**Date: 21-07-2025**

**MongoDB**

**Overview of NoSQL Databases:**

**NoSQL (Not Only SQL)** databases are designed for flexibility, scalability, and handling large volumes of unstructured or semi-structured data.

**Key Characteristics:**

* Schema-less (dynamic structure)
* Stores data in JSON-like or key-value formats
* Horizontally scalable (easier to scale out)
* Suitable for big data, real-time applications, IoT, etc.

**Types of NoSQL Databases:**

| **Type** | **Example** | **Data Format** |
| --- | --- | --- |
| Document Store | MongoDB | JSON/BSON |
| Key-Value Store | Redis, DynamoDB | Key: Value |
| Column Store | Apache Cassandra | Columns & Rows |
| Graph DB | Neo4j | Nodes & Edges |

**Key Features of MongoDB:**

| **Feature** | **Description** |
| --- | --- |
| Document-Oriented | Stores data as BSON documents (similar to JSON). |
| Dynamic Schema | No need to define the structure (schema) beforehand. |
| High Performance | Fast reads/writes with indexing and in-memory storage. |
| Rich Query Language | Supports powerful queries using conditions, projections, regex, etc. |
| Replication & Sharding | Built-in support for high availability and horizontal scaling. |
| Embedded Data | Related data can be stored together (no need for complex joins). |
| Aggregation Framework | Supports complex data processing using pipelines ($match, $group, etc.) |

**MongoDB vs. Traditional RDBMS**

| **Feature** | **Traditional RDBMS (SQL)** | **MongoDB (NoSQL)** |
| --- | --- | --- |
| Data Format | Tables (Rows & Columns) | Documents (BSON/JSON) |
| Schema | Fixed (Predefined structure) | Flexible (Schema-less) |
| Scalability | Vertical (scale-up) | Horizontal (scale-out) |
| Joins | Strong support with JOINs | Limited, use $lookup in aggregations |
| Transactions | Fully ACID-compliant | Supports ACID on replica sets (multi-doc) |
| Use Case | Structured data, relational models | Big data, real-time, unstructured data |

**Installing MongoDB on Windows:**

1. **Download MongoDB Installer**:  
    Go to<https://www.mongodb.com/try/download/community>
2. **Run the Installer**:

* Choose **Complete Setup**
* Keep **MongoDB Compass** checked if you want GUI (optional)

1. **Finish Setup**:

* After installation, MongoDB will be located in:  
   C:\Program Files\MongoDB\Server\<version>\bin

1. **Add MongoDB to PATH** (optional but recommended):

* Add the bin folder to the environment variable PATH to use mongosh globally from cmd.

1. **Start Mongo Shell**:  
     
   mongosh

**SQL vs MongoDB — Mnemonics:**

Table → Collection

Row → Document

Column → Field

SQL SELECT → MongoDB find()

Primary Key → \_id

**SQL vs MongoDB — Basic Terms Comparison:**

| **SQL Term** | **MongoDB Term** | **Description** |
| --- | --- | --- |
| Database | Database | Same in both: a container for collections/tables. |
| Table | Collection | A set of rows (in SQL) or documents (in MongoDB). |
| Row (Record) | Document | A single entry of data (JSON object in MongoDB). |
| Column | Field | A data attribute (like name, age, etc.). |
| Primary Key | \_id field | Uniquely identifies each document. MongoDB auto-generates \_id. |
| Schema | Dynamic Schema | SQL requires schema definition; MongoDB allows flexible, dynamic structure. |
| JOIN | $lookup (Aggregation) | Used to join documents from different collections. |
| INSERT INTO | insertOne / insertMany | For inserting records/documents. |
| SELECT | find() | For retrieving data. |
| WHERE | Query Filter Object | Used to filter documents with conditions. |
| UPDATE | updateOne / updateMany | To modify documents. |
| DELETE FROM | deleteOne / deleteMany | To delete documents. |
| CREATE TABLE | createCollection() | Used to create a new table/collection. |
| ALTER TABLE | ❌ Not needed | MongoDB collections are schema-less; no alter command required. |
| DROP TABLE | drop() | Deletes a collection. |
| TRUNCATE TABLE | deleteMany({}) | Deletes all documents from a collection. |
| SELECT COUNT(\*) | count() / countDocuments() | To count the number of matching documents. |
| DISTINCT | distinct() | Fetches unique values for a field. |

**Collection Operations:**

1. **Create Collection:**

db.createCollection("user")

db.createCollection("collection\_name")

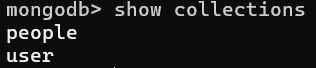
*Creates a new collection in the current database.*



1. **Show All Collections:**

show collections

*Lists all collections in the current database.*

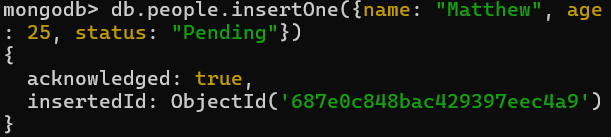
**

1. **Create (Insert) Operations:**

* Insert One Document:

db.people.insertOne({ name: "Matthew", age: 25, status: "Pending" })

*Adds a single document to the people collection.*

**

* Insert Multiple Documents:

db.people.insertMany([

{ name: "Clare", age: 21, status: "Pending" },

{ name: "Bickin", age: 19, status: "Pending" },

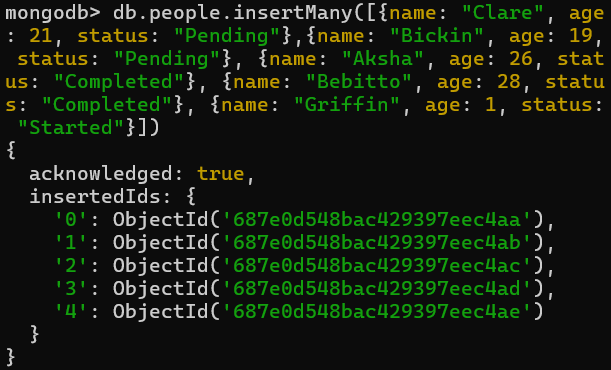
{ name: "Aksha", age: 26, status: "Completed" },

{ name: "Bebitto", age: 28, status: "Completed" },

{ name: "Griffin", age: 1, status: "Started" }

])

*Adds multiple documents to the people collection in one call.*

**

**SQL vs MongoDB Insert**

SQL:

INSERT INTO people (user\_id, age, status) VALUES ("bcd001", 45, "A");

MongoDB:

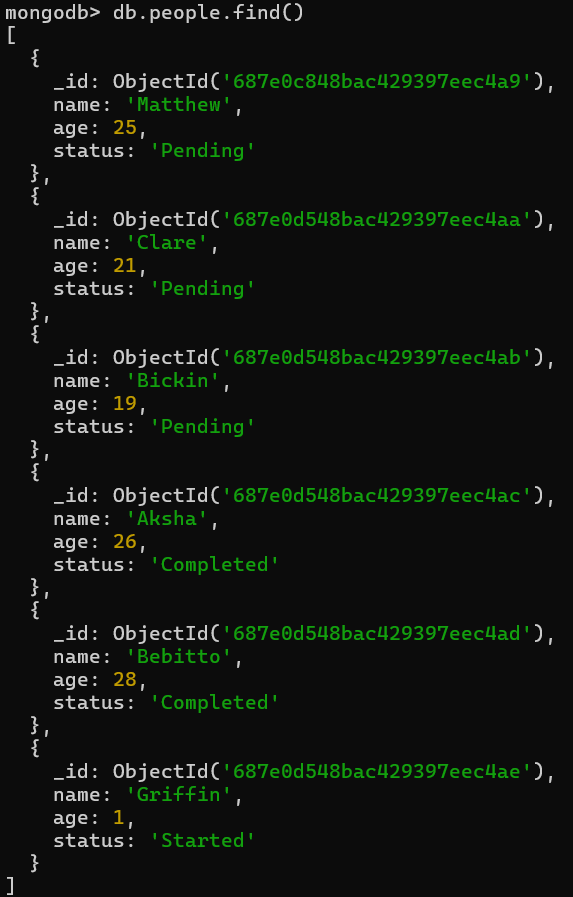
db.people.insertOne({ user\_id: "bcd001", age: 45, status: "A" })

*MongoDB uses JSON-like objects to store data; there's no fixed schema like in SQL.*

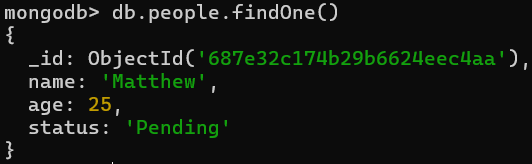
**Read (Find) Operations:**

Basic Queries:

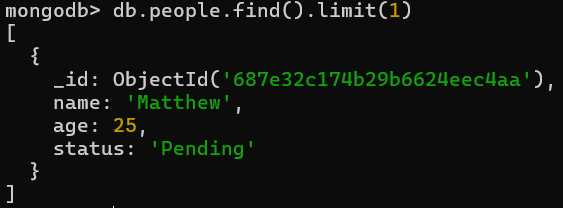
1. db.people.find() // Find all



1. db.people.findOne() // Find first document

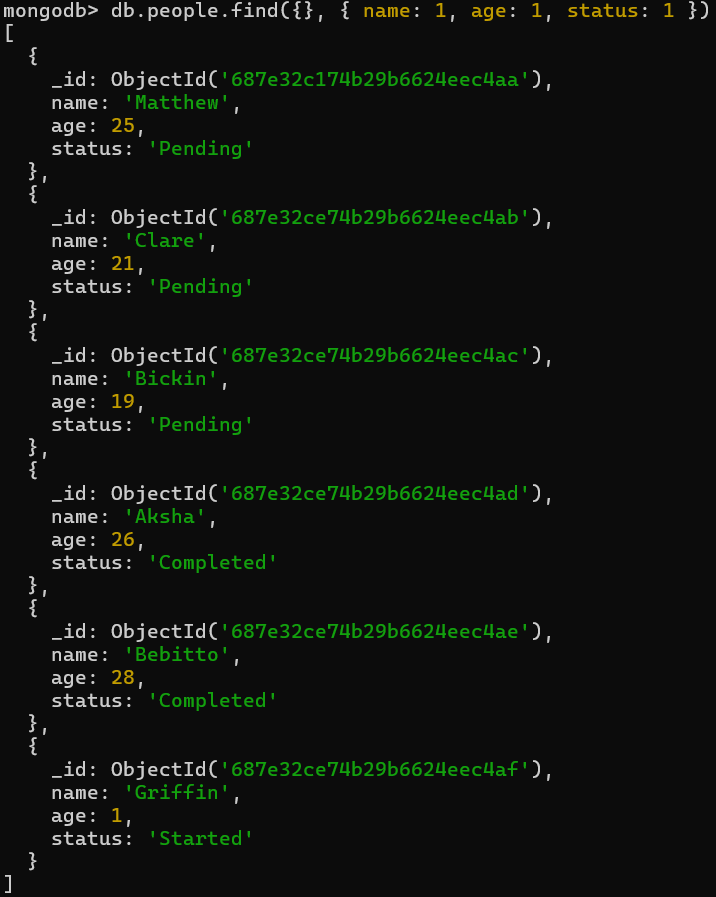


1. db.people.find().limit(1) // Limit to 1 document

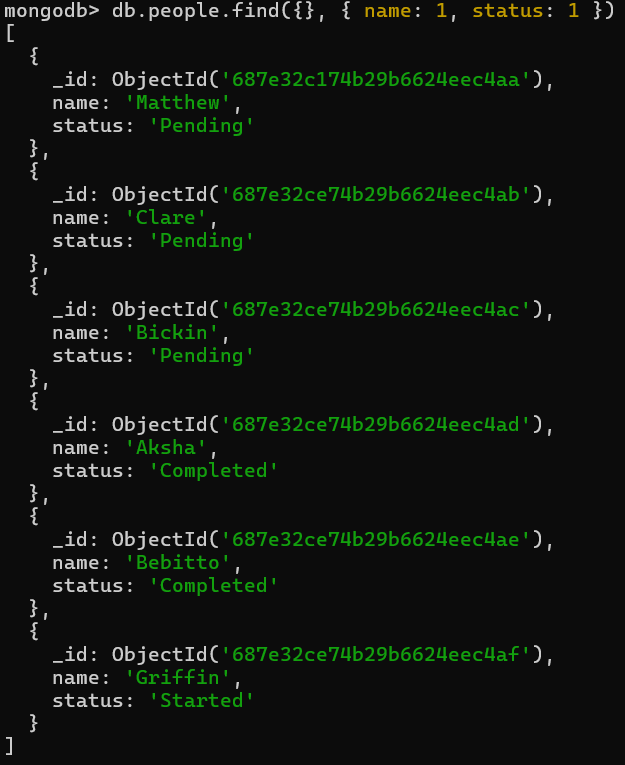


Field Projections:

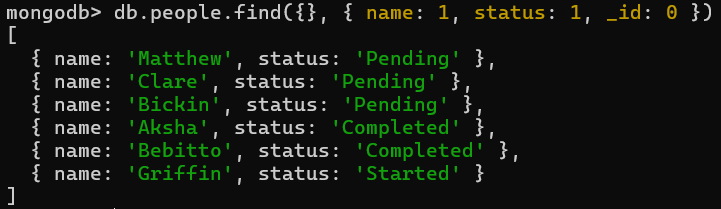
1. db.people.find({}, { name: 1, age: 1, status: 1 }) // Include specific fields



1. db.people.find({}, { name: 1, status: 1 }) // Only name & status

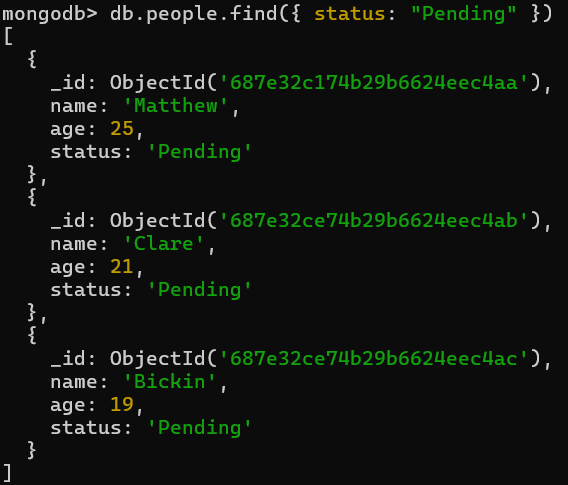


1. db.people.find({}, { name: 1, status: 1, \_id: 0 }) // Exclude \_id

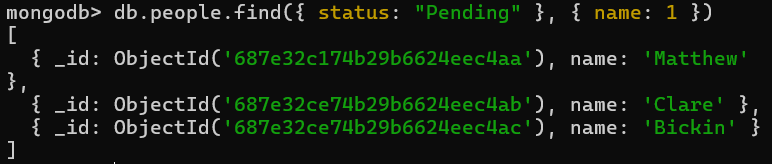


Conditional Queries:

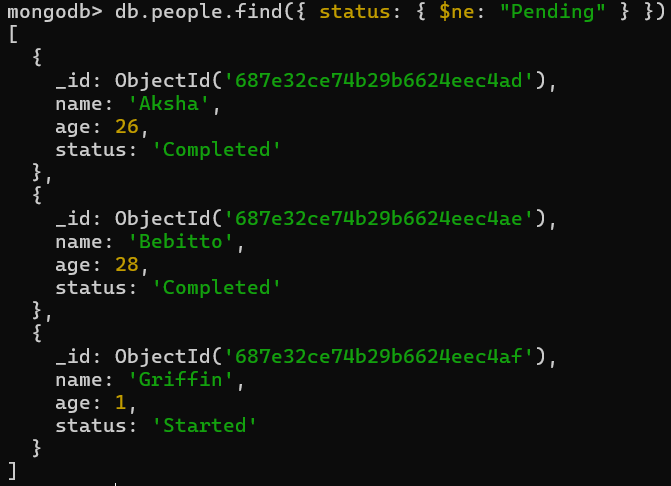
1. db.people.find({ status: "Pending" }) // Match exact status



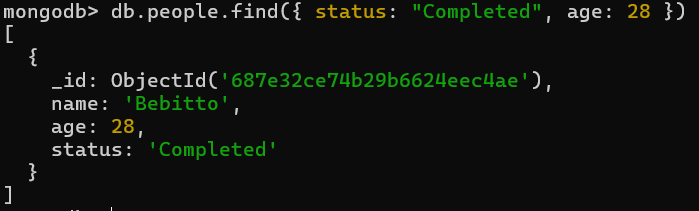
1. db.people.find({ status: "Pending" }, { name: 1 }) // Return only name



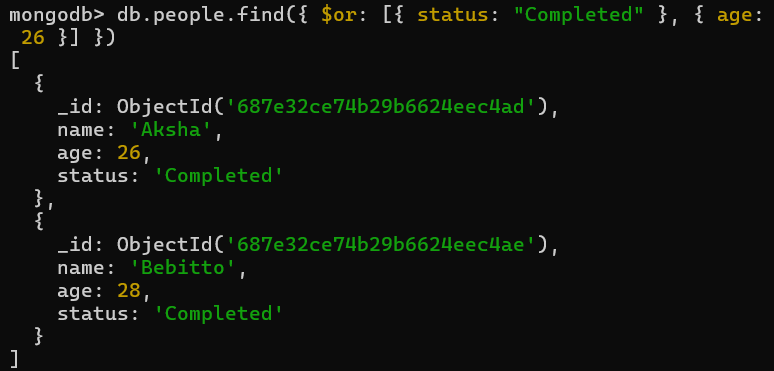
1. db.people.find({ status: { $ne: "Pending" } }) // Not equal



1. db.people.find({ status: "Completed", age: 28 }) // Multiple conditions (AND)



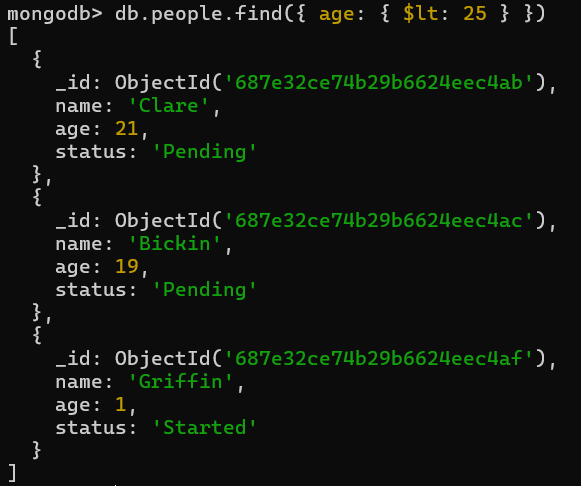
1. db.people.find({ $or: [{ status: "Completed" }, { age: 26 }] }) // OR query



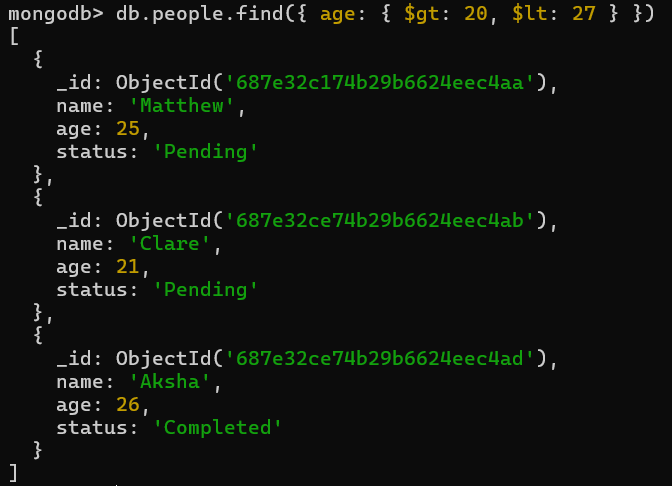
1. db.people.find({ age: { $gt: 25 } }) // Greater than



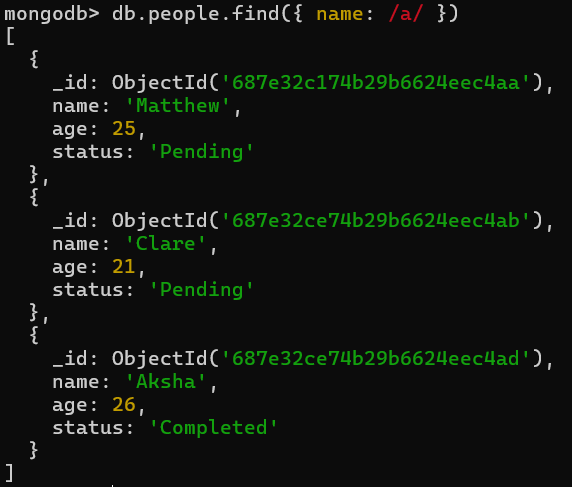
1. db.people.find({ age: { $lt: 25 } }) // Less than

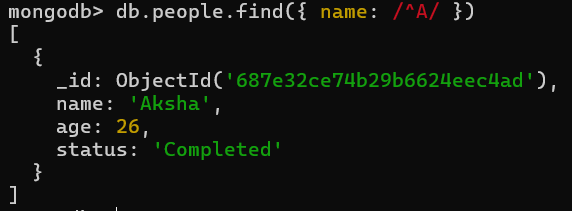


1. db.people.find({ age: { $gt: 20, $lt: 27 } }) // Range



Regex Matching (Pattern Search):

1. db.people.find({ name: /a/ }) // Contains letter "a"
2. db.people.find({ name: /^A/ }) // Starts with "A"



Sorting:

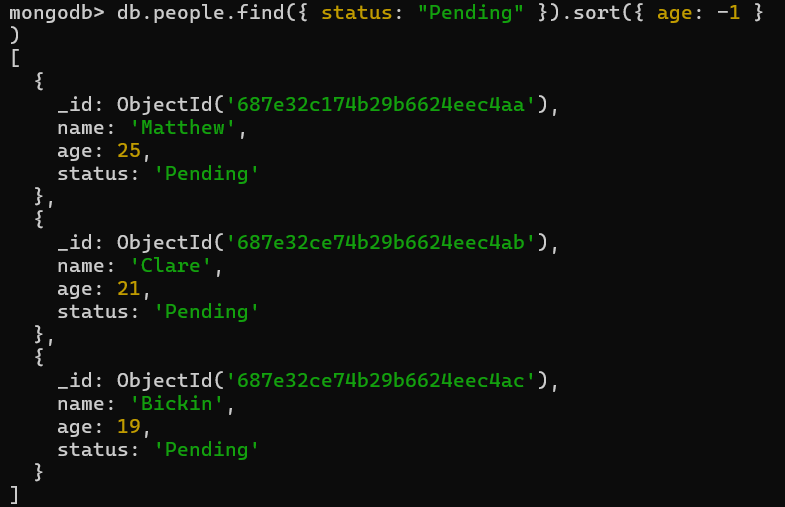
1. db.people.find().sort({ age: 1 }) // Ascending by age



1. db.people.find({ status: "Completed" }).sort({ age: 1 }) // Filter + sort



1. db.people.find({ status: "Pending" }).sort({ age: -1 }) // Descending by age



Count Documents:

1. db.people.count() // Total count
2. db.people.find().count() // Same as above



1. db.people.find({ age: { $exists: true } }).count() // With age field



1. db.people.find({ age: { $gt: 20 } }).count() // Count with condition



1. db.people.find({ age: { $lt: 20 } }).count()



1. db.people.find({ age: { $gt: 20, $lt: 22 } }).count()



Distinct Values:

1. db.people.distinct("status") // All unique status values



1. db.people.distinct("age") // All unique age values



Query Execution Info:

1. db.people.find({ status: "Completed" }).explain()

*Shows how MongoDB executes the query — useful for performance tuning.*



**SQL vs MongoDB — Command Comparison Table**

| **Operation** | **SQL Syntax** | **MongoDB Syntax** |
| --- | --- | --- |
| **Select all records** | SELECT \* FROM people; | db.people.find() |
| **Select with condition** | SELECT \* FROM people WHERE status = 'Pending'; | db.people.find({ status: "Pending" }) |
| **Select with multiple filters** | SELECT \* FROM people WHERE status = 'Completed' AND age = 28; | db.people.find({ status: "Completed", age: 28 }) |
| **OR condition** | SELECT \* FROM people WHERE status = 'Completed' OR age = 26; | db.people.find({ $or: [{ status: "Completed" }, { age: 26 }] }) |
| **Greater than condition** | SELECT \* FROM people WHERE age > 25; | db.people.find({ age: { $gt: 25 } }) |
| **Less than condition** | SELECT \* FROM people WHERE age < 25; | db.people.find({ age: { $lt: 25 } }) |
| **Between two values** | SELECT \* FROM people WHERE age BETWEEN 20 AND 27; | db.people.find({ age: { $gt: 20, $lt: 27 } }) |
| **Pattern matching (LIKE '%a%')** | SELECT \* FROM people WHERE name LIKE '%a%'; | db.people.find({ name: /a/ }) |
| **Pattern matching (starts with 'A')** | SELECT \* FROM people WHERE name LIKE 'A%'; | db.people.find({ name: /^A/ }) |
| **Sorting ASC** | SELECT \* FROM people ORDER BY age ASC; | db.people.find().sort({ age: 1 }) |
| **Sorting DESC** | SELECT \* FROM people ORDER BY age DESC; | db.people.find().sort({ age: -1 }) |
| **Count rows** | SELECT COUNT(\*) FROM people; | db.people.count() or db.people.find().count() |
| **Insert one row** | INSERT INTO people (name, age, status) VALUES ('Matthew', 25, 'Pending'); | db.people.insertOne({ name: "Matthew", age: 25, status: "Pending" }) |
| **Insert multiple rows** | INSERT INTO people (name, age, status) VALUES (...), (...); | db.people.insertMany([{...}, {...}, ...]) |
| **Select specific columns** | SELECT name, status FROM people; | db.people.find({}, { name: 1, status: 1 }) |
| **Exclude a column** | *(Not directly supported — only include needed columns)* | db.people.find({}, { age: 0 }) |
| **Distinct values from column** | SELECT DISTINCT status FROM people; | db.people.distinct("status") |
| **Explain query plan** | EXPLAIN SELECT \* FROM people WHERE status = 'Completed'; | db.people.find({ status: "Completed" }).explain() |

**Update Operations:**

1. updateOne()

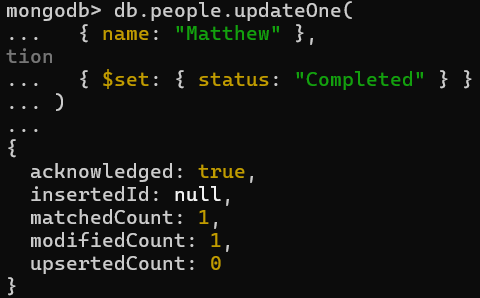
db.people.updateOne(

{ name: "Matthew" }, // Filter condition

{ $set: { status: "Completed" } } // Update operation

)

*Updates the first document that matches the filter. In this case, it changes Matthew’s status to "Completed".*

**

2. updateMany()

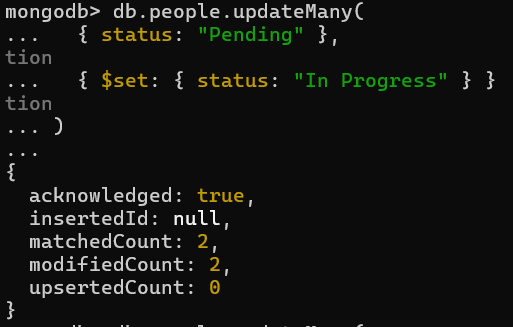
db.people.updateMany(

{ status: "Pending" }, // Filter condition

{ $set: { status: "In Progress" } } // Update operation

)

*Updates all documents where status is "Pending" to "In Progress".*

**

3. replaceOne()

db.people.replaceOne(

{ name: "Griffin" }, // Filter condition

{ name: "Griffin", age: 2 } // Entire replacement document

)

*Replaces the entire document for Griffin with the new one — old fields like status will be lost unless included.*

**

**Delete Operations:**

1. deleteOne():

db.people.deleteOne({ name: "Matthew" })

*removes only the first match.*



1. deleteMany():

db.people.deleteMany({ status: "Pending" })

*for bulk deletion.*



1. Delete all Documents:

db.people.deleteMany({})



1. Drop whole Collection:

db.people.drop()

