Building a Knowledge Graph Enriched With Large Language Models

Oğuzhan Güngör 24411006

Goal of the Presentation

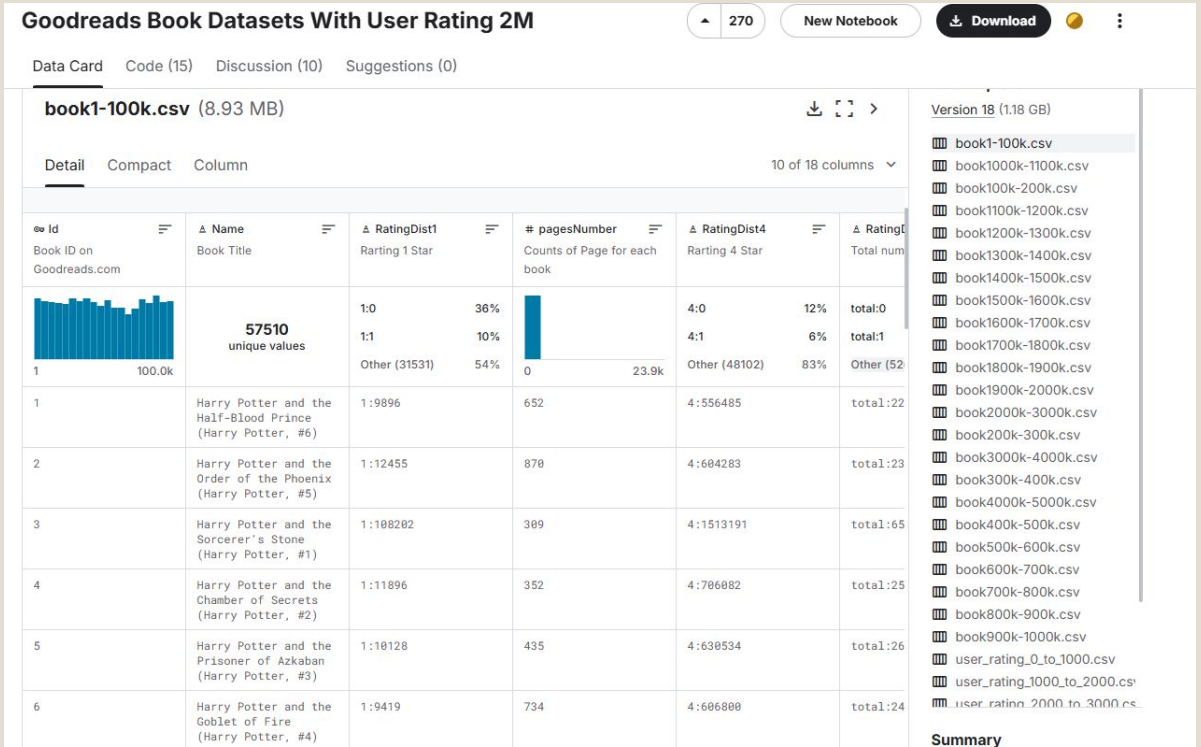
To explain the current state of the project and present the deliverables.

The project is creating a knowledge graph and enhance the Large Language Model (LLM).

Using the knowledge graph to enrich the LLM and link it to DBpedia for enhanced semantic connections and information retrieval.

Data Preparation

In this project, Goodreads Book Dataset has been used for generating the initial database.



Data Pipeline Methods

There were 3 approaches when storing data from csv file to Neo4j graph database:

1) Utilizing Concurrency in Python

2) Utilizing Multiprocessing in Python

3) Utilizing Concurrency at Database Level

Data Pipeline Methods Pros and Cons

Concurrency in Python: It is fast. Comes with race conditions, managing connection pools is hard. Lacks data integrity.

Multiprocessing in Python: Easier solution. It is slow. Don't need to manage connection pools. Data integrity is full.

Concurrency in Neo4j using APOC: It is fastest. Comes with race conditions therefore comes with failed insertions. Lacks data integrity. This method was picked due to its accuracy.

Data Pipeline Methods- APOC

```
CALL apoc.periodic.iterate(
  'LOAD CSV WITH HEADERS FROM "file:///18/book100k-200k.csv" AS row RETURN row',
  '
  CALL apoc.do.when(
    row.Id IS NULL OR row.Authors IS NULL,
    "RETURN null",
    "
    MERGE (b:Book {id: toInteger(row.Id)})
    SET b.name = row.Name, b.language = row.Language, b.publisher = row.Publisher, b.publishMonth = row.PublishMonth, b.rating = row.Rating, b.ISBN = row.ISBN
    MERGE (a:Author {name: row.Authors})
    MERGE (b)-[:WRITTEN_BY]→(a)
    ", {row: row})
```

Table | Text | Code

	batches	failedBatches	total	timeTaken	committedOperations
1	115	23	57046	420	45546

Started streaming 1 records after 1 ms and completed after 420491 ms.

Environment Generation

For the development environment, Docker container has been utilizied for easier deployment of Neo4j. For Python scripts, Python 3.12 has been picked due to speed benefits of Python 3.12.

For version control, Git and Github has been utilized. Since the project is a solo project with time constraints, no branch controls have been implemented.

Graph Version 1:
Generated With Only The Dataset

This version made with only using the contents of the data. In this version, graph database have meaningful connections but lacks the depth.

Nodes:
Books, Authors, Users

Edges:
WRITTEN_BY: Book -> Author
REVIEWED_BY: Book -> User

Graph Version 1:
Generated With Only The Dataset

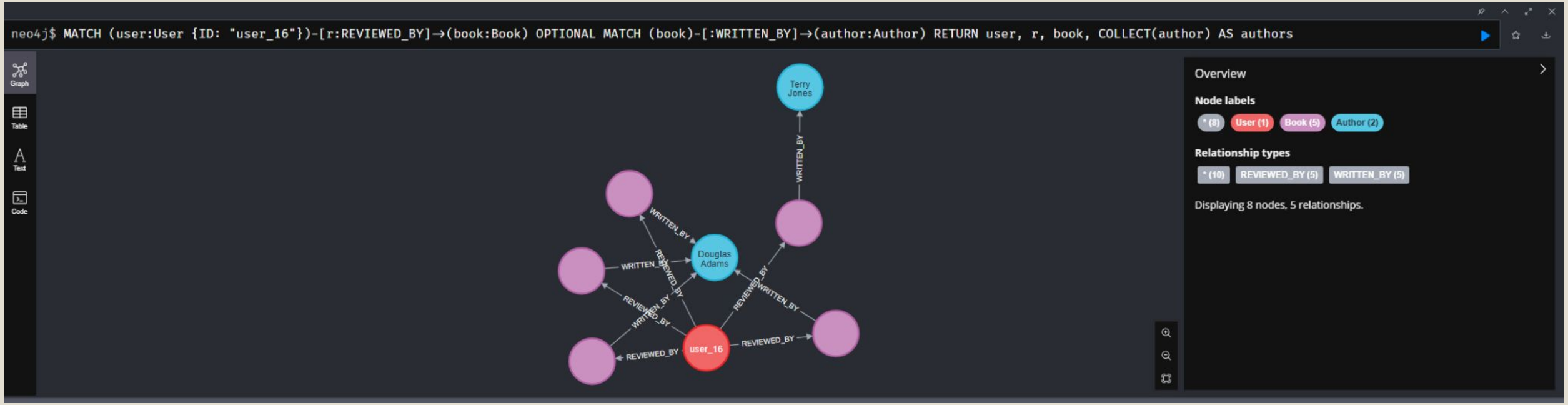
```
1 MATCH (user:User {ID: 'user_1'})-[r:REVIEWED_BY]→(book:Book)
2 RETURN user.ID AS userId, book.Name AS bookName, r.rating AS rating
3
```

Table | Text | Code

	userId	bookName	rating
1	"user_1"	"The Golden Compass (His Dark Materials, #1)"	"it was amazing"
2	"user_1"	"The Scarlet Letter"	"liked it"
3	"user_1"	"The Scarlet Letter"	"liked it"
4	"user_1"	"Frankenstein"	"really liked it"
5	"user_1"	"Street Boys"	"really liked it"
6	"user_1"	"Where the Red Fern Grows"	"it was amazing"
7			

Started streaming 636 records in less than 1 ms and completed after 3 ms.

Graph Version 1:
Generated With Only The Dataset



Graph Version 1:
Generated With Only The Dataset

DBpedia Integration

Since the data for books are lacking Genre of the books, the awards that author or the book got using DBpedia to obtain genres of the books was the solution for improving the graph database.

Graph Version 11:
Improved Version With DBpedia

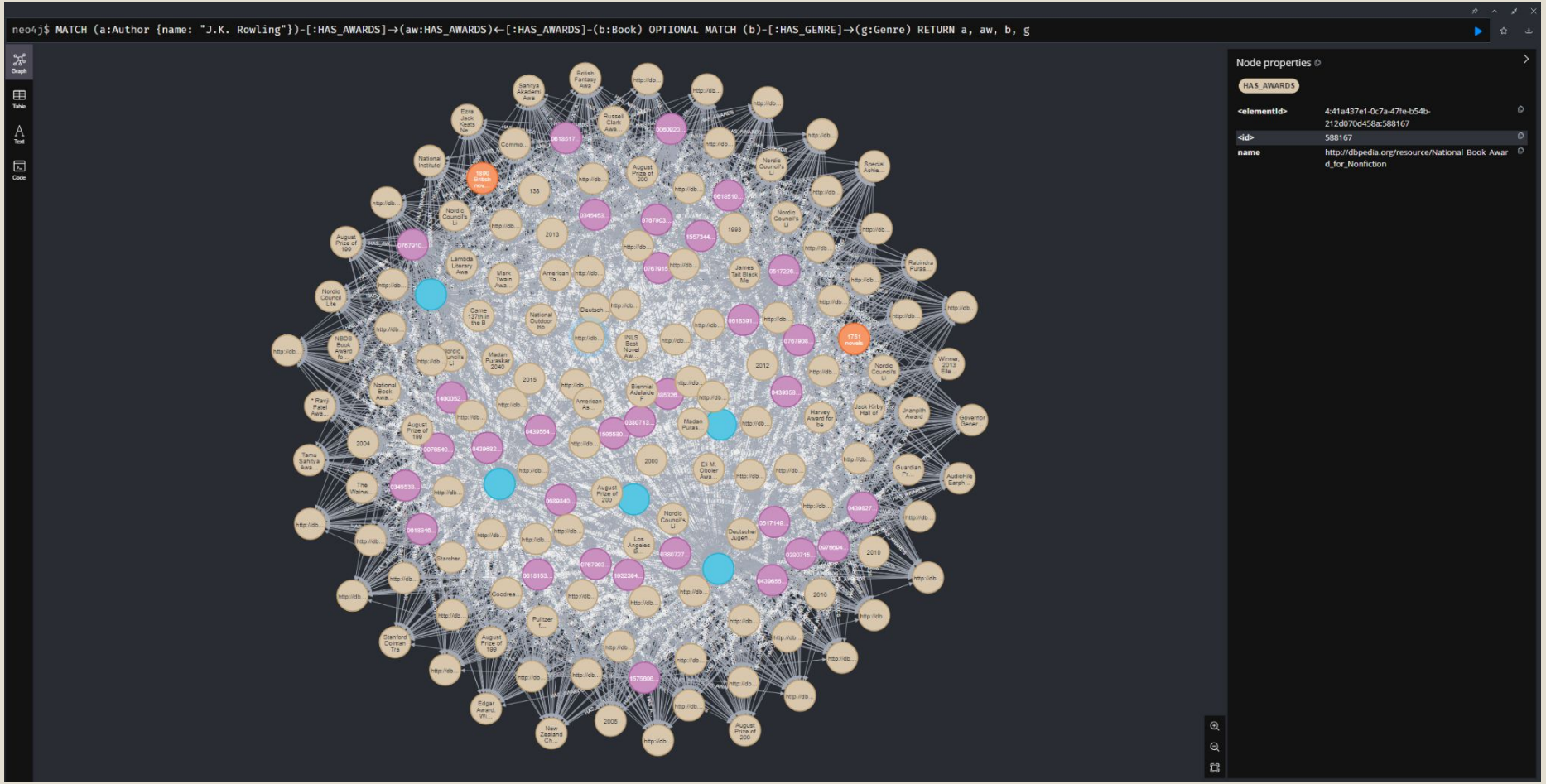
This version made with addition to DBpedia data. In this version, graph database have meaningful connections but still lacks the depth.

Improvements:
Added Genre, Award, Descriptions

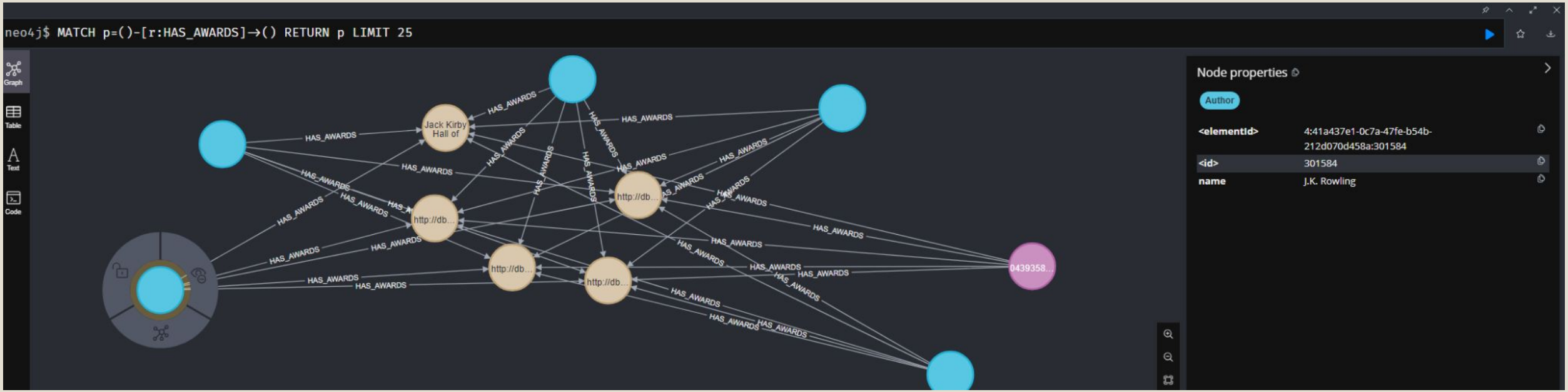
Nodes:
Books, Authors, Users, Genre, Award

Edges:
WRITTEN_BY: Book -> Author
REVIEWED_BY: Book -> User
HAS_GENRE: Book -> Genre
HAS_AWARDS: Book -> Award

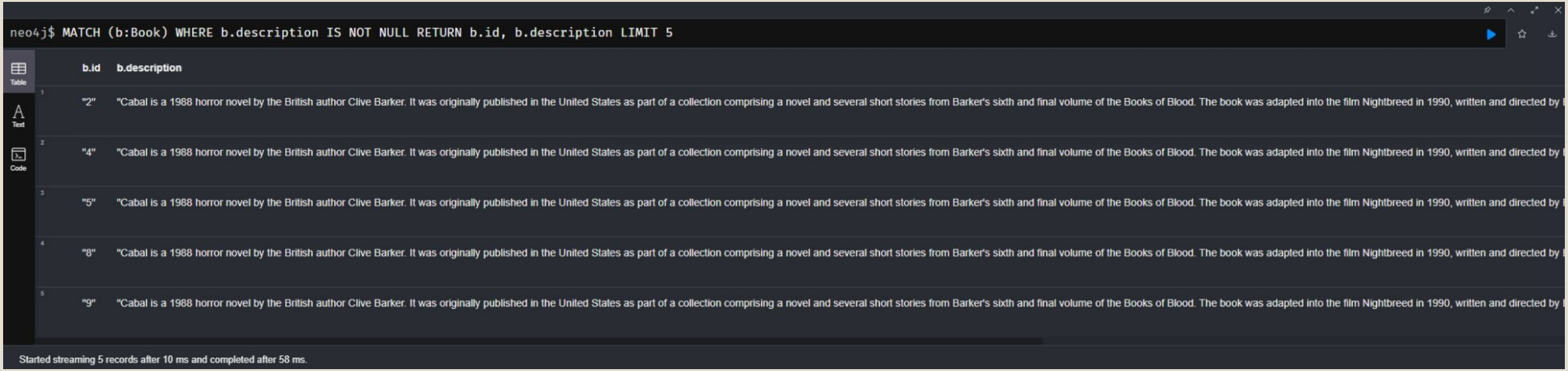
Graph Version 11:
Improved Version With DBpedia



Graph Version 11:
Improved Version With DBpedia



Graph Version 11:
Improved Version With DBpedia



LLM Integration

Since the data for books are lacking semantic relationships with comments and other books, utilizing LLM to improve the graph database was crucial.

This step is work in progress at the moment. This step was missing the crucial connection between similar books.

In this step, sentiment analysis was performed using multiple LLMs, mainly BART and OpenAI.

Graph Version III:
Improved Version With LLM Integration
BART Only

```
You: What's the sentiment for "Gatsby"
INFO:__main__:Running query:

    MATCH (u:User)-[r:REVIEWED_BY]->(b:Book)
    WHERE toLower(b.name) CONTAINS toLower($name)
    RETURN u.name AS userName, r.review AS review

PARAMS: { name: 'Gatsby' }
Chatbot:
Reviews for any book name containing 'Gatsby':
  - User: user_128
    Review: "liked it"
    BART Sentiment => POSITIVE
  - User: user_128
    Review: "it was amazing"
    BART Sentiment => POSITIVE
  - User: user_128
    Review: "really liked it"
    BART Sentiment => POSITIVE
```

Graph Version III:
Improved Version With LLM Integration
BART and OpenAI

```
Chatbot:
Reviews for any book name containing 'Gatsby':
  - User ID 4:41a437e1-0c7a-47fe-b54b-212d070d458a:585786:
    Review: "liked it"
    [OpenAI] => POSITIVE (4 stars)
    [BART]   => POSITIVE (4 stars)
  - User ID 4:41a437e1-0c7a-47fe-b54b-212d070d458a:585786:
    Review: "it was amazing"
    [OpenAI] => POSITIVE (5 stars)
    [BART]   => POSITIVE (4 stars)
  - User ID 4:41a437e1-0c7a-47fe-b54b-212d070d458a:585786:
    Review: "really liked it"
    [OpenAI] => POSITIVE (4 stars)
    [BART]   => POSITIVE (2 stars)
```

References

1) Goodreads Book Datasets With User Rating 2M
2)

Thanks for listening