# ECE:5995 Modern Databases – Fall 2020

## Homework 6 – Neo4j

Due Thursday Nov 12th, 2020 11:59pm.

For this homework we will use the movielens (https://grouplens.org/datasets/movielens/) small collection ml-latest-small.zip (100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users). For this homework we'll only use the movies and ratings files.

Unzip the file and save the csv files into the import folder in neo4j ($NEO4J_HOME/import).

Now from the neo4j browser or the shell, run the following cypher commands:

OPTIONALLY: You can create a new database called movies by editing the neo4j.conf file in your conf directory of your $NEO4J_HOME. Search for dbms.default_database=, which should have the default value of neo4j. Uncomment the line and replace neo4j with movies, and save the file. Start neo4j again.

You may want to delete all data from the database:

```
MATCH (n) OPTIONAL MATCH (n)-[r]-() DELETE n, r;
```

**Step 1. Import the data into neo4j.**

Import the movies.csv as nodes with label Movie.

Add one node per genre with label Genre and add relationship(s) between each movie and its genres. You can use :AUTO USING PERIODIC COMMIT 500 before the LOAD CSV statement to prevent running out of memory (the data will be committed periodically releasing memory). Make sure to convert the data to the appropriate format (toInteger, toFloat).

Q1.0. (Done)

```
:AUTO USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM "file:///movies.csv" AS row
MERGE (m:Movie {movieId: toInteger(row.movieId), title: row.title})
WITH m, row
UNWIND split(row.genres, '|') AS genres
MERGE (g:Genre {name: genres})
MERGE (m)-[r:IN_GENRE]->(g);
```

Q1.1.

Create an index on movieId.

Q1.2.

Import the ratings with the label RATED for the relationship.

**Step 2. Update the movie nodes**

Q2.1. Write a cypher query to update the Movie nodes to add a release year property (extracted from the title).
Q2.2. Write a query for the movies without a release year
Q2.3. The following query counts the number of movies per decade 1980, 1990, 2000. Modify it so it uses the min and max values for the range from the movies release year (without hard-coding it).

```
UNWIND range(1980,2000,10) as startDecade
WITH startDecade, startDecade + 9 as endDecade
MATCH (movie:Movie)
WHERE movie.year >= startDecade and movie.year <= endDecade
RETURN startDecade + "-" + endDecade as years, count(movie)
ORDER BY years
```

Q2.4. Get all the movies released in 2000 with the genres of Comedy or Romance and with title that starts with M (include the png of the resulting graph including movie and genre nodes).

**Step 3. Run some queries**

Q3.1. All the movies in genre '(no genres listed)'
Q3.2. The number of movies in each genre in descending order, output genre.name and count
Q3.3. The top 5 movies with the most reviews
Q3.4. The top 10 highest rated movies (by average rating) and their genres.
Q3.5. Find all the movies that User 3 has rated.
Q3.6. The average rating for User 1 per genre

**Step 4. Update the graph.**

**REF: [https://neo4j.com/graphgist/movie-recommendations-with-k-nearest-neighbors-and-cosine-similarity](https://neo4j.com/graphgist/movie-recommendations-with-k-nearest-neighbors-and-cosine-similarity)**
Q4.1. Update the graph to add a relation labeled similarity between each pair of users with their cosine similarity as a property. (See REF-Q3)
Q4.2. Find the top 10 most similar users to user 1 (Similar to REF-Q6).
Q4.3. Update the graph to add a relation labeled genre_pref between users and genres nodes with a preference property computed to be the average rating for the movies in that genre the user has rated (do not add a relationship if the user has not rated any movies in the genre).

**Step 5. Movie recommendations**

Q5.1. Get movie recommendations for user X by selecting the top 5 rated movies that X has not yet rated in the genre for which X has the highest preference. Start your query WITH 1 as X so it is easy to change the userId.
Q5.2. REF-Q7 computes movie recommendations for user x by taking the average rating for the top 3 most similar users to x that have rated the movie. Generate a similar query that computes a

weighted average for the recommendations and returns the 10 highest recommended movies. The rate for user x, movie i ($r_{xi}$) can be computed as:

$$r_{xi} = \frac{\sum_{y \in N} sim_{xy} \cdot r_{yi}}{\sum_{y \in N} sim_{xy}}$$

Where N is the set of k=3 users most similar to x who have rated movie i, and $s_{xy}$ is the cosine similarity between user x and user y.

**Some other useful links:**
**https://neo4j.com/developer/cypher/guide-build-a-recommendation-engine/**
**http://guides.neo4j.com/sandbox/recommendations**
**https://neo4j.com/graphgists/?category=real-time-recommendations**

**Submission:**

For each Step/query, include the cypher statement and when appropriate, a png of the resulting graph, or a table with the results.
Submit your homework as a single file to the dropbox for Homework 6 in ICON by **Thursday Nov 12th 11:59pm.**