No-SQL versus relational databases

Course Assignment 3

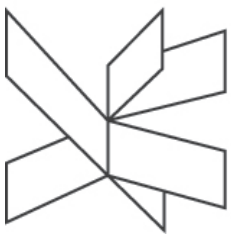
Jordi Rafael Lazo Florensa

9th May 2022

Software Technology Engineering

Teacher: Ole Ildsgaard HougaardSend

IT-NSQ1-S22



VIA University College

1 Question 1 – Model database

The graph model that I have created for the book store is the following:

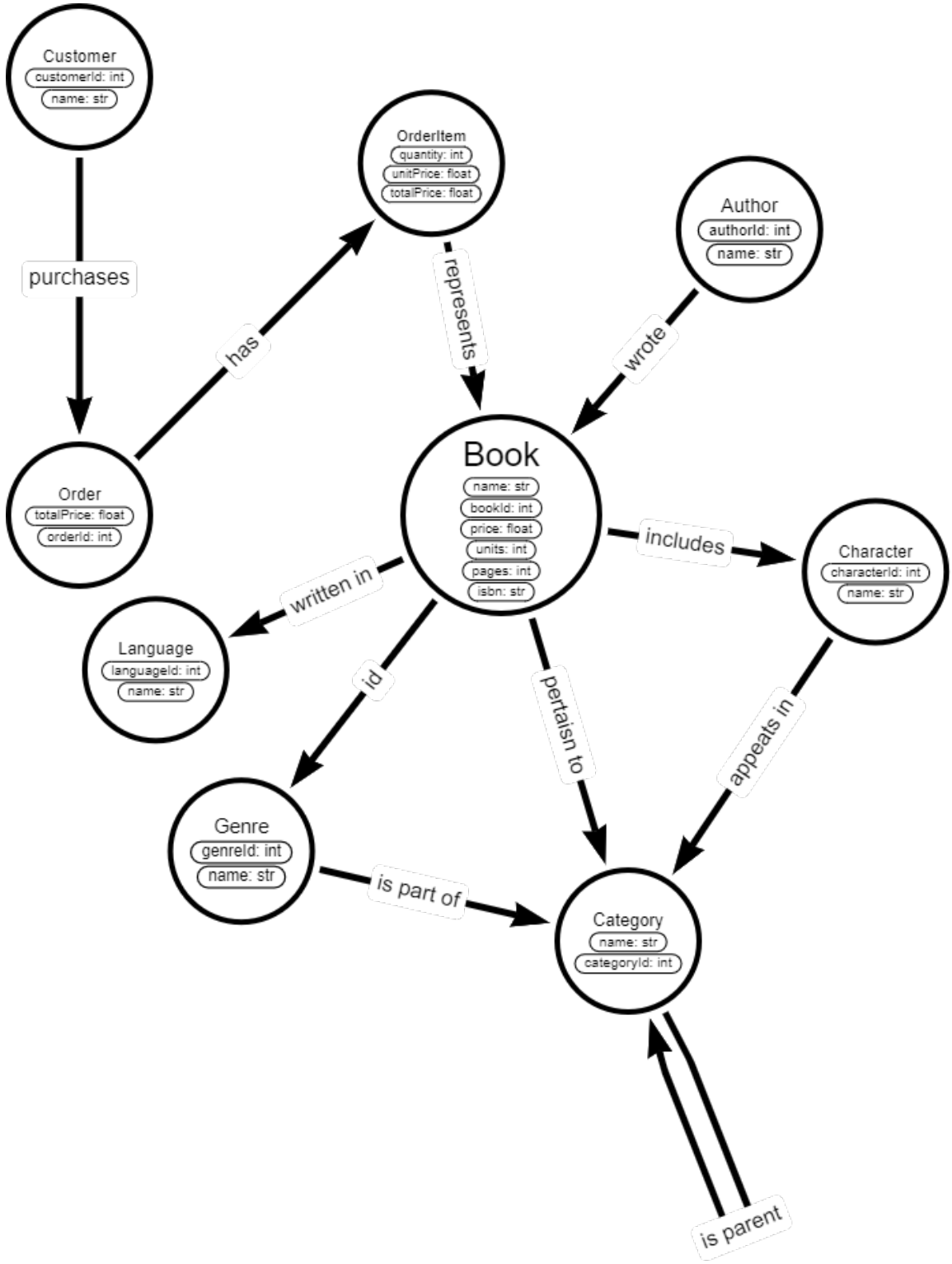


Figure 1: Graph database model.

To insert the data in the database, the following command lines have been written in the *neo4j* console:

```
WITH "https://gist.githubusercontent.com/santo0
/e4ba9939e5146c5fe9c75d967f4fcb73/raw
/98359e7b7f447b2615dcc6e0e8d9137221804f15/books.json" AS url
CALL apoc.load.json(url) YIELD value
UNWIND value as book
CREATE (b:Book{
    name:book.name,
    price:book.price,
    units:book.units,
    pages:book.pages,
    isbn:book.isbn})
FOREACH (category IN book.categories |
    MERGE (c:Category {
        name:category.name
    })
    MERGE (b) -[:PERTAINS_TO]->(c)
)
FOREACH (language IN book.languages |
    MERGE (l:Language {
        name:language.name
    })
    MERGE (b) -[:WRITTEN_IN]->(l)
)
FOREACH (character IN book.characters |
    MERGE (c:Character {
        name:character.name
    })
    MERGE (b) -[:INCLUDES]->(c)
)
FOREACH (genre IN book.genres |
    MERGE (g:Genre {
        name:genre.name
    })
    MERGE (b) -[:IS]->(g)
)
FOREACH (author IN book.authors |
    MERGE (a:Author {
        name:author.name
    })
    MERGE (b) <-[:WROTE]-(a)
)


WITH "https://gist.githubusercontent.com/santo0
/3ff6124dcb2cbd1d82156555022a3fbf/raw
/96cb689a9ab923e7dc0cb13386ca28fdd18e788b/categories.json" AS url
CALL apoc.load.json(url) YIELD value
UNWIND value as category
MERGE (c1:Category{
    name:category.name
```

```
})


WITH category, c1
WHERE NOT category.parentCategory IS NULL
MERGE (c2:Category{
    name:category.parentCategory
})
MERGE (c2) -[:IS_PARENT]-> (c1)


WITH "https://gist.githubusercontent.com/santo0
/b75c33cd922ee81477c37d6249074f51/raw
/8f84fef4d695da00ac7ea98985481d6bac03cf6a/characters.json" AS url
CALL apoc.load.json(url) YIELD value
UNWIND value as character
MERGE (c:Character{
    name:character.name
})
MERGE (ca:Category{
    name:character.category
})
MERGE (c)-[:APPEARS_IN]->(ca)


WITH "https://gist.githubusercontent.com/santo0
/afeec3df662d3317fcc28b8a53baeaaa/raw
/06e96b76cf3a70e3c6b34e0ac6742dfbf9d29c0c/genres.json" AS url
CALL apoc.load.json(url) YIELD value
UNWIND value as genre
MERGE (g:Genre{
    name:genre.name
})
MERGE (ca:Category{
    name:genre.category
})
MERGE (g)-[:IS_PART_OF]->(ca)

WITH "https://gist.githubusercontent.com/santo0
/9836ac6497a92f1734db1f92c049ab53/raw
/8e11f93bbfef0623fc5915188bcc8cd149bb8171/authors.json" AS url
CALL apoc.load.json(url) YIELD value
UNWIND value as author
MERGE (a:Author{
    name:author.name
})

WITH "https://gist.githubusercontent.com/santo0
/60eb770cda64a516bd588f5618556fa1/raw
/80a09b8cbceded380d356fbb6ec0b1c0bd406024/languages.json" AS url
CALL apoc.load.json(url) YIELD value
```

```
UNWIND value as language
MERGE (c:Language{
    name:language.name
})

WITH "https://gist.githubusercontent.com/santo0
/0b5917ef542bd1bd792dfc02ee0e3020/raw
/d0b2163361dee94b073c28e2075d6c2db1ffb328/customers.json" AS url
CALL apoc.load.json(url) YIELD value
UNWIND value AS customer
CREATE (c:Customer{
    name: customer.name})
FOREACH (order IN customer.orders |
MERGE (c) -[:PURCHASES]-> (o:Order {
    orderId:order.orderId,
    totalPrice:order.totalPrice})
    FOREACH (orderItem IN order.orderItems |
    MERGE(oi:OrderItem {
        name: orderItem.name,
        quantity:orderItem.quantity,
        unitPrice:orderItem.unitPrice,
        totalPrice:orderItem.totalPrice})
    MERGE (oi) <-[:HAS]- (o)
    MERGE (b:Book{name:orderItem.name})
    MERGE (b) <-[:REPRESENTS]- (oi)
    ))
```

2 Question 2 – Work with data

2.1 Modifying data

1. Sell a book to a customer.

```
        CREATE (oi:OrderItem{unitPrice:9.99, quantity:1, totalPrice:9.99,
            name:"1984"})
        CREATE (o:Order{totalPrice:19.98, orderId:5})
        WITH oi, o
        MATCH (c:Customer{name:"John"})
        CREATE (o)-[:HAS]->(oi)
        CREATE (c) -[:PURCHASES]->(o)
        WITH *
        MATCH (b:Book{name:"1984"})
        SET b.units = b.units-1
```

2. Change the address of a customer.

```
        MATCH(c:Customer{name:{"Pep"})
        SET c.address = "Lleida, Lleida"
```

3. Add an existing author to a book.

```
MATCH (a:Author{name:"Stephen King"})
MATCH (b:Book{name:"Dune"})
MERGE (a)-[:WROTE]->(b)
```

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.

```
MATCH
    (catParent:Category)-[:IS_PARENT]->(cat:Category{name:"Space
    Opera"})
OPTIONAL MATCH (cat)-[:IS_PARENT]->(catChild:Category)
WITH catParent, cat, catChild
CREATE (catParent)-[:IS_PARENT]->(catChild)
DETACH DELETE cat
```

5. Sell 3 copies of one book and 2 of another in a single order.

```
CREATE (oi1:OrderItem{unitPrice:9.99, quantity:2,
    totalPrice:19.98, name:"It"})
CREATE (oi2: OrderItem{unitPrice:9.99, quantity:3,
    totalPrice:29.97, name:"Maus"})
CREATE (o:Order{totalPrice:49.95, orderId:6})
WITH oi1, oi2, o
MATCH (c:Customer{name:"Pep"})
CREATE (o)-[:HAS]->(oi1)
CREATE (o)-[:HAS]->(oi2)
CREATE (c) -[:PURCHASES]->(o)
WITH *
MATCH (b1:Book{name:"It"})
SET b1.units = b1.units-2
WITH *
MATCH (b2:Book{name:"Maus"})
SET b2.units = b2.units-3
```

2.2 Querying data

1. All books by an author.

```
MATCH(:Author {name:"Stephen King"})-[:WROTE]->(book:Book)
RETURN book
```

2. Total price of an order.

```
MATCH(o:Order {orderId:1})
RETURN o.totalPrice
```

3. Total sales (in £) to a customer.

```
MATCH(c:Customer{name:"Pep"})-[:PURCHASES]->(o:Order)
RETURN sum(o.totalPrice)
```

4. Books that are categorized as neither fiction nor non-fiction.

```
MATCH(b:Book)-[PERTAINS_TO]->(c:Category)
WHERE NOT c.name="Fiction" AND NOT c.name="Biography"
RETURN b
```

5. Average page count by genre.

```
MATCH (b:Book)-[:IS]->(g:Genre)
RETURN avg(b.pages),g
```

6. Categories that have no sub-categories.

```
MATCH (c:Category)
WHERE NOT (c)-[:IS_PARENT]->(:Category)
RETURN c
```

7. ISBN numbers of books with more than one author.

```
MATCH (a:Author) -[:WROTE]-> (b:Book)
WITH count(a) AS numAuthors,b
WHERE numAuthors >= 2
RETURN b.isbn
```

8. ISBN numbers of books that sold at least X copies (you decide the value for X).

```
MATCH (oi: OrderItem) -[:REPRESENTS]-> (b:Book)
WITH sum(oi.quantity) as qty, b
WHERE qty >= 2
RETURN b.isbn
```

9. Number of copies of each book sold – unsold books should show as 0 sold copies.

```
MATCH (b:Book)
OPTIONAL MATCH (oi: OrderItem) -[:REPRESENTS]-> (b)
WITH sum(oi.quantity) as quantity, b.name as bookName
RETURN bookName, quantity
```

10. Best-selling books: The top 10 selling books ordered in descending order by number of sales.

```
MATCH (b:Book)
OPTIONAL MATCH (oi: OrderItem) -[:REPRESENTS]-> (b)
WITH sum(oi.quantity) as quantity, b.name as bookName
RETURN bookName, quantity
ORDER BY quantity DESC
LIMIT 10
```

11. Best-selling genres: The top 3 selling genres ordered in descending order by number of sales.

```
MATCH (b:Book) -[:IS]-> (g:Genre)
OPTIONAL MATCH (oi: OrderItem) -[:REPRESENTS]-> (b)
WITH sum(oi.quantity) as quantity, g.name as genreName
RETURN genreName, quantity
ORDER BY quantity DESC
LIMIT 3
```

12. All science fiction books. Note: Books in science fiction subcategories like cyberpunk also count as science fiction. Don't use your knowledge of the concrete category structure.

```
MATCH (catParent:Category{name:"Science Fiction & Fantasy"})
    -[:IS_PARENT * 1..4]-> (c:Category)
OPTIONAL MATCH (b1:Book) -[:PERTAINS_TO]-> (c:Category)
OPTIONAL MATCH (b2:Book) -[:PERTAINS_TO]-> (catParent:Category)
WITH coalesce(b1.name, []) + coalesce(b2.name, []) as bookNames
RETURN bookNames
```

13. Characters used in science fiction books. Note from (12) applies here as well.

```
MATCH (catParent:Category{name:"Science Fiction & Fantasy"})
    -[:IS_PARENT * 1..4]-> (c:Category)
OPTIONAL MATCH (b1:Book) -[:PERTAINS_TO]-> (c:Category)
OPTIONAL MATCH (b2:Book) -[:PERTAINS_TO]-> (catParent:Category)
WITH apoc.coll.toSet(collect(coalesce(b1.name, [])) +
    collect(coalesce(b2.name, []))) as bookNames
UNWIND bookNames as bookName
MATCH (:Book{name:bookName})-[:INCLUDES]-> (c:Character)
RETURN c.name
```

14. Number of books in each category including books in subcategories.

```
MATCH (catParent:Category{name:"Science Fiction & Fantasy"})
    -[:IS_PARENT * 1..4]-> (c:Category)
WITH apoc.coll.toSet(collect(coalesce(catParent.name, [])) +
    collect(coalesce(c.name, []))) as catNames
UNWIND catNames AS catName
MATCH (cat:Category{name:catName})
OPTIONAL MATCH (b:Book) -[:PERTAINS_TO]->(cat)
WITH count(b) as numBook, cat
RETURN cat.name, numBook
```

3 Question 3 – Graph Data Science

3.1 Similar Customers

```
CALL gds.graph.create.cypher('customer-book-purchase',
"MATCH (n)
 WHERE n:Customer or n:Book
 RETURN id(n) AS id, labels(n) AS labels",
"
  MATCH (c:Customer)-[p:PURCHASES]->(o:Order)-[h:HAS]->
  (oi:OrderItem)-[r:REPRESENTS]->(b:Book)
  RETURN id(c) AS source, id(b) AS target, 'HAS_PURCHASED_BOOK' as type
")
YIELD graphName AS graph, nodeQuery, nodeCount
AS nodes, relationshipQuery, relationshipCount AS rels

CALL gds.louvain.stream('customer-book-purchase')
YIELD nodeId, communityId, intermediateCommunityIds
RETURN gds.util.asNode(nodeId).name
AS name, communityId, intermediateCommunityIds
ORDER BY name ASC
```

3.2 Key Customers

```
CALL gds.graph.create.cypher('order-purchased-by-customer',
"MATCH (n)
 WHERE n:Customer or n:Order
 RETURN id(n) AS id, labels(n) AS labels",

"
  MATCH (c:Customer)-[p:PURCHASES]->(o:Order)
  RETURN id(o) AS source, id(c) AS target, 'PURCHASED_BY' as type
")
YIELD graphName AS graph, nodeQuery, nodeCount
AS nodes, relationshipQuery, relationshipCount AS rels

CALL gds.articleRank.stream('order-purchased-by-customer')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC, name ASC
```

3.3 Book Suggestions

```
MATCH (c:Customer {name: 'Miquel'})
MATCH (b:Book)
RETURN gds.alpha.linkprediction.adamicAdar(c, b,
{relationshipQuery:
  "
  MATCH (c:Customer)-[p:PURCHASES]->(o:Order)-[h:HAS]->
  (oi:OrderItem)-[r:REPRESENTS]->(b:Book)
  RETURN id(c) AS source, id(b) AS target, 'HAS_PURCHASED_BOOK' as type
"
}) AS score, b.name as book
```

4 Question 4 – Report

- What were the decisions taken in the modelling?
  I had to decide on the nodes and, more crucially, the relationships between the nodes throughout the modeling process, based on both the notion I had for the amazon database and a viable model for querying later on. The relationships are all one-way. A consumer makes a purchase that includes orderItems. These orderItems indicate the Order's purchased Books. These books are written in specific languages and belong to specific categories and genres. Characters are also included. Finally, a book was written by one or more authors. That is the model's overall concept. I decided to reduced a number of the properties we had on previous models because we didn't really need them.

- Why were these decisions taken?
  This model is constructed in a way that I believe is very intuitive for everyone to comprehend; it logically expresses the notion behind every operation in the bookshop, and it is straightforward and intelligible while updating and querying.

- What were the consequences of these decisions?
  The queries are easy and understandable as a result of the model's intuitive approach. Because the direction of the category relation has changed from previous models, it now points to its child instead of its parent, the queries have been adjusted accordingly.

- What were the difficult and easy parts of the exercise?
  Overall, I thought the data science question was the most perplexing. Overall, graphs, neo4j, and cypher were all fascinating to me, therefore questions 1 and 2 were not too difficult for me.

- How does that compare to the other exercises?
  I believe this was the most enjoyable project. The technology is easy to use and understand.

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

  - The structures are agile and flexible.
  - The representation of relationships between entities is explicit.
  - Queries output real-time results. The speed depends on the number of relationships.

  Disdvantatges:

  - There is no standardized query language. The language depends on the platform used.
  - Graphs are inappropriate for transactional-based systems.
  - The user-base is small, making it hard to find support when running into a problem.