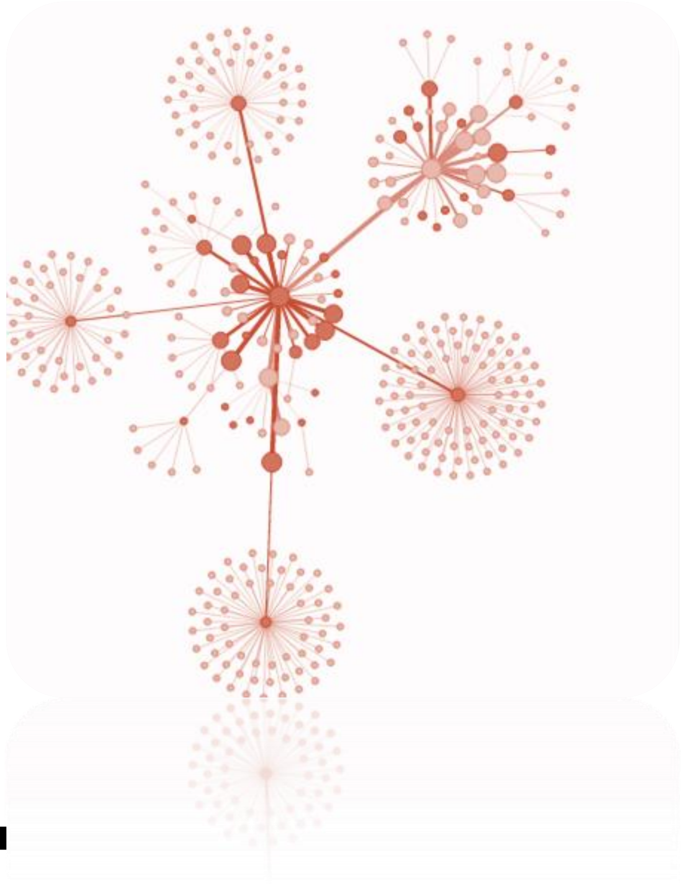
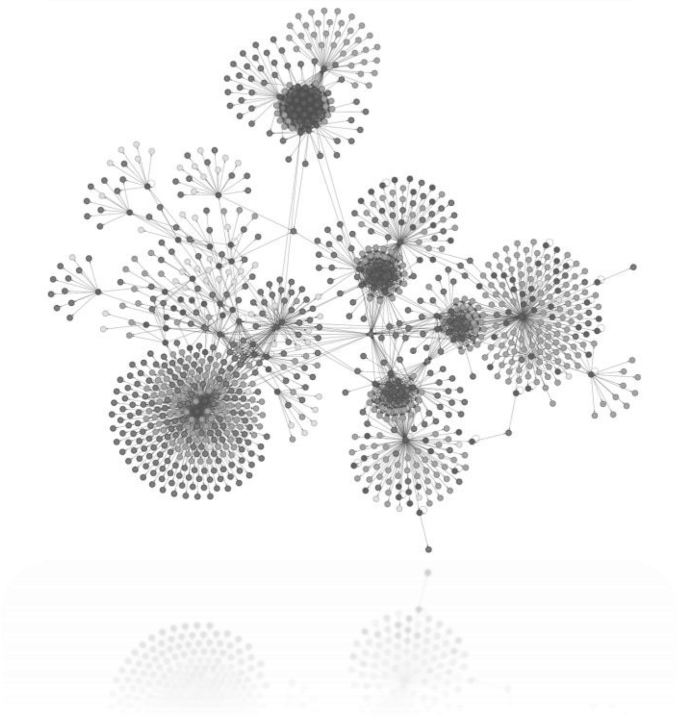
Neo4j Recommender System





Alessia Cecere

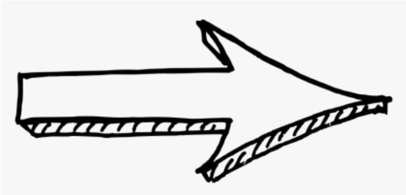
08274A

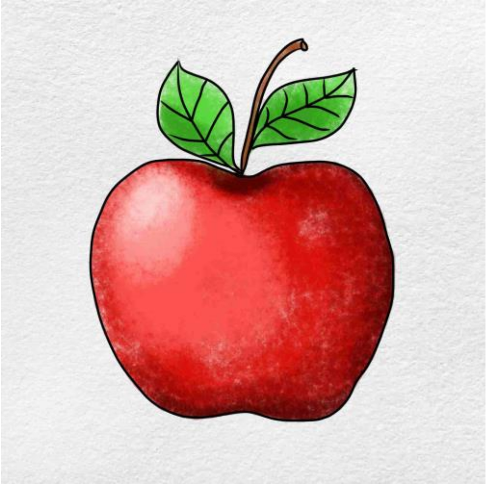
New Generation Data Models and DBMSs

What is a Recommender System?

An **information filtering** software that provides suggestions for **items** that are most pertinent to a particular **user.**

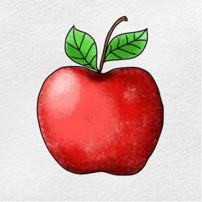
?





Recommending Techniques

Collaborative Filtering

 ⟷ 

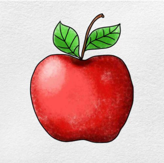
Item-based

User-based

 ⟷ 

+

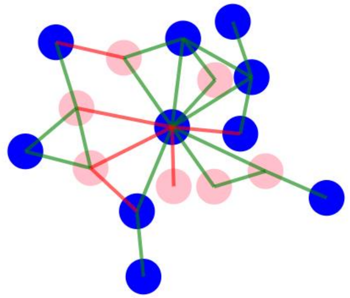
Content-based filtering

	fruit	red	green
	x	x	
	x		x

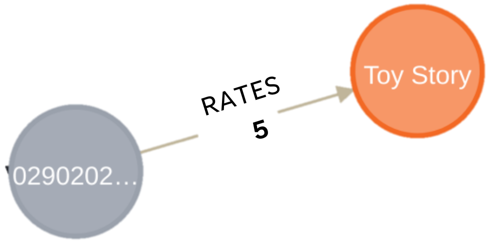
x	x	x

Why Neo4j?

Relationships **navigation**



Relationships **weight**



Dense representation

Graph algorithms

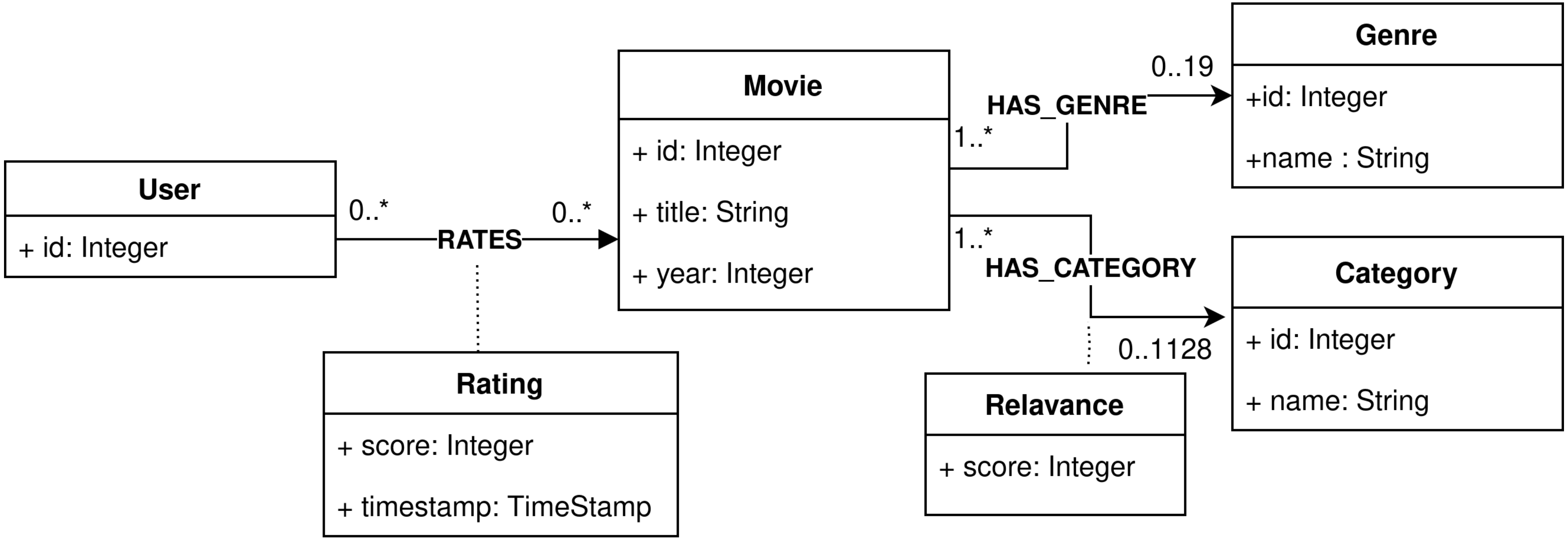
+

MovieLens 25M Dataset

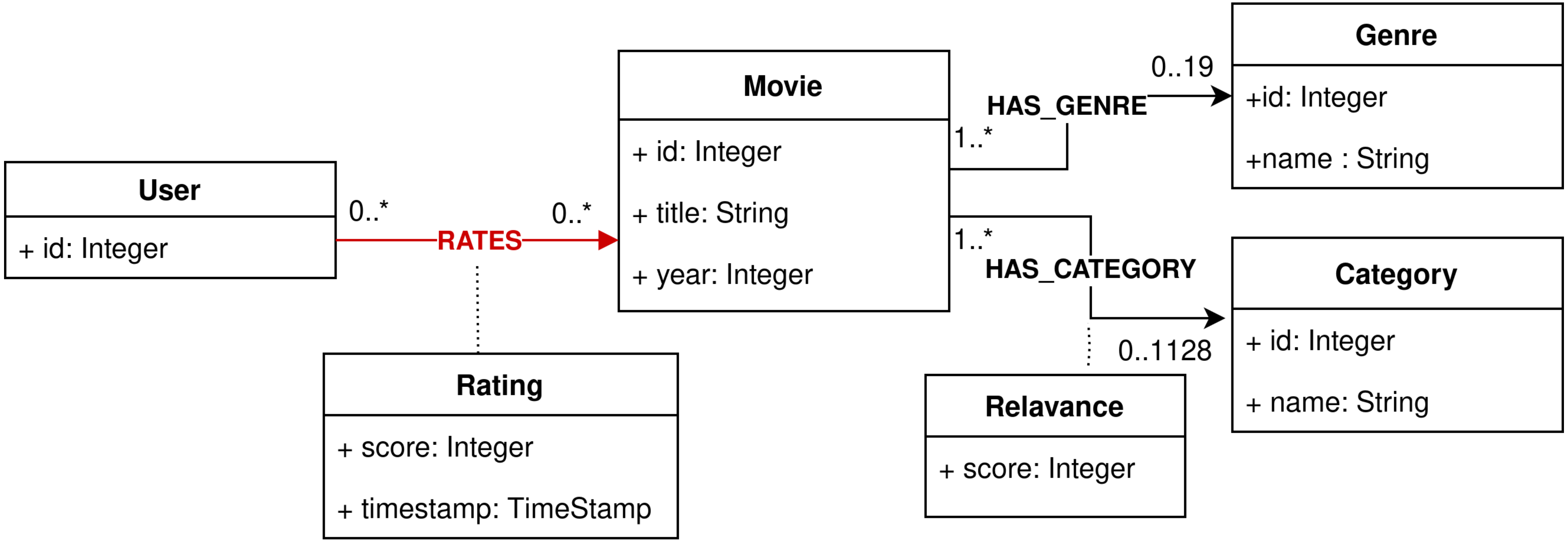
25 million **ratings** and one million **tag** applications applied to 62,000 **movies** by 162,000 **users**.

Includes **tag genome** data with 15 million relevance scores across 1,129 tags.

UML diagram


User
+ id: Integer
0..*
RATES
0..*
Movie
+ id: Integer
+ title: String
+ year: Integer
Rating
+ score: Integer
+ timestamp: TimeStamp
HAS_GENRE
1..*
0..19
Genre
+id: Integer
+name : String
HAS_CATEGORY
1..*
0..1128
Category
+ id: Integer
+ name: String
Relavance
+ score: Integer

UML diagram

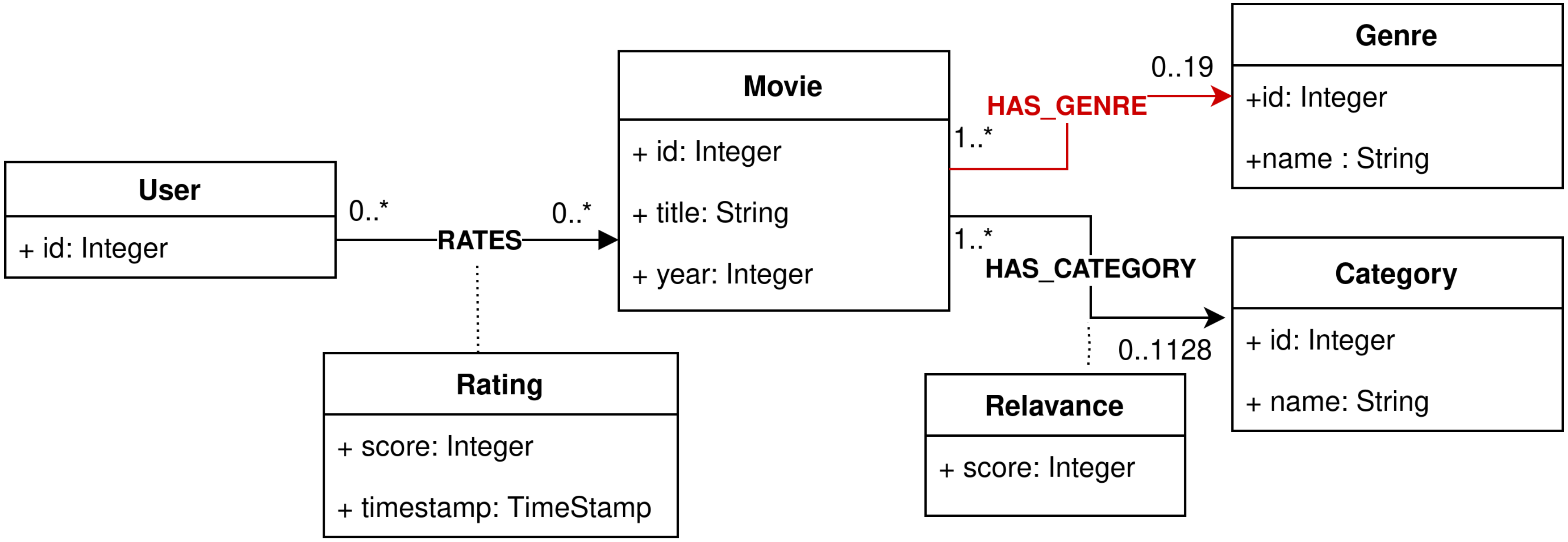


User
+ id: Integer
0..*
RATES
0..*
Movie
+ id: Integer
+ title: String
+ year: Integer
Rating
+ score: Integer
+ timestamp: TimeStamp
HAS_GENRE
1..*
0..19
Genre
+id: Integer
+name : String
1..*
HAS_CATEGORY
0..1128
Category
+ id: Integer
+ name: String
Relavance
+ score: Integer

Workflow

1. Given a **User**, find his **top k Genres**
2. Given a **User**, find his **top k Categories**
3. Given a **Genre**, find its **top k Movies**
4. Given a **Category**, find its **top k Movies**
5. Given a **User**, find **similar users**
6. Given a **User**, recommend Movies based on similar users (**collaborative filtering**)
7. Given a **Movie**, find **similar movies**
8. Given a **User**, recommend similar Movies to the ones he has watched (**content-based filtering**)

UML diagram

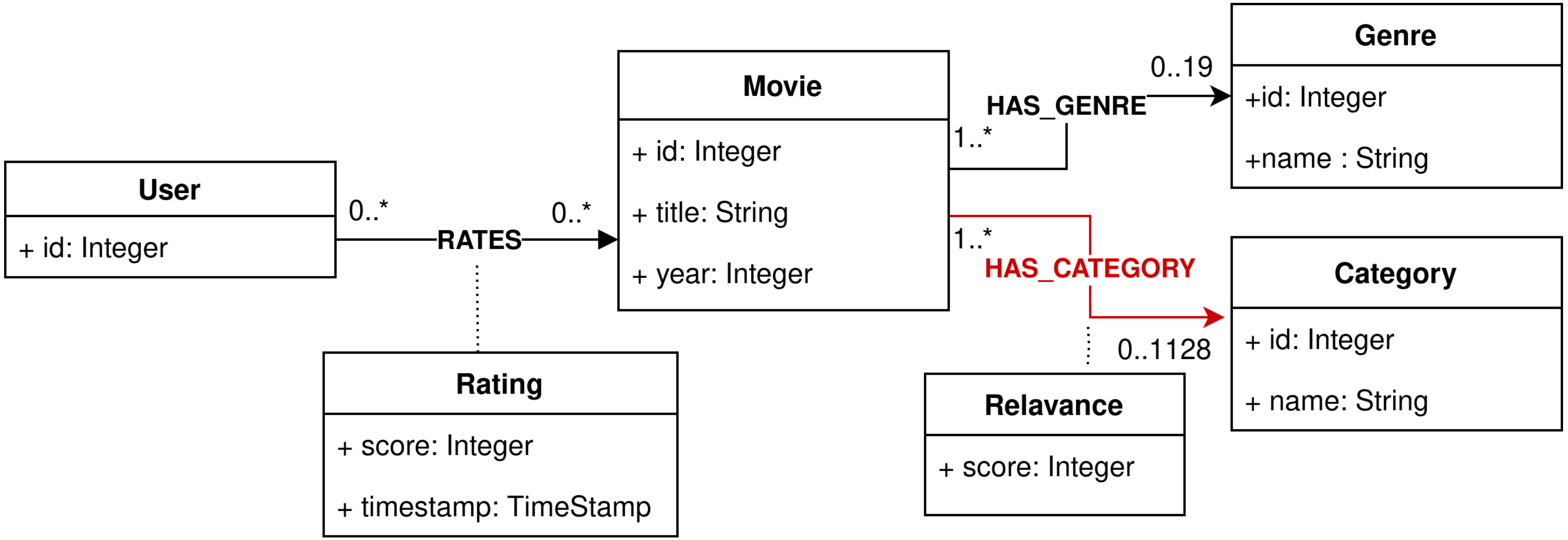


User
+ id: Integer
0..*
RATES
0..*
Movie
+ id: Integer
+ title: String
+ year: Integer
Rating
+ score: Integer
+ timestamp: TimeStamp
HAS_GENRE
1..*
0..19
Genre
+id: Integer
+name : String
1..*
HAS_CATEGORY
0..1128
Category
+ id: Integer
+ name: String
Relavance
+ score: Integer

Workflow

1. Given a **User**, find his **top k Genres**
2. Given a **User**, find his **top k Categories**
3. Given a **Genre**, find its **top k Movies**
4. Given a **Category**, find its **top k Movies**
5. Given a **User**, find **similar users**
6. Given a **User**, recommend Movies based on similar users (**collaborative filtering**)
7. Given a **Movie**, find **similar movies**
8. Given a **User**, recommend similar Movies to the ones he has watched (**content-based filtering**)

UML diagram



User
+ id: Integer
0..*
RATES
0..*
Movie
+ id: Integer
+ title: String
+ year: Integer
Rating
+ score: Integer
+ timestamp: TimeStamp
HAS_GENRE
1..*
0..19
Genre
+id: Integer
+name : String
1..*
HAS_CATEGORY
0..1128
Category
+ id: Integer
+ name: String
Relavance
+ score: Integer

Workflow

1. Given a **User**, find his **top k Genres**
2. Given a **User**, find his **top k Categories**
3. Given a **Genre**, find its **top k Movies**
4. Given a **Category**, find its **top k Movies**
5. Given a **User**, find **similar users**
6. Given a **User**, recommend Movies based on similar users (**collaborative filtering**)
7. Given a **Movie**, find **similar movies**
8. Given a **User**, recommend similar Movies to the ones he has watched (**content-based filtering**)

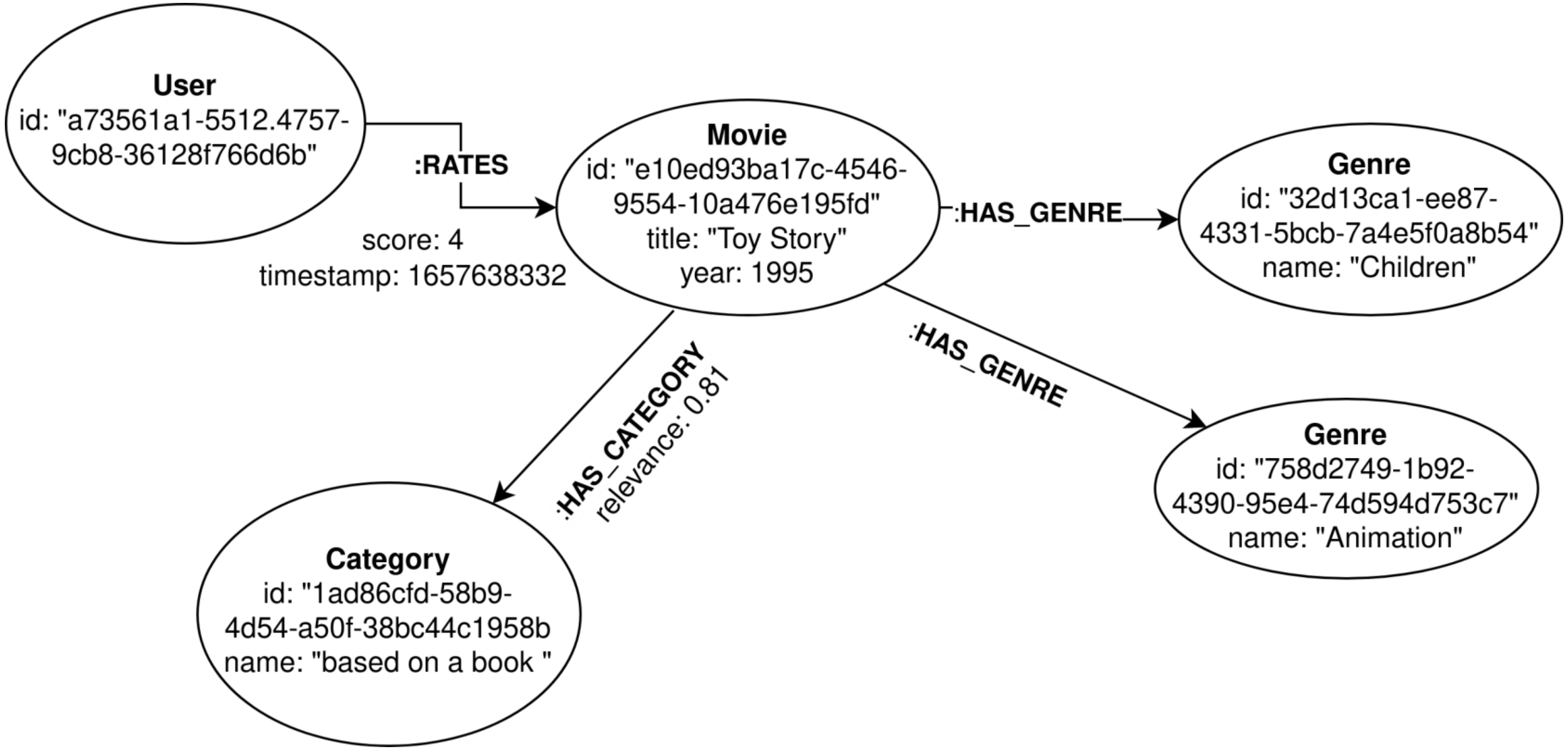
Population script

- **Py2neo:** a client library and toolkit for working with **Neo4j** from within **Python applications** and from the command line

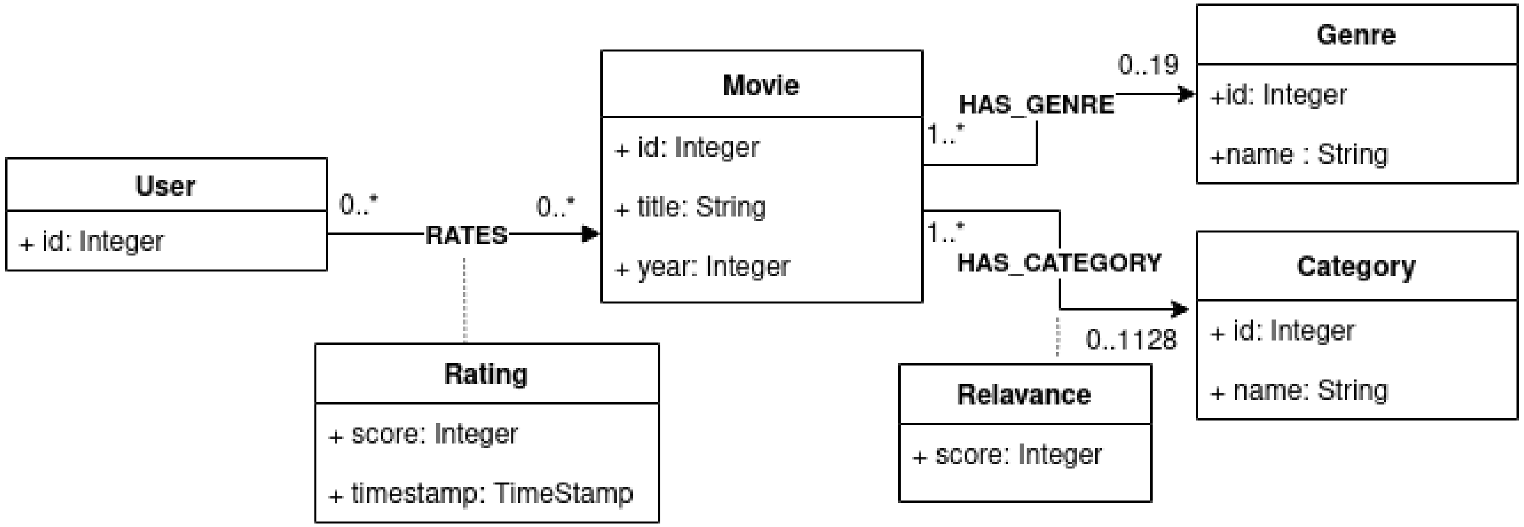
- **Nodes: CREATE** from **CSV** tables **ratings**, **movies** and **genome_tags**

- **Relations:**
  - **MATCH** and **CREATE** for **HAS_GENRE**
  - **bulk operations** for **RATES** and **HAS_CATEGORY** (batches of 10.000)

Logical model



Collaborative filtering



To find **similar users** we would need to visit an **average** of

153 MOVIES
X
432 USERS
X
153 MOVIES
=
~10 milions nodes!

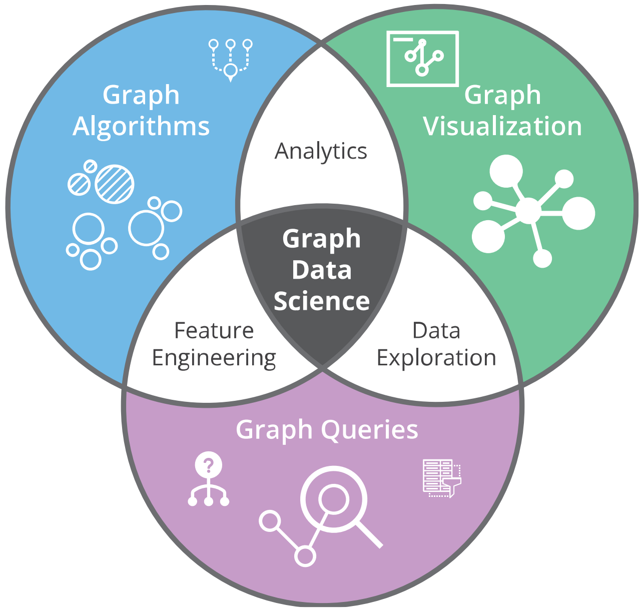
+

GDS library

- **Neo4j library** containing the efficient and parallel implementation of **different graph algorithms**, often utilized for recommendation

- **FastRP (Fast Random Projection):** creates an **embedding** to represent a node, based on its **neighbors**

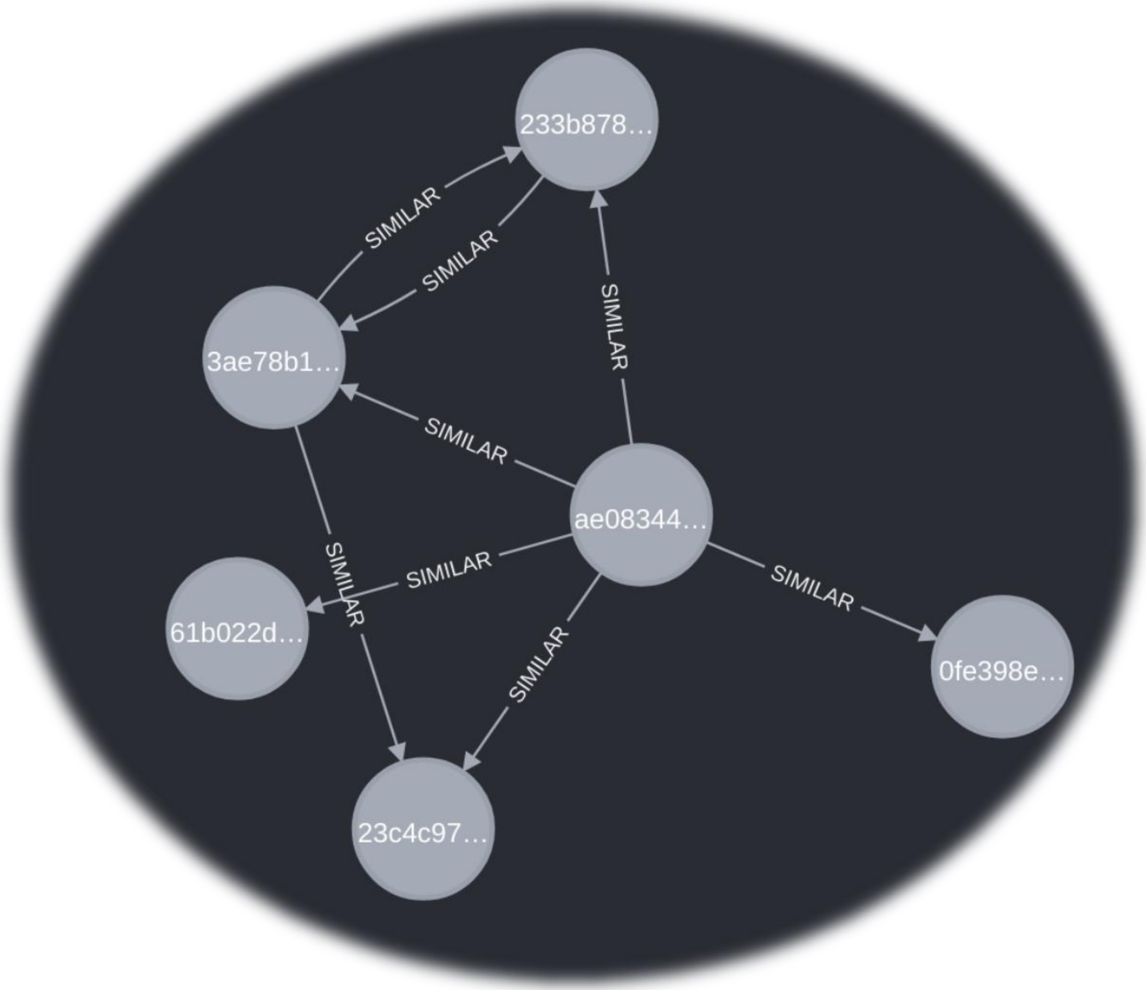
- **KNN:** finds the **k nearest neighbors** of a node



Collaborative filtering

User based

- **FastRP** on a subgraph containing **Movies**, **Users** and their relationship **rates** to create **embeddings**
- **KNN** on Users based on the embedding, **similar** relationship created



Collaborative filtering

User based

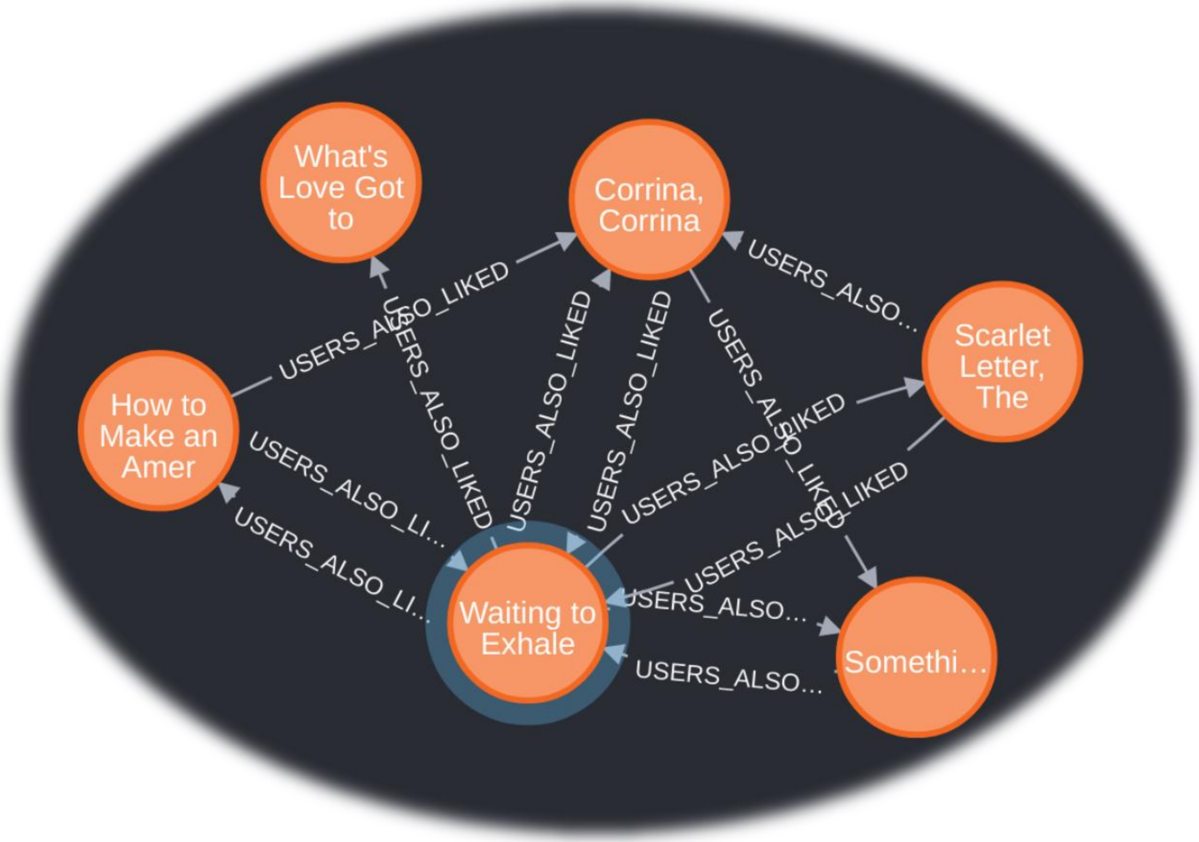
- **FastRP** on a subgraph containing **Movies**, **Users** and their relationship **rates** to create **embeddings**
- **KNN** on Users based on the embedding, **similar** relationship created

```
MATCH (u:User{ id: user_id})-[r:RATES]->(m:Movie)
WITH collect(m.id) AS watchedMoviesIds
MATCH (u)-[s:SIMILAR]->(u2:User)-[r:RATES]->(m:Movie)
WHERE NOT m.id in watchedMoviesIds
RETURN m.id as id, m.title as title,
        AVG(r.score) * AVG(s.score) AS score
ORDER BY score DESC
LIMIT 10
```

Collaborative filtering

Item based

- **FastRP** on a subgraph containing **Movies**, **Users** and their relationship **rates** to create **embeddings**
- **KNN** on **Movie** based on the embedding, **users-also-liked** relationship created

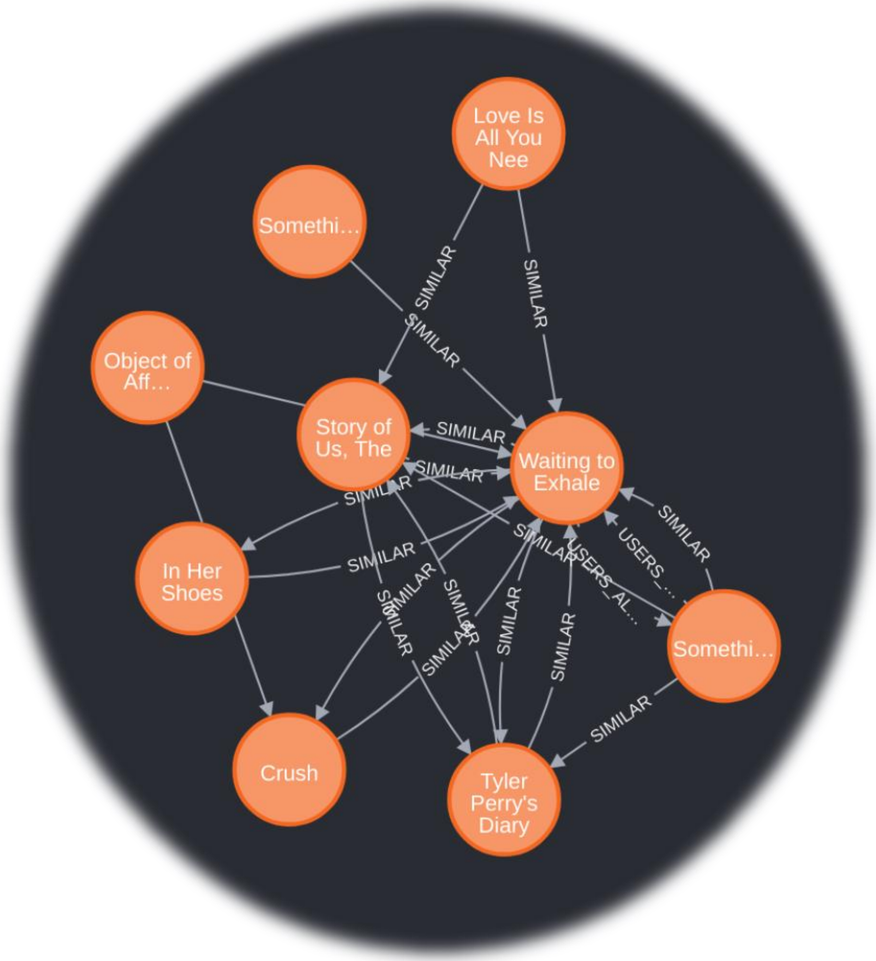


Content-based filtering

- **FastRP** on a subgraph containing **Movies, Genres** and **Categories** and their relationship **has_genre** and **has_category** to create **embeddings**

- **KNN** on **Movie** based on the embedding, **similar** relationship created

- **Hybrid approach!**



Performances

1. Given a **User**, find his **top k Genres**
2. Given a **User**, find his **top k Categories**
3. Given a **Genre**, find its **top k Movies**
4. Given a **Category**, find its **top k Movies**
5. Given a **User**, find **similar users**
6. Given a **User**, recommend Movies based on similar users (**collaborative filtering**)
7. Given a **Movie**, find **similar movies**
8. Given a **User**, recommend similar Movies to the ones he has watched (**content-based filtering**)

QUERY	TOTAL TIME	AVG TIME
1	17m 48s	6.5 ms
2	1h 23m	30 ms
3	6m	19s
4	3h 20s	10s
5	22m 6s	8 ms
6	41m 41s	15 ms
7	5m 24s	5 ms
8	34m 14s	12 ms

Performances

1. Given a **User**, find his **top k Genres**
2. Given a **User**, find his **top k Categories**
3. Given a **Genre**, find its **top k Movies**
4. Given a **Category**, find its **top k Movies**
5. Given a **User**, find **similar users**
6. Given a **User**, recommend Movies based on similar users (**collaborative filtering**)
7. Given a **Movie**, find **similar movies**
8. Given a **User**, recommend similar Movies to the ones he has watched (**content-based filtering**)

QUERY	TOTAL TIME	AVG TIME
1	17m 48s	6.5 ms
2	1h 23m	30 ms
3	6m	19s
4	3h 20s	10s
5	22m 6s	8 ms
6	41m 41s	15 ms
7	5m 24s	5 ms
8	34m 14s	12 ms

Performances

1. Given a **User**, find his **top k Genres**
2. Given a **User**, find his **top k Categories**
3. Given a **Genre**, find its **top k Movies**
4. Given a **Category**, find its **top k Movies**
5. Given a **User**, find **similar users**
6. Given a **User**, recommend Movies based on similar users (**collaborative filtering**)
7. Given a **Movie**, find **similar movies**
8. Given a **User**, recommend similar Movies to the ones he has watched (**content-based filtering**)

QUERY	TOTAL TIME	AVG TIME
1	17m 48s	6.5 ms
2	1h 23m	30 ms
3	6m	19s
4	3h 20s	10s
5	22m 6s	8 ms
6	41m 41s	15 ms
7	5m 24s	5 ms
8	34m 14s	12 ms

Conclusions

- This project was an opportunity to go deeper into the management of database **resources**, and their application to the **recommendation problem**

- **Further developments**
    - Tuning
    - Systematical **evaluation** of results (division between **test** and **train** graph)
    - Improvement of performances exploiting **parallelization** and **vertical scaling**

Thank you for your attention!