GameGraph: Exploring Video Game Data with Neo4j and Flexible Data Serialization

Ole J. Berg
ole_julius.berg@smail.th-koeln.de
Technische Hochschule Köln

Lennard Feuerbach
lennard.feuerbach@smail.th-koeln.de
Technische Hochschule Köln

14/01/2025

Agenda

- Introduction
- Technical Basics
- Project Steps
- Live Demo
- Reflection
- References

Introduction

- The project focuses on visualizing video game and related data
- Provide an approach for querying the data efficiently and flexibly
  - Create a simple visual interface for easily querying the data
- The results of those queries should be conform to a defined ontology

Technical Basics

- The data source for the whole project is MobyGames
- First the data is stored in a Neon relational database
- Afterwards moved the data to a Neo4j graph database
- Neo4j was setup on an Ubuntu server hosted by DigitalOcean
- Further tools: Neo4j Desktop, Python, JS, HTML, GitHub, Quarto …

Project Steps

Step 1: Acquiring the Data

- *MobyGames* is data source
- Provide the possibility of retrieving data via `API`
- Downloaded the top 2,500 games based on the *MobyGames* rating
- Got further data on the games, genres and genre types by `API`
- Company-related data was extracted from the `HTML`

Step 2: Saving in Relational DB

- Principle: Save all the data in a relational database (*Neon*) first
- API limitation: Save API responses as `jsonb` and do processing later
- The data and tables were mostly normalized to reduce redundancy

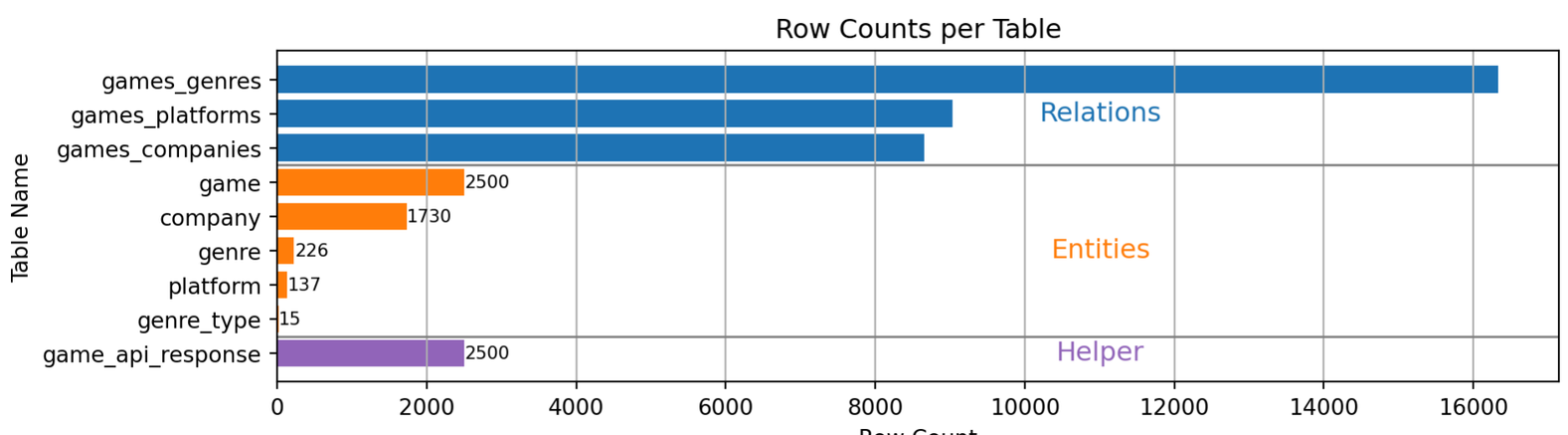

Figure 1: The number of entities and relations extracted from MobyGames

Step 3: From Relational to Graph

- Data is now moved from the relational database to a graph database
- For plugins reasons: Deployment of *Neo4j* server on Ubuntu server
- Installation of `neosemantics` and `APOC`, for *Schema.org* and *Cypher* support
- The basic tables get transformed into nodes with their attributes
- The foreign keys in the relational database get transformed to relationships

Step 4: Saving in Graph DB

- Extracted data from the relational database using `psycopg2` package
- Saved the extracted data in the graph database using `neo4j` package

```python
# Tables from the Relational Database [Table, [Columns], Label]
TABLES = [
    ("company", ["company_id", "company_name"], "Company"),
    ("game", ["game_id", "title", "score", "release_date"], "Game"),
    ("genre", ["genre_id", "name"], "Genre"),
    ("genre_type", ["genre_type_id", "name"], "GenreType"),
    ("platform", ["platform_id", "name"], "Platform")
]

[...]

def fetch_data_from_neon(table_name, columns):
    neon_connection = psycopg2.connect(
        database=os.getenv("DB_NAME"),
        user=os.getenv("DB_USER"),
        password=os.getenv("DB_PASSWORD"),
```

Step 5: Mapping Schema.org

- For exporting onthology conforming data, mapping can be used in Neo4j
- Realized by the `neosemantics` plugin, *Schema.org* can be incorporated [2]
- First create a namespace using:
  - `CALL n10s.nsprefixes.add("sch","http://schema.org/");`
- Then create mappings between attribute and types, e. g.:
  - `CALL n10s.mapping.add("http://schema.org/VideoGame", "Game");`
- For attributes which have no fitting type, own ones were created

Live Demo

Reflection

- Starting with data in a relational database caused several follow-up issues
- These issues complicated transfer to graph database and ontology mapping
  - e.g. normalizing data or creating tables for each category
- Manual `JSON` manipulation was required to ensure *Schema.org* compliance
- Significant time and effort were spent fixing these issues afterwards
- Future projects should prioritize graph database design from the start

References

- [1] GitHub, “Project in WS 24/25 for the module Linked Open Data and Knowledge Graphs in the Master Digital Sciences” Accessed: Jan. 13, 2025. [Online]. Available: https://github.com/ole-berg/DS_LOD_and_Knowledge_Graphs_2024_Berg_Feuerbach
- [2] Neo4j, “Mapping graph models”. Accessed: Jan. 13, 2025. [Online]. Available: https://neo4j.com/labs/neosemantics/4.0/mapping/