**day03\_107856406\_dsdipt\_sudipto\_21may2025**

**Employee Code:** 107856406

**Login ID:** dsdipt

**Email :** dsdipt@amazon.com

**Name:** Sudipto Das

**Date:** 21 May 2025 (Day 03)

### ***Task 1: Create a file named 21st May.txt and push it to your github.***

**git add "21st May.txt"**

**git commit -m "Added recap for 21st May session"**

**git push origin main**

### ***Task 2: ACID - definitions***

**Atomicity**The entire transaction happens completely or not at all.  
Example: If ₹100 is debited from Account A but fails to credit Account B, the debit is rolled back, and no money is lost.

**Consistency**The database remains in a valid state before and after the transaction. Correctness of data.  
The total balance in the system (A + B) remains the same before and after the transfer, ensuring data integrity.

**Isolation**Transactions don't affect each other while executing concurrently.  
Example: If two users are transferring money at the same time, each transfer works as if it's happening alone—no overlapping data errors.

**Durability**Once a transaction is committed, the changes are saved permanently.  
Example: After the ₹100 transfer is complete, even if the system crashes, the transaction won't be lost—the balances remain updated.

### ***Task 3: MongoDB Installation***

**For Windows (using MongoDB Community Server MSI Installer)**

**Download:** Go to [MongoDB Download Center](https://www.mongodb.com/try/download/community)

* + Select "Community Server". Choose your Windows version (e.g., "Windows (x86\_64)") and "MSI" package.
  + Click "Download".

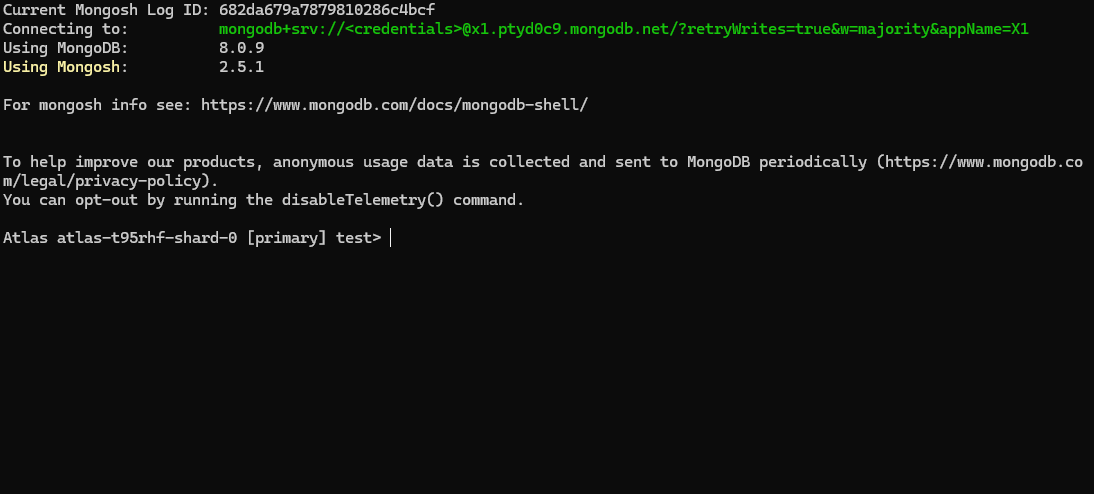
**Run Installer:** Double-click the downloaded .msi file.

* + Choose "Complete" installation (recommended).
  + **Crucially, check "Install MongoDB Compass"** (the GUI tool).

**Verify:** Open Command Prompt/PowerShell.

* + Navigate to C:\Program Files\.....
  + Type mongod --version and mongosh --version to confirm.

**Connect:** Open "MongoDB Compass" from your Start Menu or type mongosh in the bin directory's terminal.



### ***Task 4: MongoDB CRUD operations***

🔹 **CREATE – insertOne() / insertMany()**

// Insert a single document

db.users.insertOne({

name: "Alice", age: 28, email: "[alice@example.com](mailto:alice@example.com)"})

// Insert multiple documents

db.users.insertMany([

{ name: "Bob", age: 30, email: "bob@example.com" },

{ name: "Charlie", age: 25, email: "charlie@example.com" }])

**🔍 READ – find() / findOne()**

// Find all users

db.users.find()

// Find users with a specific condition

db.users.find({ age: { $gt: 26 } })

// Find a single document

db.users.findOne({ name: "Alice" })

// Project only specific fields

db.users.find({ age: { $gt: 25 } }, { name: 1, email: 1 })

**✏️ UPDATE – updateOne() / updateMany() / replaceOne()**

// Update one document

db.users.updateOne(

{ name: "Alice" },

{ $set: { age: 29 } }

)

// Update multiple documents

db.users.updateMany(

{ age: { $lt: 30 } },

{ $inc: { age: 1 } }

)

// Replace an entire document

db.users.replaceOne(

{ name: "Charlie" },

{ name: "Charlie", age: 26, email: "newcharlie@example.com" }

)

**❌ DELETE – deleteOne() / deleteMany()**

// Delete one document

db.users.deleteOne({ name: "Alice" })

// Delete multiple documents

db.users.deleteMany({ age: { $lt: 30 } })

### ***Task 5: Explore MongoDB***

**🔁 Replica Set in MongoDB**

A replica set is a group of MongoDB servers that maintain the same data set, providing high availability and data redundancy.

* Primary node – receives all write operations.
* Secondary nodes – replicate data from the primary. They can serve read operations (if configured).
* If the primary fails, one of the secondaries is automatically elected as the new primary.
* This supports automatic failover and data durability.

**Example**

# A 3-node replica set:

- mongo1.example.net (Primary)

- mongo2.example.net (Secondary)

- mongo3.example.net (Secondary)

**Use case:** Prevent data loss in production systems and ensure uptime.

**🧱 Scalable**

Scalability is the ability of a system to handle increased load by adding more resources.

**In MongoDB:**

* Vertical scaling – Add more CPU/RAM to a single server.
* Horizontal scaling – Add more machines using sharding.

MongoDB supports horizontal scalability via sharding, where data is distributed across multiple servers (shards) based on a shard key.

**🔄 Resilient**

A resilient system can recover quickly from failures or continue operating even when parts of the system fail.

**In MongoDB:**

* Replica sets make MongoDB resilient to server crashes.
* If a primary goes down, a secondary is elected to maintain availability.

Resilience is about fault tolerance, self-healing, and minimizing downtime.

**🚨 Mission-Critical**

Mission-critical systems are essential to the operation of a business or service. If they fail, it causes significant disruption, loss, or danger.

**Examples:**

* E-commerce checkout system
* Online banking
* Hospital record systems

**In MongoDB:**

MongoDB is often used in mission-critical applications with replica sets and sharding, plus monitoring and backups, to ensure zero downtime, high availability, and data integrity.

**Authentication and Authorization**

**What is authentication?**  
Authentication verifies who you are, typically using credentials like a username and password.

**What is authorization?**  
Authorization determines what you are allowed to do, such as access levels or permissions.

Which comes first: authentication or authorization?  
Authentication comes first; you must prove your identity before being granted access.

**Encryption & Decryption**

Encryption converts data into a secret code, while decryption restores it to its original form.

**What is a Capped Collection?**

A capped collection is a fixed-size collection in MongoDB that automatically overwrites its oldest entries when it reaches its maximum size. These are useful for logging, caching, or streaming use-cases where you only care about the most recent data.

**Key Properties:**

* Fixed size (specified in bytes).
* Insertion order is preserved.
* Oldest documents are automatically removed when the size limit is reached.
* No deletions or updates that increase document size are allowed (since it breaks the size cap).

**Is the name “capped” mandatory?**

No, the name can be anything. "capped" is just an option used when creating the collection, not part of the collection name. You can name your collection anything you want.

### ***Task 6: MongoDB Operations***

**Create Collections**

**Implicit Creation (via insert):**

db.students.insertOne({ name: "kim", age: 12, standard: "seventh" })

**Explicit Creation:**

db.createCollection("teachers")

**Insert Multiple Documents**

db.teachers.insertMany([

{ name: "roger", subject: "maths" },

{ name: "anita", subject: "chemistry" },

{ name: "thomas", subject: "physics" },

{ name: "tina", subject: "maths" },

{ name: "jonty", subject: "english" }

])

**Finding Documents**

**Find all documents:**

db.teachers.find()

**Pretty print format:**

db.teachers.find().forEach(printjson)

**Capped Collections**

**Create a capped collection:**

db.createCollection("cap1", { capped: true, size: 10000 })

**Check if capped:**

db.cap1.isCapped()

**Limit entries with max:**

db.createCollection("cap2", { capped: true, size: 100000, max: 5 })

**Insert into capped collection:**

db.cap2.insertMany([

{ \_id: 10, name: "neena", age: 12, standard: "fifth" },

{ \_id: 11, name: "prince", age: 7, standard: "third" },

{ \_id: 12, name: "leeta", age: 8, standard: "fourth" },

{ \_id: 13, name: "jhon", age: 4, standard: "first" }

])

db.cap2.insertMany([

{ \_id: 14, name: "sheena", age: 16, standard: "junior" },

{ \_id: 15, name: "kris", age: 18, standard: "senior" }

])

**View entries:**

db.cap2.find()

**Convert a collection to capped:**

db.createCollection("convertcap1")

db.runCommand({ convertToCapped: "convertcap1", size: 10000 })

**Bulk Operations**

**1. Ordered Bulk (stops at first error):**

var bulkOrdered = db.bulk1.initializeOrderedBulkOp()

bulkOrdered.insert({ \_id: 1, name: "Murugan" })

bulkOrdered.insert({ \_id: 2, name: "Niyas" })

bulkOrdered.insert({ \_id: 3, name: "Nitin" })

bulkOrdered.insert({ \_id: 1, name: "Murugan" }) // duplicate

bulkOrdered.insert({ \_id: 4, name: "Adarsh" })

bulkOrdered.execute()

**2. Unordered Bulk (tries all even with errors):**

var bulkUnordered = db.bulk2.initializeUnorderedBulkOp()

bulkUnordered.insert({ \_id: 1, name: "Murugan" })

bulkUnordered.insert({ \_id: 2, name: "Niyas" })

bulkUnordered.insert({ \_id: 3, name: "Nitin" })

bulkUnordered.insert({ \_id: 1, name: "Murugan" }) // duplicate

bulkUnordered.insert({ \_id: 4, name: "Adarsh" })

bulkUnordered.execute()

**Delete Operations**

**Insert sample data into deleteplist:**

db.deleteplist.insertMany([

{ product: "register", quantity: 25, size: { height: 14, weight: 21, unit: "cm" }, status: "A" },

{ product: "book", quantity: 50, size: { height: 8.5, weight: 11, unit: "in" }, status: "A" },

{ product: "sheet", quantity: 100, size: { height: 8.5, weight: 11, unit: "in" }, status: "D" },

{ product: "timetable", quantity: 75, size: { height: 22.85, weight: 30, unit: "cm" }, status: "D" },

{ product: "envelop", quantity: 45, size: { height: 10, weight: 15.25, unit: "cm" }, status: "A" }

])

**Delete Examples**

**Delete by field value:**

db.deleteplist.deleteOne({ product: "sheet" })

**Delete by embedded document:**

db.deleteplist.deleteOne({ size: { height: 14, weight: 21, unit: "cm" } })

**Delete by field inside embedded doc:**

db.deleteplist.deleteOne({ "size.unit": "in" })

**Delete using numeric condition:**

db.deleteplist.deleteOne({ "size.height": { $lt: 15 } })

**Insert additional data with arrays:**

db.deleteplist.insertMany([

{ product: "register", quantity: 25, tags: ["blank", "red"], dim\_cm: [14, 21] },

{ product: "book", quantity: 50, tags: ["red", "blank"], dim\_cm: [14, 21] },

{ product: "sheet", quantity: 100, tags: ["red", "blank", "plain"], dim\_cm: [14, 21] },

{ product: "timetable", quantity: 75, tags: ["blank", "red"], dim\_cm: [22.85, 30] },

{ product: "envelop", quantity: 45, tags: ["blue"], dim\_cm: [10, 15.25] }

])

**Delete using exact array match:**

db.deleteplist.deleteOne({ tags: ["red", "blank"] }) // Order matters

db.deleteplist.deleteMany({ tags: ["red", "blank"] })

**Delete by array element value:**

db.deleteplist.deleteMany({ tags: "red" })

**Embedded Array of Documents**

**Insert data:**

db.deleteplist.insertMany([

{ product: "register", instock: [ { warehouse: "A", quantity: 5 }, { warehouse: "C", quantity: 15 } ] },

{ product: "book", instock: [ { warehouse: "C", quantity: 5 } ] },

{ product: "sheet", instock: [ { warehouse: "A", quantity: 60 }, { warehouse: "B", quantity: 15 } ] },

{ product: "timetable", instock: [ { warehouse: "A", quantity: 40 }, { warehouse: "B", quantity: 5 } ] },

{ product: "envelop", instock: [ { warehouse: "B", quantity: 15 }, { warehouse: "C", quantity: 35 } ] }

])

**Delete exact embedded match:**

db.deleteplist.deleteMany({ instock: { warehouse: "A", quantity: 5 } })

**Deprecated Method**

**Old way to delete:**

db.deleteplist.remove({ "instock.quantity": { $lte: 20 } })

🛑 *Deprecated in MongoDB 6+. Use deleteMany() instead.*

**Timestamp-Based Deletion**

**Insert documents with timestamps:**

db.deleteplist.insertMany([

{

product: "register",

status: "A",

size: { height: 14, weight: 21, unit: "cm" },

instock: [ { warehouse: "A", quantity: 5 } ],

creationts: ISODate("2015-11-01T12:30:15Z")

},

{

product: "book",

status: "A",

size: { height: 8.5, weight: 11, unit: "in" },

instock: [ { warehouse: "C", quantity: 5 } ],

creationts: ISODate("2015-11-01T12:30:15Z")

},

{

product: "sheet",

status: "D",

size: { height: 8.5, weight: 11, unit: "in" },

instock: [ { warehouse: "A", quantity: 60 } ],

creationts: ISODate("2015-11-01T12:30:15Z")

},

{

product: "timetable",

status: "D",

size: { height: 22.85, weight: 30, unit: "cm" },

instock: [ { warehouse: "A", quantity: 40 } ],

creationts: ISODate("2015-11-01T12:30:15Z")

},

{

product: "envelop",

status: "A",

size: { height: 10, weight: 15.25, unit: "cm" },

instock: [ { warehouse: "B", quantity: 15 }, { warehouse: "C", quantity: 35 } ],

creationts: ISODate("2015-11-01T12:30:15Z")

}

])

**Delete by multiple conditions:**

db.deleteplist.deleteMany({

"instock.quantity": 5,

"instock.warehouse": "A",

creationts: { $lt: ISODate("2019-11-01T12:30:15Z") }

})

### ***Task 7: MongoDB Aggregation***

Aggregation operations process data records and return computed results. These operations group values from multiple documents together and can perform operations on the grouped data to return a single result.

**Types of Aggregation in MongoDB**

* Aggregation Pipeline
* Map-Reduce
* Single-purpose Aggregation Methods

**Insert Sample Data**

**Insert sample product data into a collection called aggregateproduct:**

db.aggregateproduct.insertMany([

{ product: "register", quantity: 25, size: { height: 14, weight: 21, unit: "cm" }, status: "A" },

{ product: "book", quantity: 50, size: { height: 8.5, weight: 11, unit: "in" }, status: "A" },

{ product: "sheet", quantity: 100, size: { height: 8.5, weight: 11, unit: "in" }, status: "D" },

{ product: "timetable", quantity: 75, size: { height: 22.85, weight: 30, unit: "cm" }, status: "D" },

{ product: "envelop", quantity: 45, size: { height: 10, weight: 15.25, unit: "cm" }, status: "A" },

{ product: "register", quantity: 25, size: { height: 14, weight: 21, unit: "cm" }, status: "A" },

{ product: "book", quantity: 50, size: { height: 8.5, weight: 11, unit: "in" }, status: "A" },

{ product: "sheet", quantity: 100, size: { height: 8.5, weight: 11, unit: "in" }, status: "D" },

{ product: "timetable", quantity: 75, size: { height: 22.85, weight: 30, unit: "cm" }, status: "D" },

{ product: "envelop", quantity: 45, size: { height: 10, weight: 15.25, unit: "cm" }, status: "A" },

{ product: "register", quantity: 25, size: { height: 14, weight: 21, unit: "cm" }, status: "A" },

{ product: "book", quantity: 50, size: { height: 8.5, weight: 11, unit: "in" }, status: "A" }

]);

**Aggregation Pipeline - $match and $group**

db.aggregateproduct.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$product", total: { $sum: "$quantity" } } }

])

**Explanation:**

* **$match**: Filters documents with status: "A".
* **$group**: Groups by product and sums the quantity.

**Add $sort to the Pipeline**

db.aggregateproduct.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$product", total: { $sum: "$quantity" } } },

{ $sort: { total: 1 } }

])

**Other Aggregation Stages — $addFields**

**Example 1:**

db.aggregateproduct.aggregate([

{ $match: { status: "A" } },

{ $addFields: {

maxWeight: { $max: "$size.weight" },

minSizeHeight: { $min: "$size.height" }

}}

])

**Example 2:**

db.aggregateproduct.aggregate([

{ $match: { status: "A" } },

{ $addFields: {

maxQ: { $max: "$quantity" },

minSizeHeight: { $min: "$size.height" }

}}

])

**Use $project Before Grouping**

db.aggregateproduct.aggregate([

{ $project: { product: 1, quantity: 1 } },

{ $group: { \_id: "$product", total: { $sum: "$quantity" } } },

{ $sort: { total: 1 } }

])

**Example Output:**

[

{ \_id: 'register', total: 75 },

{ \_id: 'envelop', total: 90 },

{ \_id: 'book', total: 150 },

{ \_id: 'timetable', total: 150 },

{ \_id: 'sheet', total: 200 }

]

**Introduction to Map-Reduce**

db.aggregateproduct.mapReduce(

function() {

emit(this.product, 1);

},

function(key, values) {

return Array.sum(values);

},

{

query: { status: "A" },

out: { merge: "map\_reduce\_result" }

}

)

db.map\_reduce\_result.find();

**Note:**

* MapReduce is deprecated.
* Not supported on MongoDB Atlas free or serverless instances.
* Use Aggregation Pipeline instead when possible.

**Aggregation Pipeline vs Map-Reduce**

**Aggregation pipeline is usually preferred over MapReduce due to:**

* Better performance
* Simpler syntax
* Built-in stages for common operations

**Pipeline RAM Limit**

* Aggregation pipelines are limited to **100MB** of RAM.
* Use allowDiskUse: true to enable temporary file storage.

**Single Purpose Aggregation Commands**

**Count documents:**

db.aggregateproduct.count()

**Estimated document count:**

db.aggregateproduct.estimatedDocumentCount()

**Get distinct values:**

db.aggregateproduct.distinct("status")

db.aggregateproduct.distinct("product")