**Question 7 – MongoDB CRUD operations/neo4j Modifying**

1. *Explain the MongoDB CRUD operations to insert, query, update and delete documents.*

**MongoDB CRUD Operations**

Create (Insert)

* Shell/Driver Methods
  + **db.collection.insertOne(doc)** inserts a single document.  
    const result = db.parent.insertOne({ date: new Date() });  
    const id = result.insertedId;
  + **db.collection.insertMany([doc1, doc2, …])** bulk‐inserts multiple documents.
* Common Use in Assignments
  + In Assignment 2: used orders.insertOne(…) to add a new order document to the orders collection.

Read (Query)

* Basic Queries
  + **db.collection.find(filter, projection)** returns a cursor over matching documents.
  + **db.collection.findOne(filter)** returns the first match.
* Aggregation Pipeline
  + For more advanced grouping, filtering and reshaping, you chain stages:  
    db.collection.aggregate([  
     { $match: { status: "shipped" } },  
     { $group: { \_id: "$customer\_id", total: { $sum: "$total\_price" } } }  
    ]};

Update

* Single vs. Multiple
  + db.collection.updateOne(filter, updateDoc) updates the first matching document.
  + **db.collection.updateMany(filter, updateDoc)** updates *all* matching documents.
* Update Operators
  + **$set** replaces or adds specific fields, e.g. **{ $set: { status: "delivered" } }**.
  + **$inc** increments numeric fields, e.g. **{ $inc: { quantity: 1 } }**.
  + Array operators like **$push, $addToSet, $pull** let us modify embedded arrays.
* Example from a Trigger:

posts.updateOne(  
 { number: post\_id },  
 { $inc: { numberOfComments: 1 } }  
);  
posts.updateMany(  
 { "user.username": username },  
 { $inc: { "user.numberOfComments": 1 } }  
);

Delete

* Single vs. Multiple
  + **db.collection.deleteOne(filter)** removes the first matching document.
  + **db.collection.deleteMany(filter)** removes *all* matching documents.

Transactions (Advanced)

* Multi-Document ACID

const session = client.startSession();  
session.startTransaction();  
try {  
 // multiple inserts/updates here  
 await session.commitTransaction();  
} finally {  
 session.endSession();  
}

* Use Case
  + In Assignment 2: wrapped an order‐creation and its related stock update in a transaction to ensure atomicity.

1. *Compare data modification operations in assignment 2 and 3.*

**Comparing Assignment 2 (MongoDB) vs. Assignment 3 (Neo4j)**

**Document (Assignment 2)**

Insert

* **insertOne()** / **insertMany()** into a collection (e.g. **orders.insertOne({...})**)

Read

* **find()** / **findOne()** or **aggregate()** pipelines

Update

* **updateOne()** / **updateMany()** with **$set, $inc,** array operators

Delete

* **deleteOne()** / **deleteMany()**

**Neo4j (Assignment 3)**

Insert

* **CREATE** a node or relationship **MERGE** to idempotently create if absent

Read

* **MATCH (n:Label {…}) RETURN n** (pattern matching)

Update

* **MATCH (n) SET n.prop = value** to update properties **MERGE** can also create or update relationships atomically

Delete

* **MATCH (n)-[r]-() DELETE r** (relationship) **DETACH DELETE n** to remove node and its relationships

**Key Differences**

1. Language & Paradigm
   * MongoDB uses an imperative API (**insertOne, updateMany,** etc.).
   * Neo4j uses *declarative* Cypher: you declare the graph pattern to create, match, update or delete.
2. Transactions
   * In MongoDB you explicitly start/commit a transaction in your driver code.
   * In Neo4j, each Cypher script or transaction block is implicitly ACID; you rarely see explicit transaction commands in simple scripts.
3. Schema & Constraints
   * MongoDB is schemaless, though you *can* enforce JSON Schema on collections.
   * Neo4j allows you to declare uniqueness or existence constraints on node labels and relationship properties.
4. Relationships
   * MongoDB embeds or references documents via ObjectId fields or arrays.
   * Neo4j models relationships as first‐class edges, which you create/update/delete directly in Cypher (**CREATE (a)-[:REL]->(b)**).