National University of Computer & Emerging Sciences, Peshawar

FAST School of Computing –Artificial Intelligence Department
Spring 2025, Lab Manual – 12

Course Code: CL-2005	**Course: Database Systems Lab**
Instructor:	**Yasir Arfat**

MongoDB – From Basics to Advanced Operations
Overview

This lab builds on Lab 11, where you learned MongoDB basics like creating databases, collections, and querying documents. Lab 2 introduces more powerful features to make your database faster, smarter, and more flexible. We'll start with simple concepts (like sorting and limiting results) and move to advanced ones Each section includes real data examples and step-by-step instructions.

Contents

1. Sorting and Limiting Results

Sorting arranges documents in a specific order (e.g., by name or date). Limiting controls how many documents you get back.

Sorting

- Use sort({ field: 1 }) for ascending order (A-Z, 1-9).
- Use sort({ field: -1 }) for descending order (Z-A, 9-1).

Example: Sort books by publication year (newest first).

```
db.books.find().sort({ "year": -1 })
```

Sample Data (Books collection):

```
[
  { "title": "1984", "author": "George Orwell", "year": 1949 },
  { "title": "The Alchemist", "author": "Paulo Coelho", "year": 1988 },
  { "title": "Harry Potter", "author": "J.K. Rowling", "year": 1997 }
]
```

Output:

```
{ "title": "Harry Potter", "year": 1997, ... }
{ "title": "The Alchemist", "year": 1988, ... }
{ "title": "1984", "year": 1949, ... }
```

Limiting

- Use limit(n) to return only n documents.
- Combine with skip(m) to skip the first m documents (useful for pagination).

Example: Get the 2 newest books.

```
db.books.find().sort({ "year": -1 }).limit(2)
```

Output:

```
{ "title": "Harry Potter", "year": 1997, ... }
{ "title": "The Alchemist", "year": 1988, ... }
```

2. Counting Documents

Count how many documents match a query or exist in a collection.

Example

- Count all books:

```
db.books.countDocuments()
```

Output: 3 (based on sample data above).

- Count books published after 1980:

```
db.books.countDocuments({ "year": { "$gt": 1980 } })
```

Output: 2 (The Alchemist and Harry Potter).

3. Projection (Selecting Fields)

Projection lets you choose which fields to include or exclude in query results.

- Use { field: 1 } to include a field.
- Use { field: 0 } to exclude a field. (Note: _id is always included unless explicitly excluded.)

Example: Get book titles and authors only.

```
db.books.find({}, { "title": 1, "author": 1, "_id": 0 })
```

Output:

```
{ "title": "1984", "author": "George Orwell" }
{ "title": "The Alchemist", "author": "Paulo Coelho" }
{ "title": "Harry Potter", "author": "J.K. Rowling" }
```

4. Indexing for Performance

Indexes make queries faster by avoiding full collection scans. Think of them like a book's index, helping you find pages quickly.

Creating an Index

- Single field index:

```
db.books.createIndex({ "author": 1 })
```

- Compound index (multiple fields):

```
db.books.createIndex({ "year": 1, "title": -1 })
```

Example: Query using the index.

```
db.books.find({ "author": "George Orwell" })
```

Without an index, MongoDB scans all documents. With the index, it's much faster.

View Indexes

```
db.books.getIndexes()
```

Drop an Index

```
db.books.dropIndex({ "author": 1 })
```

5. Aggregation Framework

Aggregation in MongoDB is the process of processing multiple documents and transforming them into summarized results.

It is similar to SQL operations like GROUP BY, SUM(), AVG(), COUNT(), etc.

Aggregation allows you to:

Filter documents
Group data together
Calculate totals, averages, maximum, minimum, etc.
Sort and reshape data into a new format

It is mainly done using the aggregate() method, where we pass a series of **stages** like:

$match → to filter documents (like WHERE clause)
$group → to group documents and perform calculations
$sort → to sort the result
$project → to reshape documents (choose fields to show or hide)

📍 Example Data

Imagine we have a MongoDB collection called **orders** with these documents:

```
[
  { "_id": 1, "item": "apple", "price": 10, "quantity": 5 },
  { "_id": 2, "item": "banana", "price": 5, "quantity": 10 },
  { "_id": 3, "item": "orange", "price": 8, "quantity": 7 },
  { "_id": 4, "item": "apple", "price": 10, "quantity": 3 },
  { "_id": 5, "item": "banana", "price": 5, "quantity": 5 }
]
```

Each document represents:

- The **item** name (e.g., apple, banana)
- The **price** per unit
- The **quantity** ordered

Simple Aggregation Example

Suppose you want to find **total quantity** ordered for each **item**.

You can use the aggregation pipeline like this:

```
db.orders.aggregate([
  {
    $group: {
      _id: "$item",               // Group by 'item' field
      totalQuantity: { $sum: "$quantity" }  // Sum the 'quantity' field
    }
  }
])
```

Output:

```
[
  { "_id": "apple", "totalQuantity": 8 },
  { "_id": "banana", "totalQuantity": 15 },
  { "_id": "orange", "totalQuantity": 7 }
]
```

Meaning:

Total apples ordered = 8

Total bananas ordered = 15

Total oranges ordered = 7

6. Text Search and Regular Expressions

MongoDB supports searching text fields with text indexes or pattern matching with regular expressions.

Text Search

1. Create a text index:

```
db.books.createIndex({ "title": "text", "author": "text" })
```

2. Search for books with "Harry" in title or author:

```
db.books.find({ $text: { $search: "Harry" } })
```

Output:

```
{ "title": "Harry Potter", "author": "J.K. Rowling", "year": 1997, ... }
```

Regular Expressions

Find books with titles starting with "The":

```
db.books.find({ "title": { $regex: "^The", $options: "i" } })
```

Output:

```
{ "title": "The Alchemist", "author": "Paulo Coelho", "year": 1988, ... }
```

8. Bulk Write Operations

Bulk writes combine multiple operations (insert, update, delete) into one command for efficiency.

Example: Update a book and insert a new one.

```
db.books.bulkWrite([
  { updateOne: { filter: { title: "1984" }, update: { $set: { copies: 10 } } } },
  { insertOne: { document: { title: "Dune", author: "Frank Herbert", year: 1965, copies: 4 } } }
])
```

Output:

```
{
  "acknowledged": true,
  "insertedCount": 1,
  "matchedCount": 1,
  "modifiedCount": 1,
  "deletedCount": 0
}
```

9. Data Validation

Validation ensures documents meet specific rules (e.g., required fields, data types).

Example: Create a collection requiring positive book copies.

```
db.createCollection("books", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["title", "copies"],
      properties: {
        title: { bsonType: "string" },
        copies: { bsonType: "int", minimum: 0 }
      }
    }
  }
})
db.books.insertOne({ "title": "Test Book", "copies": 5 }) // Works
db.books.insertOne({ "title": "Invalid", "copies": -1 }) // Fails
```

Error: Document failed validation.