MongoDB Class 2

Basic Shell Commands

Database Commands:

1. db.version()
2. show dbs
3. use *DB_NAME*
4. db
5. db.dropDatabase()

Collection Commands:

1. db.createCollection(*COLLECTION_NAME*)
2. show collections
3. db.*COLLECTION_NAME*.drop()

CRUD Operations

Operation	Commands
Create	insertOne, insertMany
Read	findOne, find
Update	updateOne, updateMany, replaceOne
Delete	deleteOne, deleteMany

Insert Operation

1. insertOne:

   Command: db.<collection>.insertOne(<document>, options)

   Example:

```
db.students.insertOne({
  name: "Amit",
  age: 21,
  branch: "CSE"
})
```

2. insertMany:

   Command: db.<collection>.insertMany([<document1>, <document2>, ...], options)

   Example:
   ```
   db.students.insertMany([
     { name: "Neha", age: 22, branch: "ECE" },
     { name: "Rahul", age: 20, branch: "ME" }
   ])
   ```

3. insert: *[Depricated command]*

   Command:
   ```
   db.<collection>.insert(<document>)              // Single insert
   db.<collection>.insert([<doc1>, <doc2>, ...])     // Multiple inserts
   ```

   Example:
   ```
   db.students.insert({
     name: "Kriti",
     age: 23,
     branch: "Civil"
   })

   db.students.insert([
     { name: "Sahil", age: 24 },
     { name: "Aarav", age: 19 }
   ])
   ```

Options:

1. Ordered:
   Default value is *True*, i.e. Stop at the first error.
   False: Tries to insert all.

2. writeConcern:
   {w: true, j: false, wtimeout:0}

   w ((Write Acknowledgment)
   The w parameter determines the level of acknowledgment required from the MongoDB server for a write operation to be considered successful.
   a. w: 1 (Default):
      1. Requests acknowledgment from the primary server.
      2. The operation will return an acknowledgment (e.g., acknowledged: true) and the generated _id.
   b. w: 0 (No Acknowledgment):
      1. The write operation is sent to the server, but the client does not wait for any acknowledgment.
      2. This makes the operation faster as there's no waiting time.
      3. However, the client will not know if the write was successful or failed.

   j (Journal Acknowledgment):
   The j parameter ensures that the write operation is recorded in the on-disk journal before being acknowledged. This provides durability against server crashes.
   a. j: true:
      1. The operation is acknowledged only after it has been written to the journal.
      2. This makes the operation slightly slower but guarantees that the data can be recovered even if the server shuts down unexpectedly before the data is fully flushed to the data files.
   b. j: false (Default):

    1. The operation is acknowledged without waiting for it to be written to the journal.
    2. This is faster but carries a small risk of data loss if the server crashes before the data is written to the journal.

wtimeout (Write Timeout)

- The wtimeout parameter specifies a time limit (in milliseconds) for the write operation to be acknowledged by the specified w level.
- If the required acknowledgment (e.g., from the primary or multiple replicas) is not received within the wtimeout period, the operation will fail and return an error, even if the write might eventually succeed.

Read Operation

1. find():

   Command: db.<collection>.find(query, projection)

   Example:
   ```
   db.students.find(
     { branch: "CSE" },
     { name: 1, branch: 1, _id: 0 }
   )
   ```

   What it does:
   - Finds all documents that match the query.
   - Returns a cursor (not the full data directly).
   - You won't see more than 20 documents in shell output until you iterate or convert it.

   To see full results:

   db.students.find({}).toArray()

2. findOne():

   Command: db.<collection>.findOne(query, { projection: { field: 1/0 } })

Example:

```
db.students.findOne(
  { name: "Amit" },
  { projection: { name: 1, branch: 1, _id: 0 } }
)
```

What it does:

- Finds and returns the first matching document only.
- Returns directly as a JSON object, not a cursor.

What is a Cursor?

A cursor is a pointer to the result set of a MongoDB query.

Why cursor?

- It allows MongoDB to return results in chunks (especially for large datasets).
- Helps in memory optimization and pagination.

How it behaves:

- When you do find(), MongoDB doesn’t immediately give you all results.
- You get a cursor, which loads documents in batches (default 20) when using the shell.
- To convert to an array of documents:
  db.students.find().toArray()

Useful in scripts where you need actual data, not a cursor.

Cursor Methods (Important for Pagination & More)

Method	Description
.toArray()	Converts the cursor to an array.
.forEach()	Iterates over each document.
.limit(n)	Limits the number of documents.
.skip(n)	Skips n documents (useful in paging).
.sort({...})	Sorts results based on fields.

Update Operation

1. updateOne():

   Command:
   ```
   db.<collection>.updateOne(
     { <filter> },
     { <update operators> },
     { <options> }
   )
   ```

   Example:
   ```
   db.students.updateOne(
     { name: "Amit" },
     { $set: { branch: "ECE" } },
     { upsert: false }
   )
   ```

2. updateMany():

   Command:
   ```
   db.<collection>.updateMany(
     { <filter> },
     { <update operators> },
     { <options> }
   )
   ```

   Example:
   ```
   db.students.updateMany(
   ```

```
  { branch: "CSE" },
  { $set: { passed: true } },
  { upsert: false }
)
```

3. update: *[Depricated command]*

   Command:
   ```
    db.<collection>.update(
     { <filter> },
     { <update operators> },
     { <options> }
   )// Single/Multiple Update
   ```

   Example:
   ```
   db.students.update(
     { branch: "CSE" },
     { $set: { placement: "on-campus" } },
     { multi: true }
   )
   ```

4. replaceOne:

   Command:
   ```
    db.<collection>.replaceOne(
     <filter>,
     <replacementDocument>,
     { upsert: <boolean> } // optional
   )
   ```

   Example:
   ```
   db.students.replaceOne(
     { rollNo: 101 },
     {
       rollNo: 101,
       name: "Riya",
       branch: "IT"
   ```

```
  }
)
```
∗This replaces the entire document where rollNo: 101.

What is Upsert?

upsert is a special option available in MongoDB update operations (updateOne, updateMany, and replaceOne) that inserts a new document if no document matches the query filter.

Without upsert, if the query doesn't match any document, nothing happens. But with upsert: true, MongoDB will:

- Try to find a document matching the filter.
- If it finds one, it updates it.
- If it doesn't, it creates a new document using the filter + update fields.

Common Update Operators?

Command	Description
$set	Set/Update a field
$unset	Remove a field
$push	Add value to an array (at the end)
$pull	Remove matching values from an array
$rename	Rename a field
$inc	Increment a numeric value
$mul	Multiply numeric value
$min / $max	Set value only if less/greater than current
$addToSet	Add value to array only if not already present

Delete Operation

1. deleteOne():

   Command: db.collection.deleteOne(<filter>, <options?>)

   Example:
   db.students.deleteOne({ branch: "CSE" })

2. deleteMany():

   *Command:*db.collection.deleteMany(<filter>, <options?>)

   Example:
   db.students.deleteMany({ branch: "CSE" })

   *db.students.deleteMany({})
   This will delete all documents in the collection.

CRUD Practice on sample dataset

Create

- Insert a new student into the collection with your own details.
- Add 3 students in one command using insertMany.

Read

- Find all students in the CSE branch.
- Find students who are in 2nd year and are hostellers.
- Find the student who scored more than 90 in Maths.
- Show name and branch of all students (only those fields).
- Count the number of students in CSE branch.
- Get all students sorted by their Physics marks in descending order.

Update

- Update the Chemistry marks of roll number "CSE001" to 80.
- Add a new field cgpa with default value 8.0 to all students.
- Change hosteller status to false for all 4th-year students.

- Rename the field branch to department.

Delete
- Delete the student whose roll number is "ECE002".
- Delete all students who are in 1st year.

MongoDB Datatypes

Datatypes emphasise the importance of understanding these types for effective data modeling and management.

1. Text (Strings)
2. Boolean
3. Number:
    a. Integer (32-bit):
    b. Long (64-bit)
    c. Decimal
4. Object ID
5. ISO Date
6. Timestamp
7. Arrays
8. Embedded Documents (Nested Documents)

**typeOf*: This command is used for getting type of fields