoClients.create("mongodb://localhost:27017");
    }

    public static void main(String[] args) {
        MongoClient mongoClient = connectToMongo();
        String dbName = "testdb";
        String collectionName = "Offence";

        // B. Create a Collection (MongoDB creates collection automatically on first insert; explicit
create shown here)
        MongoDatabase database = mongoClient.getDatabase(dbName);
        // Drop if exists for fresh start
        for (String collName : database.listCollectionNames()) {
            if (collName.equals(collectionName)) {
                database.getCollection(collectionName).drop();
                System.out.println("Existing collection dropped.");
            }
        }
        database.createCollection(collectionName);
        System.out.println("Collection created: " + collectionName);
```

```
        // C. Getting/Selecting a Collection
        MongoCollection<Document> collection = database.getCollection(collectionName);

        // D. Insert a Document
        Document offence1 = new Document("_id", 1)
                .append("type", "Theft")
                .append("description", "Stealing property")
                .append("severity", "Medium");
        collection.insertOne(offence1);
        System.out.println("Document inserted: " + offence1.toJson());

        // Insert multiple documents
        Document offence2 = new Document("_id", 2)
                .append("type", "Assault")
                .append("description", "Physical attack")
                .append("severity", "High");
        Document offence3 = new Document("_id", 3)
                .append("type", "Fraud")
                .append("description", "Deception to gain money")
                .append("severity", "High");
        collection.insertMany(Arrays.asList(offence2, offence3));
        System.out.println("Multiple documents inserted.");

        // E. Retrieve All Documents
        System.out.println("All documents in Offence collection:");
        for (Document doc : collection.find()) {
            System.out.println(doc.toJson());
        }

        // F. Update Document
        // Update description of offence type Theft
        collection.updateOne(Filters.eq("type", "Theft"), Updates.set("description", "Stealing
personal property"));
        System.out.println("Updated document with type 'Theft':");
        System.out.println(collection.find(Filters.eq("type", "Theft")).first().toJson());

        // G. Delete a Document
        // Delete offence with type 'Fraud'
        collection.deleteOne(Filters.eq("type", "Fraud"));
        System.out.println("Document with type 'Fraud' deleted.");

        // Check remaining documents
        System.out.println("Documents after deletion:");
```

```
        for (Document doc : collection.find()) {
            System.out.println(doc.toJson());
        }

        // H. Dropping a Collection
        collection.drop();
        System.out.println("Collection '" + collectionName + "' dropped.");

        // I. Listing All the Collections
        System.out.println("Listing all collections in database:");
        for (String collName : database.listCollectionNames()) {
            System.out.println(collName);
        }

        // Close client connection
        mongoClient.close();
    }
}
```