1. Database Operations

- `use <database_name>` → Switch to or create a database.
- `show dbs` → List all databases.
- `db` → Display the current database.
- `show collections` → List all collections in the current database.
- `sdb.collection.drop()` → Delete a collection if that's the only collection then the db is also.
- db.dropDatabase(): Deletes the current database.
- `exit` → Close the shell.

---

2. CRUD Operations (Create, Read, Update, Delete)

Create (Insert Data)

- `db.collection.insertOne({ key: value })` → Insert a single document.
- `db.collection.insertMany([{ key1: value1 }, { key2: value2 }])` → Insert multiple documents.

Read (Query Data)

- `db.collection.find()` → Retrieve all documents.
- `db.collection.find().pretty()` → Format the result for readability.
- `db.collection.find({ key: value })` → Find documents matching a condition.
- `db.collection.findOne({ key: value })` → Find a single document matching a condition.

Update (Modify Data)

- `db.collection.updateOne({ key: value }, { $set: { key: newValue } })` → Update a single document.
- `db.collection.updateMany({ key: value }, { $set: { key: newValue } })` → Update multiple documents.
- `db.collection.replaceOne({ key: value }, { newDocument })` → Replace an entire document.

Delete (Remove Data)

- `db.collection.deleteOne({ key: value })` → Delete a single document.
- `db.collection.deleteMany({ key: value })` → Delete multiple documents.

---

3. Complex Filtering & Logical Operators

Comparison Operators

- `$gt` → `{ key: { $gt: value } }` → Greater than.
- `$lt` → `{ key: { $lt: value } }` → Less than.
- `$gte` → `{ key: { $gte: value } }` → Greater than or equal.
- `$lte` → `{ key: { $lte: value } }` → Less than or equal.
- `$ne` → `{ key: { $ne: value } }` → Not equal.
- `$in` → `{ key: { $in: [value1, value2] } }` → Matches any value in an array.
- `$nin` → `{ key: { $nin: [value1, value2] } }` → Does not match any value in an array.

Logical Operators

- `$and` → `{ $and: [ { key1: value1 }, { key2: value2 } ] }` → Both conditions must be true.
- `$or` → `{ $or: [ { key1: value1 }, { key2: value2 } ] }` → At least one condition must be true.
- `$not` → `{ key: { $not: { condition } } }` → Negates a condition.
- `$nor` → `{ $nor: [ { key1: value1 }, { key2: value2 } ] }` → None of the conditions should be true.

---

4. Complex Update Operations

- `$set` → `{ $set: { key: newValue } }` → Update a specific field.
- `$unset` → `{ $unset: { key: "" } }` → Remove a field.
- `$inc` → `{ $inc: { key: value } }` → Increment a numeric field.
- `$mul` → `{ $mul: { key: value } }` → Multiply a numeric field.
- `$rename` → `{ $rename: { "oldKey": "newKey" } }` → Rename a field.
- `$min` → `{ $min: { key: value } }` → Update only if the new value is smaller.
- `$max` → `{ $max: { key: value } }` → Update only if the new value is larger.
- `$currentDate` → `{ $currentDate: { key: true } }` → Set the field to the current date.

---

5. Read Modifiers

- `db.collection.find({}, { key: 1, _id: 0 })` → Select specific fields (`_id: 0` hides `_id`).
- `db.collection.find().sort({ key: 1 })` → Sort in ascending order (`-1` for descending).
- `db.collection.find().limit(n)` → Limit the number of results.
- `db.collection.find().skip(n)` → Skip the first `n` results.

---

6. Indexing

- `db.collection.createIndex({ key: 1 })` → Create an ascending index.
- `db.collection.createIndex({ key: -1 })` → Create a descending index.
- `db.collection.getIndexes()` → List all indexes.

---

7. Aggregation Framework

Aggregation allows advanced data processing like filtering, grouping, sorting, and transformations.

Basic Syntax
db.collection.aggregate([{ stage1 }, { stage2 }])

Aggregation Stages

- `$match` → `{ $match: { key: value } }` → Filters documents.
- `$group` → `{ $group: { _id: "$field", total: { $sum: "$amount" } } }` → Groups and performs operations.
- `$sort` → `{ $sort: { key: 1 } }` → Sorts documents (`1` for ascending, `-1` for descending).
- `$project` → `{ $project: { field1: 1, field2: 1, _id: 0 } }` → Reshapes documents.
- `$limit` → `{ $limit: n }` → Limits the number of documents.
- `$skip` → `{ $skip: n }` → Skips `n` documents.
- `$count` → `{ $count: "totalDocs" }` → Counts the number of documents.
- `$sum` → `{ $sum: "$field" }` → Calculates the sum of a field.
- `$subtract` → `{ $subtract: "$field" }` → Calculates the subtract of a field.
- `$multiply` → `{ $multiply: "$field" }` → Calculates the multiplication of a field.
- `$divide` → `{ $divide: "$field" }` → Calculates the division of a field.
- `$mod` → `{ $mod: "$field" }` → Calculates the mod of a field.
- `$avg` → `{ $avg: "$field" }` → Calculates the average of a field.
- `$min` → `{ $min: "$field" }` → Finds the minimum value in a field.
- `$max` → `{ $max: "$field" }` → Finds the maximum value in a field.

Example Aggregation Query

```
db.orders.aggregate([
  { $match: { status: "completed" } },
  { $group: { _id: "$customer", totalSpent: { $sum: "$amount" } } },
  { $sort: { totalSpent: -1 } },
  { $limit: 5 }
]);
```