NoSQL

Problem Statement 1: Schema-less Social Network

Design and build a simple social network schema using a Document Store database (e.g., MongoDB). The schema should accommodate user profiles with fields like name, location, interests, and an array of posts (including content and timestamp).

Task:

1. Create a NoSQL database and collection to store user profiles.
2. Define a document structure for user profiles, incorporating schema-less flexibility.
3. Populate the collection with sample user data (minimum 10 users).
4. Demonstrate how to insert, update, and delete user profiles using CRUD operations.
5. Query the database to find users based on specific interests or locations.

Sample Dataset:

JSON

```
{
 "name": "John Doe",
 "location": "Bangalore, India",
 "interests": ["Technology", "Music"],
 "posts": [
            {
            "content": "Learning NoSQL is exciting!",
            "timestamp": "2024-06-05"
            }
     ]
},
{
 "name": "Jane Smith",
 "location": "New York, USA",
 "interests": ["Travel", "Art"],
 "posts": []
}
```

Problem Statement 1: Schema-less Social Network (MongoDB)

1. Create a NoSQL database and collection

```
use social_network;
db.createCollection("users");
```

2. Define a flexible user profile structure

```
{
  "name": "John Doe",
  "location": "Bangalore, India",
  "interests": ["Technology", "Music"],
  "posts": [
    {
      "content": "Learning NoSQL is exciting!",
      "timestamp": "2024-06-05"
    }
  ]
}
```

3. Insert Sample Users (10 users)

```
db.users.insertMany([
  {
    "name": "John Doe",
    "location": "Bangalore, India",
    "interests": ["Technology", "Music"],
    "posts": [
      { "content": "Learning NoSQL is exciting!", "timestamp": "2024-06-05" }
    ]
  },
  {
    "name": "Jane Smith",
    "location": "New York, USA",
    "interests": ["Travel", "Art"],
    "posts": []
  }
  // Add 8 more users...
]);
```

4. CRUD Operations

- Insert a New User
- db.users.insertOne({ "name": "Alice Brown", "location": "London, UK", "interests": ["Cooking", "Photography"], "posts": [] });
- Update User Interests
- db.users.updateOne({ "name": "John Doe" }, { $push: { "interests": "AI" } });
- Delete a User
- db.users.deleteOne({ "name": "Jane Smith" });

5. Querying Users

- Find users based on interest
- db.users.find({ "interests": "Technology" });
- Find users based on location
- db.users.find({ "location": "Bangalore, India" });

Problem Statement 2: Movie Recommendation Engine

Develop a prototype recommendation engine using a Key-Value Store database (e.g., Redis).

The system will recommend movies based on user ratings and genre preferences.

Task:

1. Create a Key-Value Store and define keys to store user IDs, movie IDs, genres, and

user ratings for movies.

2. Implement functions to add user ratings and movie information.

3. Design a recommendation algorithm that considers user ratings and genre

preferences when suggesting movies.

4. Simulate user interactions by adding ratings and retrieving recommendations.

Sample Dataset:

1. Key: user_1 (Value: {"name": "John", "ratings": {"movie_1": 4,"movie_2": 5}})
2. Key: movie_1 (Value: {"title": "The Matrix", "genre": "SciFi"})
3. Key: movie_2 (Value: {"title": "The Godfather", "genre":"Drama"})

Problem Statement 2: Movie Recommendation Engine (Redis)

1. Create a Key-Value Store

```
redis-cli
SET user_1 '{"name": "John", "ratings": {"movie_1": 4, "movie_2": 5}}'
SET movie_1 '{"title": "The Matrix", "genre": "SciFi"}'
SET movie_2 '{"title": "The Godfather", "genre": "Drama"}'
```

2. Add User Ratings and Movie Data

```
HSET user_2 name "Alice"
HSET user_2 ratings '{"movie_1": 3, "movie_3": 4}'
HSET movie_3 title "Inception"
HSET movie_3 genre "SciFi"
```

3. Recommend Movies Based on Ratings and Genre

- Fetch user preferences
- GET user_1
- Find movies in the same genre
- KEYS movie_*
- Recommend based on rating & genre
- HGET movie_1 title