**EXP-NO-2**

**A. Distributed Database Design and Implementation**

**Aim:**

To configure MongoDB's replica sets and sharding, and perform CRUD operations to test data distribution, high availability, and fault tolerance in a distributed database environment.

Procedure:

```javascript

const { MongoClient } = require('mongodb');

async function run() {

const uri = "mongodb://localhost:27025";

const client = new MongoClient(uri);

try {

await client.connect();

console.log("Connected to MongoDB via mongos router on port 27025");

const adminDb = client.db("admin");

**Step 1: Enable Sharding on Database**

console.log("Enabling sharding on 'myDatabase'...");

await adminDb.command({ enableSharding: "myDatabase" });

console.log("Sharding enabled on 'myDatabase'.");

**Step 2: Shard the Collection**

console.log("Sharding collection 'myDatabase.myCollection' on 'userId'...");

await adminDb.command({

shardCollection: "myDatabase.myCollection",

key: { userId: 1 }

});

console.log("Sharding enabled for 'myDatabase.myCollection'.");

**Step 3: Insert Data**

const db = client.db("myDatabase");

const collection = db.collection("myCollection");

console.log("Inserting documents...");

await collection.insertMany([

{ userId: 1, name: "Alice", age: 25, location: "Region1" },

{ userId: 2, name: "Bob", age: 30, location: "Region2" },

{ userId: 3, name: "Charlie", age: 28, location: "Region3" }

]);

console.log("Documents inserted.");

**Step 4: Query Data**

console.log("Querying all documents...");

const docs = await collection.find().toArray();

console.log("Documents in 'myCollection':", docs);

**Step 5: Update Data**

console.log("Updating document with userId 1...");

await collection.updateOne(

{ userId: 1 },

{ $set: { age: 26 } }

);

console.log("Document updated.");

**Step 6: Delete Data**

console.log("Deleting document with userId 2...");

await collection.deleteOne({ userId: 2 });

console.log("Document deleted.");

**Step 7: Check Sharding Status**

console.log("Sharding status can be checked in the MongoDB shell using `sh.status()`.");

} finally {

await client.close();

}

}

run().catch(console.dir);

```

**Output:**

Successful connection to MongoDB via the mongos router.

Sharding enabled on the database and collection.

Documents inserted successfully:

```json

[

{ "userId": 1, "name": "Alice", "age": 25, "location": "Region1" },

{ "userId": 2, "name": "Bob", "age": 30, "location": "Region2" },

{ "userId": 3, "name": "Charlie", "age": 28, "location": "Region3" }

]

```

Document with `userId: 1` updated successfully.

Document with `userId: 2` deleted successfully.

**Result:**

The configuration of MongoDB's replica sets and sharding has been successfully completed. CRUD operations were performed to test data distribution, high availability, and fault tolerance within a distributed database environment.

**B. Using a Relational Database to Store XML Documents as Text**

To create a relational database for storing XML documents as text, you need to follow these general steps. This process involves defining a table in the database to hold the XML documents and then inserting or updating records with the XML content. Below is an example using SQL, assuming you are utilizing a database system that supports SQL (e.g., MySQL, PostgreSQL, SQLite).

**Steps to Store XML Documents:**

**1. Choose a Database System:**

Select the relational database management system (RDBMS) you wish to use. Common options include MySQL, PostgreSQL, SQLite, or Microsoft SQL Server.

**2. Create a Database:**

Use SQL to create a new database if one does not already exist.

```sql

CREATE DATABASE YourDatabaseName;

```

**3. Use the Database:**

Switch to the newly created database.

```sql

USE YourDatabaseName;

```

**4. Create a Table for XML Documents:**

Define a table that includes a column for storing the XML documents as text.

```sql

CREATE TABLE XmlDocuments (

document\_id INT AUTO\_INCREMENT PRIMARY KEY,

xml\_content TEXT

);

```

**5. Insert XML Documents into the Table:**

Use SQL `INSERT` statements to add XML documents to the table.

```sql

INSERT INTO XmlDocuments (xml\_content) VALUES

('<root><person><name>John Doe</name><age>30</age></person></root>'),

('<root><person><name>Jane Doe</name><age>25</age></person></root>');

```

**6. Query and Retrieve XML Documents:**

Use SQL `SELECT` statements to query and retrieve XML documents from the table.

```sql

SELECT \* FROM XmlDocuments;

```

**7. Update XML Documents:**

If necessary, use SQL `UPDATE` statements to modify existing XML documents in the table.

```sql

UPDATE XmlDocuments SET xml\_content =

'<root><person><name>NewName</name><age>40</age></person></root>'

WHERE document\_id = 1;

```

**8. Delete XML Documents:**

If needed, use SQL `DELETE` statements to remove XML documents from the table.

```sql

DELETE FROM XmlDocuments WHERE document\_id = 2;