**1. Create and Drop Database**

**1.1 Create Database**

MongoDB creates a database when you first **use** it and insert data into it. If the database doesn't exist, MongoDB will create it once data is inserted.

* ***use myDatabase* // Switch to or create 'myDatabase'**

**MongoDB will only create the myDatabase once you insert a document into a collection in this database.**

**1.2 Drop Database**

To delete a database, use the dropDatabase() command:

* ***db.dropDatabase();***

**This will drop the current database you are connected to.**

**2. Create and Drop Collections**

**2.1 Create Collection**

MongoDB will automatically create a collection when you insert a document into it, but you can also explicitly create a collection with the createCollection() command.

* ***db.createCollection("myCollection");***

**This creates a new collection named myCollection.**

**2.2 Drop Collection**

To delete a collection, use the drop() method on that collection.

* ***db.myCollection.drop();***

**This will drop the myCollection from the database.**

**3. CRUD Operations on Documents**

**3.1 Create (Insert Document)**

To insert a new document, use insertOne() for one document or insertMany() for multiple documents.

* **Insert One Document**:

***db.myCollection.insertOne({***

***"name": "John",***

***"age": 30,***

***"city": "New York" });***

* **Insert Multiple Documents**:

***db.myCollection.insertMany([***

***{ "name": "Jane", "age": 28, "city": "London" },***

***{ "name": "Mark", "age": 35, "city": "San Francisco" }***

***]);***

**3.2 Read (Find Document)**

Use the find() method to retrieve documents from a collection.

* **Find All Documents**:

***db.myCollection.find().pretty();* // pretty() makes the output more readable**

* **Find Documents with a Query**:

***db.myCollection.find({ "city": "New York" }).pretty();***

**3.3 Update (Modify Document)**

To modify documents, you can use the updateOne(), updateMany(), or replaceOne() commands.

* **Update One Document**:

***db.myCollection.updateOne(***

***{ "name": "John" }, // Query to find the document***

***{ "$set": { "age": 31 } } // Update action***

***);***

* **Update Multiple Documents**:

***db.myCollection.updateMany(***

***{ "city": "New York" }, // Query to find matching documents***

***{ "$set": { "city": "NYC" } } // Update action***

***);***

**3.4 Delete (Remove Document)**

To delete documents, you can use deleteOne() or deleteMany().

* **Delete One Document**:

***db.myCollection.deleteOne({ "name": "John" });***

* **Delete Multiple Documents**:

***db.myCollection.deleteMany({ "city": "NYC" });***

**Cursor**

In MongoDB, a **cursor** is an object that allows you to iterate over the results of a query. When you run a query, MongoDB returns a cursor, not the actual result set all at once. This cursor points to the documents in the result set and allows you to retrieve them one by one or in batches.

**Key Features of a Cursor:**

1. **Lazy Evaluation**:
   * MongoDB doesn’t retrieve all documents at once. Instead, it fetches documents as you request them, which is efficient, especially when dealing with large datasets.
2. **Batch Processing**:
   * By default, MongoDB retrieves documents in batches. The first batch returns up to 101 documents, and subsequent batches retrieve more based on the batch size (typically 4 MB).
3. **Iterating Over Documents**:
   * A cursor allows you to traverse through the result set one document at a time using methods like .next(), .forEach(), or .toArray().

**Example of Using a Cursor:**

Let's say you have a collection named Student, and you want to retrieve all documents from it:

***var cursor = db.Student.find();***

Here, cursor contains a pointer to the result set, and you can now iterate through it:

***while (cursor.hasNext()) {***

***printon(cursor.next());***

***}***

* **hasNext()**: Checks if there are more documents in the result set.
* **next()**: Retrieves the next document in the cursor.

**Cursor Lifespan:**

* **Timeout: By default, a cursor has a time limit (10 minutes) after which it automatically expires if you don't fully iterate through it. However, you can set it to be non-timeout if needed.**
* **Exhausting the Cursor: Once all documents have been retrieved, the cursor is considered "exhausted" and cannot be reused.**

**In Summary:**

A cursor is a **pointer to the result set** that allows you to **iterate** over documents in MongoDB efficiently, especially when dealing with large datasets. It helps manage memory usage by fetching data in **batches** instead of loading everything into memory at once.

**Limit and Sort Records in MongoDB**

In MongoDB, you can use **limit** and **sort** to control how many documents you want to retrieve and the order in which they are returned.

**1. Limit Records in MongoDB**

The **limit()** method restricts the number of documents returned by a query. This is useful when you only need a subset of the results (like the first 10 records).

**Example:**

To get the first 5 students from the Student collection:

***db.Student.find().limit(5);***

* This will return **only 5 documents** from the Student collection.

**2. Sort Records in MongoDB**

The **sort()** method arranges the documents in either ascending or descending order based on the specified field(s). Sorting can be done on any field in the document.

**Syntax:**

***db.collection.find().sort({ field: 1 });***

* **1** means ascending order.
* **-1** means descending order.

**Example:**

**To sort students by age (sage) in ascending order:**

db.Student.find().sort({ sage: 1 });

To sort students by age (sage) in descending order:

***db.Student.find().sort({ sage: -1 });***

**3. Combining Limit and Sort**

**You can combine limit() and sort() to get a specific number of sorted documents.**

**Example:**

To get the top 3 youngest students (sorted by age in ascending order):

***db.Student.find().sort({ sage: 1 }).limit(3);***

* **sort({ sage: 1 })**: Sorts the students by age in ascending order (youngest to oldest).
* **limit(3)**: Limits the result to the first 3 students.

**Example 2: Get the 5 oldest students**

***db.Student.find().sort({ sage: -1 }).limit(5);***

* **sort({ sage: -1 })**: Sorts by age in descending order (oldest to youngest).
* **limit(5)**: Returns the top 5 oldest students.

**Example with Multiple Fields:**

You can also sort by multiple fields. For instance, you can first sort by age (sage) and then by name (sname):

***db.Student.find().sort({ sage: 1, sname: 1 }).limit(5);***

* This sorts students by age (ascending), and for students of the same age, it will sort by name (ascending).

**Summary:**

* **limit()**: Restricts the number of documents returned.
* **sort()**: Orders the documents by a specific field.
* You can **combine limit() and sort()** to get a limited set of sorted results.

db.Student.insertMany([

{ "sid": 1, "sname": "Alice", "sage": 22, "sinfo": "Physics major" },

{ "sid": 2, "sname": "Bob", "sage": 20, "sinfo": "Computer Science major" },

{ "sid": 3, "sname": "Charlie", "sage": 23, "sinfo": "Mathematics major" },

{ "sid": 4, "sname": "David", "sage": 21, "sinfo": "Biology major" },

{ "sid": 5, "sname": "Eve", "sage": 22, "sinfo": "Chemistry major" },

{ "sid": 6, "sname": "Frank", "sage": 24, "sinfo": "History major" },

{ "sid": 7, "sname": "Grace", "sage": 20, "sinfo": "Physics major" },

{ "sid": 8, "sname": "Hank", "sage": 23, "sinfo": "English major" },

{ "sid": 9, "sname": "Ivy", "sage": 19, "sinfo": "Mathematics major" },

{ "sid": 10, "sname": "Jack", "sage": 21, "sinfo": "Computer Science major" }

]);