Exp 6: Implement NoSQL Database Operations: CRUD operations, Arrays using MongoDB & Cassandra.

Implementing CRUD operations

MongoDB supports four primary CRUD (Create, Read, Update, Delete) operations for interacting with data:

1. Create: To create a document in MongoDB, we insert it into a collection using theáinsertOne()áoráinsertMany()ámethod. Documents can contain any valid JSON data, allowing for flexible schema design.
2. Read: To read data from MongoDB, we query witháfind()ámethod. The queries filter documents based on the specified criteria and return the results.
3. Update: To update the documents, eitheráupdateOne()áoráupdateMany()ácan be used. This method facilitates flexible data manipulation.
4. Delete: To delete documents from a collection, we use theádeleteOne()áorádeleteMany()ámethod. Deletions are performed based on specified criteria, such as matching a particular field value.

NoSQL CRUD operations (Create, Read, Update, Delete) and array handling in both MongoDB and Cassandra.

mongosh

```
test> use mydb
switched to db mydb
```

1. Create (Insert)

MongoDB has commands to insert either a single document or multiple documents at once:

Insert a Single Document

```
db.collectionName.insertOne({
  name: "John Doe",
  age: 30,
  interests: ["coding", "reading"]
});
```

```
mydb> show dbs
admin    40.00 KiB
config  108.00 KiB
local    96.00 KiB
mydb     72.00 Ki
```

Insert Multiple Documents

```
db.collectionName.insertMany([
  { name: "Jane Doe", age: 25, interests: ["music", "traveling"] },
  { name: "Jim Beam", age: 35, interests: ["gaming", "hiking"] }]);
```

2. Read (Query)

MongoDB allows you to query for specific data, apply filters, and use projections to limit which fields are returned.

Retrieve All Documents

```
db.collectionName.find({});
```

Retrieve Specific Documents with a Filter

```
db.collectionName.find({ age: { $gt: 30 } }); // Documents where age > 30
```

Apply Projection (Limit Fields Returned)

```
db.collectionName.find({}, { name: 1, interests: 1 }); // Only include 'name' and 'interests'
```

3. Update

Updating documents in MongoDB allows you to modify existing data, add fields, or work with array elements. You can update one document at a time or multiple documents that match a filter.

Update a Single Document

```
db.collectionName.updateOne(
  { name: "John Doe" },                    // Filter
{ $set: { age: 31 }, $push: { interests: "sports" } });  // Update with new values
```

Update Multiple Documents

```
db.collectionName.updateMany(
{ age: { $lt: 30 } },
  { $set: { status: "young" } } ); // Set a new field or update existing fields
```

4. Delete

MongoDB supports deleting either a single document or multiple documents that meet a specific filter condition.

Delete a Single Document

```
db.collectionName.deleteOne({ name: "Jim Beam" });
```

Delete Multiple Documents

```
db.collectionName.deleteMany({ age: { $lt: 25 } });
```

5. Working with Arrays in MongoDB

Arrays in MongoDB are treated as first-class citizens, and MongoDB provides various operators for managing array elements.

Adding Elements to Arrays

Use the $push operator to add an element to an array field.

```
db.collectionName.updateOne(
  { name: "John Doe" },
  { $push: { interests: "movies" } }
);
```

You can also use $each to add multiple items to an array:

```
db.collectionName.updateOne(
  { name: "John Doe" },
  { $push: { interests: { $each: ["swimming", "traveling"] } } }
);
```

Removing Elements from Arrays

Use the $pull operator to remove an element from an array:

```
db.collectionName.updateOne(
  { name: "John Doe" },
  { $pull: { interests: "reading" } }
);
```

Checking for Elements in an Array

To find documents where an array contains a specific element, use the array field directly in the query.

```
db.collectionName.find({ interests: "coding" });
```

To check if an array contains all specified values, use $all:

```
db.collectionName.find({ interests: { $all: ["coding", "sports"] } });
```

Updating Specific Array Elements

Use the positional $ operator to update the first matching array element based on a condition.

```
db.collectionName.updateOne(
{ name: "John Doe", "interests": "coding" },
  { $set: { "interests.$": "programming" } });
```

Using $addToSet to Avoid Duplicates in an Array

To add an element to an array only if it does not already exist, use $addToSet.

```
db.collectionName.updateOne( { name: "John Doe" }, { $addToSet: { interests: "coding" } });  // Will not add "coding" if it already exists
```