**Question 3 – Cypher patterns**

1. *Explain the different forms of patterns in Cypher and exemplify how they are used to query, create, update and delete data.*

**Cypher Patterns: Forms & Use Cases**

Cypher uses ASCII‐art patterns to describe graph structures. The main forms are:

Node Patterns

* Anonymous node: **()**
* Labeled node: **(:Label)**
* Labeled + property: **(:Label {key: value})**
* Variable binding: **(n:Label {key: value})**

Relationship Patterns

* Anonymous relationship: **()-[]-()**
* Typed relationship: **()-[:TYPE]-()**
* Directed:
  + Outgoing: **()-[:TYPE]->()**
  + Incoming: **()<-[:TYPE]-()**

Combined & Variable-Length Patterns

* Chaining: **(a)-[:R]->(b)-[:S]->(c)**
* Variable length:
  + Any depth: **[:REL\*]**
  + Bounded: **[:REL\*1..3]**

Optional Match

* Like a LEFT JOIN: returns nulls when pattern doesn’t exist

Creating Data: CREATE & MERGE

* **CREATE:** always makes new nodes/rels (relationships)
* **MERGE:** “upsert”, only creates when pattern is missing

Updating & Deleting

* **SET**: change properties
* **DELETE** / **DETACH DELETE**: remove relationships or nodes

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

**Comparing Data Modification Document vs. Graph**

**Document (Assignment 2)**

Insert

* **db.orders.insertOne({...})** within an ACID transaction for multi-collection writes

Update

* **db.collection.updateOne( filter, { <operators> } )** e.g. **$set: { title:"New" }, $inc: { "order\_details.0.quantity":1 }, $mul: { price:0.8 }**

Upsert

* **updateOne(filter, {…}, {upsert:true})**

Delete

* **deleteOne** / **deleteMany**, no cascade by default unless via application logic

Transactions

* Multi‐document ACID transactions **(session.startTransaction(), commitTransaction())**

Pattern vs. Separate

* Filters using BSON predicates **({isbn: "123"}**, regex, aggregates with **$lookup)**

Embedded vs. Separate

* Embedded sub-docs (**orders.order\_details**) for atomic writes, references for shared data (authors)

Declarative DSL

* MongoDB’s aggregation pipeline with stages **($match, $group, $project, $lookup**)

**Graph (Assignment 3)**

Insert

* **CREATE (o:Order {…})** and relationship creation in one statement

Update

* **MATCH (n:Label {…}) SET n.prop = expr [, n.other = expr2 …]** – e.g. **SET b.price = b.price\*0.8, SET r.quantity = r.quantity + 1**

Upsert

* **MERGE** to avoid duplicates

Delete

* **DETACH** **DELETE** to remove a node and all its relationships in one go

Transactions

* Neo4j supports multi-statement ACID transactions in enterprise; single‐query atomic writes by default

Pattern vs. Separate

* Graph patterns in **MATCH/OPTIONAL MATCH** for relationship traversal

Embedded vs. Separate

* Everything is nodes + edges
* no embedding, relationships are first‐class

Declarative DSL

* Cypher’s single‐query graph DSL combining pattern matching, projection, and updates

In **Assignment 2**, writes were centered around CRUD on collections, sometimes within explicit transactions to maintain consistency across related inserts/updates.

In **Assignment 3**, each GraphQL mutation translated directly into a Cypher CREATE/MERGE/SET/DELETE pattern (via the @cypher directive), leveraging the graph’s native relationship traversal and atomic single statement guarantees.