Lab-11 : Update, Delete and Rename

General Syntax: updateOne Or updateMany

```
db.<collection_name>.updateOne(
 { <condition> },
 { <update_operator> }
)
```

Update Operators:

1. **$set** : Update or add a field - **$set: { field: value }**

   Ex. Update the age of John's to 31.

   ```
   db.Student.updateOne({ name: "John" }, {$set: { age: 31 } })
   ```

   Ex. Update all documents to add a new field country with the value 'USA'.

   ```
   db.Student.updateMany({}, { $set: { country: "USA" } })
   ```

2. **$inc** : Increment numeric values - **$inc: { field: number }**

   Ex. Increment the age of all students by 1 year.

   ```
   db.Student.updateMany({}, { $inc: { age: 1 } })
   ```

3. **$mul** : Multiply numeric values - **$mul: { field: number }**

   Ex. Multiply the age of all students by 2.

   ```
   db.Student.updateMany({}, { $mul: { years: 2 } })
   ```

4. **$unset** : Remove a field - **$unset: { field: "" }**

Ex. Unset (remove) the city field for 'Tom'.

db.Student.updateOne({ name: "Tom" }, { $unset: { city: "" } })

5. **$rename** : Rename a field - $rename: { oldField: "newField" }

   Ex. Rename the age field to years for all documents.

   db.Student.updateMany({}, { $rename: { age: "years" } })

   Ex. Rename the name field to FullName for 'John'.

   db.Student.updateOne({ name: "John" }, { $rename: { name: "FullName" } })

Delete Methods - deleteOne() & deleteOne() :

General Syntax:

db.<collection_name>.deleteOne({ <condition> })

db.<collection_name>.deleteMany({ <condition> })

Ex. Delete a document of 'Nick' from the collection.

db.Student.deleteOne({ name: "Nick" })

Ex. Delete all the students aged above 35.

db.Student.deleteMany({ age: { $gt: 35 } })

Ex. Delete all students who have city field missing.

db.Student.deleteMany({ city: { $exists: false } })

upsertedCount in Update Operation:

Upsert = Update + Insert

upsertedCount shows:

How many documents were inserted instead of updated.

Or..

It indicates the number of documents inserted during an upsert operation.

Ex. Update with upsert:

```
db.Student.updateOne(
 { rollNo: 101 },
 { $set: { name: "Rahul" } },
 { upsert: true }
)
```

Here if rollNo: 101 exists -> **update**

And if rollNo: 101 does NOT exist → **insert new document & upsertedCount: 1**

NOTE :

- **If upsert option is not used, upsertedCount will always be 0.**
- **upsertedCount is greater than 0 only when upsert: true is used and no matching document exists.**

Capped Collection:

A capped collection is a fixed-size collection that stores documents in insertion order.

When the **maximum size or maximum number of documents** is reached:

- MongoDB automatically removes the oldest documents.
- New documents are inserted at the end.

This works like a **circular queue (FIFO – First In, First Out)**

Real-Life Example:

A capped collection in MongoDB works like WhatsApp auto-disappearing messages, where old messages are automatically removed when new messages arrive.

Syntax to Create a Capped Collection:

```
db.createCollection("<collection_name>", {
  capped: true,
  size: <size_in_bytes>,
  max: <maximum_documents>
})
```

Ex. Create a Capped Collection named "Employee" which allows maximum 3 documents:

```
db.createCollection("Employee", {
  capped: true,
  size: 500,
  max: 3
})
```

Task:

Advanced Capped Collection with Validation:

Que. Create a capped collection named Employee with the following conditions:

- The collection should be capped with a fixed size.
- It should allow a maximum of 5 documents.
- The collection must enforce schema validation using **$jsonSchema**.
- The fields Ecode and Ename must be mandatory.
- Ecode must be of type integer.
- Ename must be of type string.
- isActive (if present) must be a boolean.
- Validation should be strict, and any invalid document should be rejected with an error.
