NSQ1 S25 Course Assignment 3

Question 1 – Model database

1. Design a graph model for the bookstore model. Make note of the choices you make and why. Document the model using either https://arrows.app/ or screenshots from the browser.

In Assignment3.svg

Graph Model Design Choices:

- **Book-Copy**: Instead of embedding book copies as documents in MongoDB, in the graph model, book copies are nodes to allow better representation of relationships like orders containing book copies.
- **Categories and Genres**: as nodes for more flexibility in querying or visualizing the graph.

2. Enter data according to the model

In graph.cypher

Question 2 – Work with data

Answer the following questions in Neo4j using your model from question 1.

In question2.cypher

Modifying data

Use Cypher statements to execute the following scenarios. Transactions are not supported in the neo4j browser, so try to solve each exercise with a single statement.

If nothing else is stated, assume you know the object ids of the objects involved.

1. Sell a book to a customer.
2. Change the address of a customer.
3. Add an existing author to a book.
4. Retire the "Space Opera" category and assign all books from that category to the parent category. Don't assume you know the id of the parent category.
5. Sell 3 copies of one book and 2 of another in a single order

Querying data

Write Cypher queries to return the following data

1. All books by an author

2. Total price of an order
3. Total sales (in £) to a customer
4. Books that are categorized as neither science fiction nor fantasy
5. Average page count by genre
6. Categories that have no sub-categories
7. ISBN numbers of books with more than one author
8. ISBN numbers of books that sold at least X copies (you decide the value for X)
9. Number of copies of each book sold – unsold books should show as 0 sold copies.
10. Best-selling books: The top 10 selling books ordered in descending order by number of sales.
11. Best-selling genres: The top 3 selling genres ordered in descending order by number of sales.
12. All science fiction books. Note: Books in science fiction subcategories like cyberpunk also count as science fiction.
13. Characters used in science fiction books. Note from (12) applies here as well.
14. Number of books in each category including books in subcategories.

Question 3 – GraphQL

Note: It's difficult to come up with queries and mutations that aren't possible to solve somehow with standard queries and mutations in the Neo4j driver for Apollo Server, but create your own for the exercise.

In schemaDefinition.graphql

1. Create a schema definition in GraphQL Definition Language for your data model
2. Add relationships (@relationship) to connect it to the data model
3. Define queries in the Query type:
   - Given a search term, all books that have the search term as part of the title *or* has the search term as part of the author names
   - Given the email of a customer and the name of a genre, all orders from the customer that contain any books from the genre
   - Given the name of a category, all books in the category, its subcategories and their subcategories and so on
4. Define mutations in the Mutation type:
   - Create an order for a single book and a customer (1-click)
   - Create an order for many books and one customer
   - Apply x% reduction on all books by an author

Question 4 – Report

Write a report on the experience gained by completing the questions above. The report should contain answers to the questions

- What were the decisions taken in the modelling?

1. **Separation Book and BookCopy Nodes**
    - Each physical/digital copy of a book has been modelled as its own BookCopy node rather than embedding copies within the Book node. This makes it less relevant to show "an order contains n copies of a specific copy" and to evolve copy-level properties (e.g., price, format, stock) independently of the book's data.
2. **Categories and Subcategories as Nodes**
    - Each genre has been represented as a Category node, hierarchies between genres have been defined by SUBCATEGORY_OF relationships. Thanks to this, we can traverse the graph recursively without requiring complex operations.
3. **Authors, Characters, Customers, Orders as First-Class Nodes**
    - We chose to represent all entities as their own nodes (Author, Order, etc.) and establish explicit relationships of these nodes with Books and BookCopies. This allowed us to easily and directly manipulate relationships between domain elements, for example, assign multiple authors to a book without complex code.
4. **Use of Unique Constraints and IDs**
    - For integrity and identity, @id and @unique constraints have been added on Book.isbn and Customer.email. Each node has also been given an auto-generated UUID. This prevents duplicate nodes and simplifies lookup by stable keys.

- Why were these decisions taken?
    - **Flexibility**: Nodes for everything let us add new properties or relationships without altering a monolithic document/document-like structure.
    - **Query Simplicity**: Recursion and pattern-matching in Cypher become straightforward (e.g. MATCH (:Category {name:$cat})<-[:SUBCATEGORY_OF*0..]-(sub)).
    - **Separation of Concerns**: Keeping book data distinct from transactional data (BookCopy, Order) cleanly separates read from write concerns.

- What were the consequences of these decisions?
    - **Schema Complexity**: More node types and relationships than a document or relational model, which requires careful SDL maintenance.
    - **Performance**: Graph traversals are highly efficient for deep joins but less so for wide aggregations.
    - **Operational Overhead**: Managing unique constraints, migrations, and bulk imports across multiple node types can be more complex than in a single-table or single-collection system.

- What were the difficult and easy parts of the exercise?
    - **Difficult**
        - Crafting @cypher directives with the correct columnName and expressions (e.g. converting Neo4j's Date object into a GraphQL Date).
        - Correctly modelling @relationshipProperties (the ContainsRel interface) to store per-order line item quantities and subtotals.
        - Bulk population scripts in pure Cypher required careful use of separate MATCH + multi-create patterns, especially when referencing nodes created in the same clause.
    - **Easy**
        - Defining simple one-to-many relationships (e.g. Author → Book).
        - Writing GraphQL SDL with @relationship on non-recursive edges.
        - Basic CRUD operations once the model was in place (e.g. 1-click order for a single book).
- How does that compare to the other exercises?
    - **Relational Modeling**: SQL joins are like graph traversals, but handling hierarchies or paths of unknown length (such as subcategories) in SQL needs recursive CTEs, which are more verbose than using `*0..` in Cypher.
    - **Document Modeling (MongoDB)**: Embedding arrays inside documents is simple, but updating nested subdocuments (e.g. updating one copy price) is more cumbersome than updating a separate BookCopy node in a graph.
    - **Key-Value Stores**: Too simplistic; they can't easily express complex relationships without additional lookup tables or layers.

- What are the advantages and disadvantages of graph databases compared to the other database types?

Advantages	Disadvantages
Good Relationship Support: • Traverse arbitrary depth in a few characters of Cypher. • Fewer enterprise-grade tools than for SQL or MongoDB.	Solution Maturity: • Most graph databases have only matured in the last 5–10 years, so it's still harder to find solutions for things like zero-downtime upgrades, backups, replication, and enterprise-level monitoring.
Flexible Schema: • Easy to add new node/relationship types.	Learning Curve: • Developers have to learn a new way of modeling data using nodes and relationships instead of tables or JSON.
Performance on Connected Queries: • Constant-time per-hop traversal for deeply connected data.	Bad Scaling: • Splitting a highly connected graph across multiple machines without cutting critical relationships is challenging. Even small cross-partition traversals can become very expensive.
Natural Domain Modeling: • Fits domains with many real-world relationships (social media, recommendations).	Less Mapping Tools: • Compared to SQL or documents, there are fewer ORMs or graph-aware mappers.

Rules

- Make the exercise in groups of 2 – 4