This project was developed by students Ilias Katsabalos and Leon Kalderon during the course of Big Data Systems in Athens University of Economics and Business

Big Data Systems – Neo4j

1.Introduction

In this project, we will develop a recommendation system that is used for recommending movies. The dataset we will use is the MovieLens Dataset and more particularly the ml-100k.zip file. This dataset contains the following csv files:

- **u.data**: This file contains 100 k rows of movie ratings. The format of the file is:

USERID	MOVIEID	RATING	TIMESTAMP
100	23	4	88897654
100	3	2	87665445
101	3	3	79978699

- **u.item**: This file contains the master data of the movies such as name, release date, and the genres that each movie is categorized as. The format is the following

MOVEID	NAME	RelDate	URL	GENRE1	GENRE2	GENRE3	GENRE4	GENRE5	….
1	FARGO	Dd/mm/yyyy	www.url.com	0	1	1	0	0	0

- **u.user**: This file contains the master data of the user, such as age, sex, occupation etc.
- **u.genres**: this file contains the distinct genres that exist in the *u.item* file
- **u.occupations:** this file contains the distinct occupations that exist in the *u.user* file

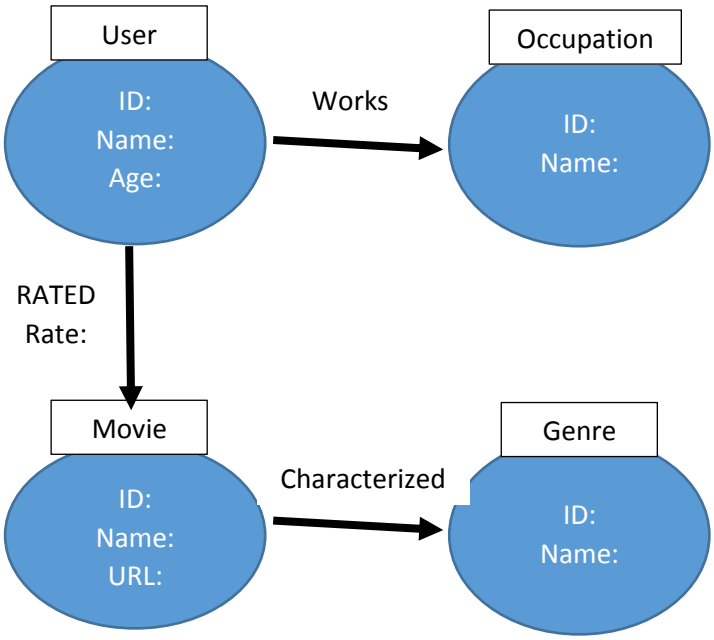
2.ETL in the u.item file

The format of the u.item file was not normalized, since the genres were listed in columns like a bitmap. Using a simple tool such as excel, we decide to separate the *u.item* file into two different files (relational tables).

- The first file contains only the master data of the movie and it kept the name u.item
- The second file is a join table/file between the new *u.item* file and the *u.genres* file, since according to the dataset, each movie can belong to many genres. The format is the following:

MOVIEID	GENREID
1	1
1	2
2	5

3.Data model in Neo4j



Above you can see the model of the data in Neo4j. As you can see we have the following nodes and vertices:

- Nodes:
  - **Users**: Holds the information of each user
  - **Occupation**: The master data of the occupation
  - **Movie**: The master data of the movie
  - **Genre**: The master data of the genres
- Relationships:
  - **Rated**: connects the users with the movies, and has the rating as a property in the relationship
  - **Works**: Connects the users with their occupations
  - **Characterized**: Connects the movies with their genres

4.Data mining on the given dataset

So far we have just utilized the data included in the ml-100k dataset. Now, in order to implement our recommendation system, we need to mine some additional information on the given dataset.

4.1 Mining association rules

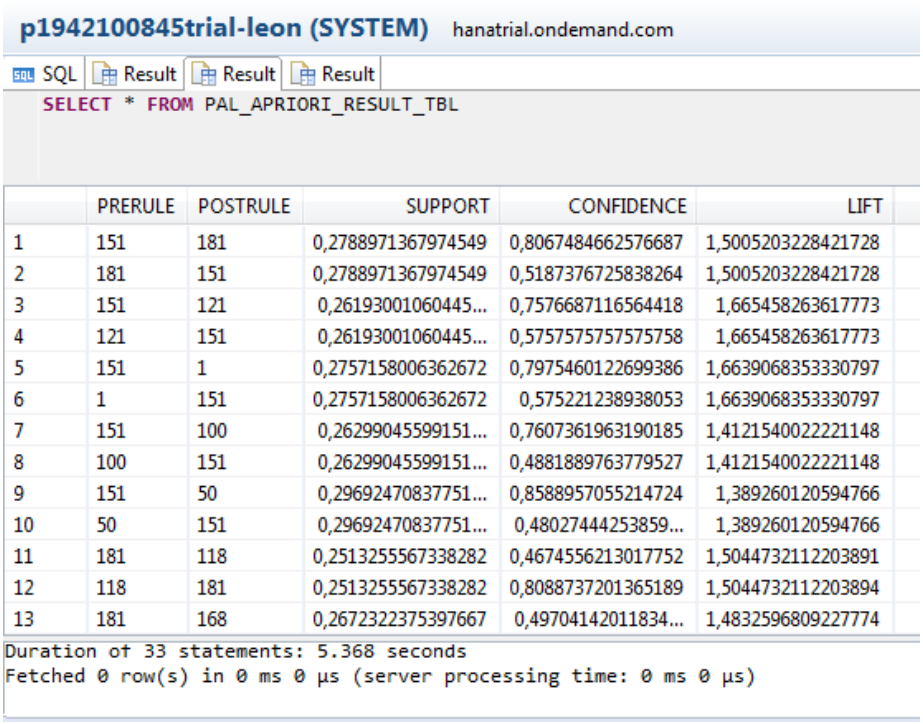
By mining association rules between the movies, we can create a new relationship amongst them that connects the pre rule with its post rule. The u.data file may serve the purpose of a transaction table that we can implement the apriori algorithm on. The platform that we decided to use is the HANA Cloud Platform. We inserted the u.data file and execute the algorithm. The code for the execution of the algorithm is the following:

```
SET SCHEMA "SYSTEM";

DROP TYPE PAL_APRIORI_DATA_T;
CREATE TYPE PAL_APRIORI_DATA_T AS TABLE(
```

```
	"TRANSID" VARCHAR(100),
	"ITEM" VARCHAR(100)
);

DROP TYPE PAL_APRIORI_RESULT_T;
CREATE TYPE PAL_APRIORI_RESULT_T AS TABLE(
	"PRERULE" VARCHAR(500),
	"POSTRULE" VARCHAR(500),
	"SUPPORT" DOUBLE,
	"CONFIDENCE" DOUBLE,
	"LIFT" DOUBLE
);

DROP TYPE PAL_APRIORI_PMMLMODEL_T;
CREATE TYPE PAL_APRIORI_PMMLMODEL_T AS TABLE(
	"ID" INTEGER,
	"PMMLMODEL" VARCHAR(5000)
);

DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
	"NAME" VARCHAR(100),
	"INTARGS" INTEGER,
	"DOUBLEARGS" DOUBLE,
	"STRINGARGS" VARCHAR (100)
);

DROP TABLE PAL_APRIORI_PDATA_TBL;
CREATE COLUMN TABLE PAL_APRIORI_PDATA_TBL(
	"POSITION" INT,
	"SCHEMA_NAME" NVARCHAR(256),
	"TYPE_NAME" NVARCHAR(256),
	"PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_APRIORI_PDATA_TBL VALUES (1, 'SYSTEM',
'PAL_APRIORI_DATA_T', 'IN');
INSERT INTO PAL_APRIORI_PDATA_TBL VALUES (2, 'SYSTEM', 'PAL_CONTROL_T',
'IN');
INSERT INTO PAL_APRIORI_PDATA_TBL VALUES (3, 'SYSTEM',
'PAL_APRIORI_RESULT_T', 'OUT');
INSERT INTO PAL_APRIORI_PDATA_TBL VALUES (4, 'SYSTEM',
'PAL_APRIORI_PMMLMODEL_T', 'OUT');

CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('SYSTEM',
'PAL_APRIORI_RULE_PROC');

CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'APRIORIRULE',
'SYSTEM', 'PAL_APRIORI_RULE_PROC', PAL_APRIORI_PDATA_TBL);


DROP TABLE PAL_APRIORI_TRANS_TBL;
CREATE COLUMN TABLE PAL_APRIORI_TRANS_TBL LIKE PAL_APRIORI_DATA_T;
INSERT INTO PAL_APRIORI_TRANS_TBL (SELECT USERID, MOVIEID FROM
"SYSTEM"."RATINGS");

DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
	"NAME" VARCHAR(100),
	"INTARGS" INTEGER,
```

```
        "DOUBLEARGS" DOUBLE,
        "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_SUPPORT', null, 0.25, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_CONFIDENCE', null, 0.3, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_LIFT', null, 1, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_CONSEQUENT', 1, null, null);

DROP TABLE PAL_APRIORI_RESULT_TBL;
CREATE COLUMN TABLE PAL_APRIORI_RESULT_TBL LIKE PAL_APRIORI_RESULT_T;

DROP TABLE PAL_APRIORI_PMMLMODEL_TBL;
CREATE COLUMN TABLE PAL_APRIORI_PMMLMODEL_TBL LIKE PAL_APRIORI_PMMLMODEL_T;

CALL "SYSTEM".PAL_APRIORI_RULE_PROC(PAL_APRIORI_TRANS_TBL,
#PAL_CONTROL_TBL, PAL_APRIORI_RESULT_TBL, PAL_APRIORI_PMMLMODEL_TBL) WITH
overview;


SELECT * FROM PAL_APRIORI_RESULT_TBL;
SELECT * FROM PAL_APRIORI_PMMLMODEL_TBL;
```

Here is the result of the algorihm:



	PRERULE	POSTRULE	SUPPORT	CONFIDENCE	LIFT
1	151	181	0,2788971367974549	0,8067484662576687	1,5005203228421728
2	181	151	0,2788971367974549	0,5187376725838264	1,5005203228421728
3	151	121	0,26193001060445...	0,7576687116564418	1,665458263617773
4	121	151	0,26193001060445...	0,5757575757575758	1,665458263617773
5	151	1	0,2757158006362672	0,7975460122699386	1,6639068353330797
6	1	151	0,2757158006362672	0,575221238938053	1,6639068353330797
7	151	100	0,26299045599151...	0,7607361963190185	1,4121540022221148
8	100	151	0,26299045599151...	0,4881889763779527	1,4121540022221148
9	151	50	0,29692470837751...	0,8588957055214724	1,389260120594766
10	50	151	0,29692470837751...	0,48027444253859...	1,389260120594766
11	181	118	0,2513255567338282	0,4674556213017752	1,5044732112203891
12	118	181	0,2513255567338282	0,8088737201365189	1,5044732112203894
13	181	168	0,2672322375397667	0,49704142011834...	1,4832596809227774

```
Duration of 33 statements: 5.368 seconds
Fetched 0 row(s) in 0 ms 0 µs (server processing time: 0 ms 0 µs)
```

According to the results, each row is relationship that connects two movies, and it has three properties: support, confidence and lift. After the execution we export the results into a csv file in order to import to neo4j

4.2 Clustering

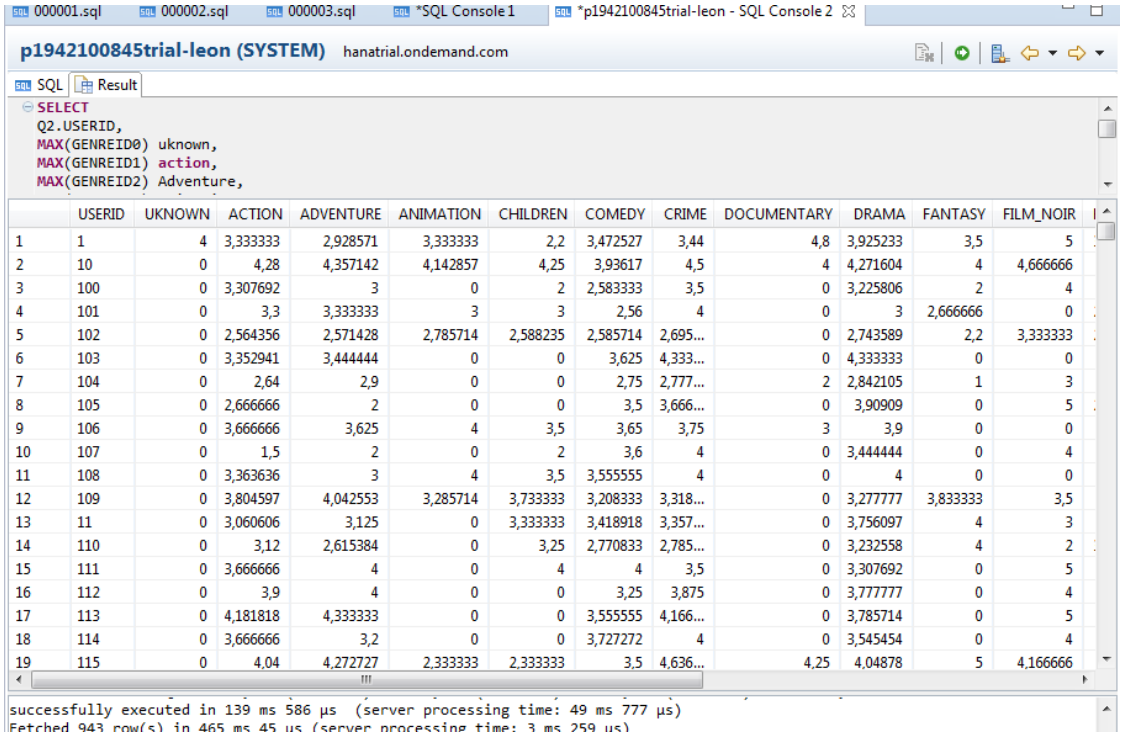
We clustered the users according to their average ratings on each genre. Though, because we have 18 genres, we implemented a dimension reduction technique, so that we have fewer than 18 dimensions to perform our clustering algorithm on. After the Principal

Component analysis, we performed the clustering on 5 principal factors derived from our PCA. Here are the steps described in more detail.

By using the SAP HANA Cloud Platform, we performed the following query so that we receive a pivot table that contains the average ratings of each user and for each genre.

```
SELECT
Q2.USERID,
MAX(GENREID0) uknown,
MAX(GENREID1) action,
MAX(GENREID2) Adventure,
MAX(GENREID3) Animation,
MAX(GENREID4) Children,
MAX(GENREID5) Comedy,
MAX(GENREID6) Crime,
MAX(GENREID7) Documentary,
MAX(GENREID8) Drama,
MAX(GENREID9) Fantasy,
MAX(GENREID10) Film_Noir,
MAX(GENREID11) Horror,
MAX(GENREID12) Musical,
MAX(GENREID13) Mystery,
MAX(GENREID14) Romance,
MAX(GENREID15) Sci_Fi,
MAX(GENREID16) Thriller,
MAX(GENREID17) War,
MAX(GENREID18) Western
FROM
(SELECT
Q1.USERID,
CASE WHEN Q1.GENREID = 0 THEN Q1.Average ELSE 0 END AS GENREID0,
CASE WHEN Q1.GENREID = 1 THEN Q1.Average ELSE 0 END AS GENREID1,
CASE WHEN Q1.GENREID = 2 THEN Q1.Average ELSE 0 END AS GENREID2,
CASE WHEN Q1.GENREID = 3 THEN Q1.Average ELSE 0 END AS GENREID3,
CASE WHEN Q1.GENREID = 4 THEN Q1.Average ELSE 0 END AS GENREID4,
CASE WHEN Q1.GENREID = 5 THEN Q1.Average ELSE 0 END AS GENREID5,
CASE WHEN Q1.GENREID = 6 THEN Q1.Average ELSE 0 END AS GENREID6,
CASE WHEN Q1.GENREID = 7 THEN Q1.Average ELSE 0 END AS GENREID7,
CASE WHEN Q1.GENREID = 8 THEN Q1.Average ELSE 0 END AS GENREID8,
CASE WHEN Q1.GENREID = 9 THEN Q1.Average ELSE 0 END AS GENREID9,
CASE WHEN Q1.GENREID = 10 THEN Q1.Average ELSE 0 END AS GENREID10,
CASE WHEN Q1.GENREID = 11 THEN Q1.Average ELSE 0 END AS GENREID11,
CASE WHEN Q1.GENREID = 12 THEN Q1.Average ELSE 0 END AS GENREID12,
CASE WHEN Q1.GENREID = 13 THEN Q1.Average ELSE 0 END AS GENREID13,
CASE WHEN Q1.GENREID = 14 THEN Q1.Average ELSE 0 END AS GENREID14,
CASE WHEN Q1.GENREID = 15 THEN Q1.Average ELSE 0 END AS GENREID15,
CASE WHEN Q1.GENREID = 16 THEN Q1.Average ELSE 0 END AS GENREID16,
CASE WHEN Q1.GENREID = 17 THEN Q1.Average ELSE 0 END AS GENREID17,
CASE WHEN Q1.GENREID = 18 THEN Q1.Average ELSE 0 END AS GENREID18

FROM
(SELECT T0.USERID, T1.GENREID, AVG(T0.RATING) AS Average FROM
"SYSTEM"."RATINGS" T0
INNER JOIN "SYSTEM"."MOVIE_GENRE" T1 ON T0.MOVIEID=T1.MOVIEID
GROUP BY T0.USERID, T1.GENREID ) AS Q1) AS Q2
GROUP BY q2.USERID ORDER BY 1,2
```

The results of the query are:



000001.sql | 000002.sql | 000003.sql | *SQL Console 1 | *p1942100845trial-leon - SQL Console 2

p1942100845trial-leon (SYSTEM) hanatrial.ondemand.com

SQL | Result

```
SELECT
  Q2.USERID,
  MAX(GENREID0) uknown,
  MAX(GENREID1) action,
  MAX(GENREID2) Adventure,
```

	USERID	UKNOWN	ACTION	ADVENTURE	ANIMATION	CHILDREN	COMEDY	CRIME	DOCUMENTARY	DRAMA	FANTASY	FILM_NOIR
1	1	4	3,333333	2,928571	3,333333	2,2	3,472527	3,44	4,8	3,925233	3,5	5
2	10	0	4,28	4,357142	4,142857	4,25	3,93617	4,5	4	4,271604	4	4,666666
3	100	0	3,307692	3	0	2	2,583333	3,5	0	3,225806	2	4
4	101	0	3,3	3,333333	3	3	2,56	4	0	3	2,666666	0
5	102	0	2,564356	2,571428	2,785714	2,588235	2,585714	2,695...	0	2,743589	2,2	3,333333
6	103	0	3,352941	3,444444	0	0	3,625	4,333...	0	4,333333	0	0
7	104	0	2,64	2,9	0	0	2,75	2,777...	2	2,842105	1	3
8	105	0	2,666666	2	0	0	3,5	3,666...	0	3,90909	0	5
9	106	0	3,666666	3,625	4	3,5	3,65	3,75	3	3,9	0	0
10	107	0	1,5	2	0	2	3,6	4	0	3,444444	0	4
11	108	0	3,363636	3	4	3,5	3,555555	4	0	4	0	0
12	109	0	3,804597	4,042553	3,285714	3,733333	3,208333	3,318...	0	3,277777	3,833333	3,5
13	11	0	3,060606	3,125	0	3,333333	3,418918	3,357...	0	3,756097	4	3
14	110	0	3,12	2,615384	0	3,25	2,770833	2,785...	0	3,232558	4	2
15	111	0	3,666666	4	0	4	4	3,5	0	3,307692	0	5
16	112	0	3,9	4	0	0	3,25	3,875	0	3,777777	0	4
17	113	0	4,181818	4,333333	0	0	3,555555	4,166...	0	3,785714	0	5
18	114	0	3,666666	3,2	0	0	3,727272	4	0	3,545454	0	4
19	115	0	4,04	4,272727	2,333333	2,333333	3,5	4,636...	4,25	4,04878	5	4,166666

successfully executed in 139 ms 586 µs (server processing time: 49 ms 777 µs)
Fetched 943 row(s) in 465 ms 45 µs (server processing time: 3 ms 259 µs)

We export this query and transfer it to SPSS. In the SPSS we perform a factor analysis (PCA) and we receive 5 principal components according to our analysis:

Total Variance Explained

Component	Initial Eigenvalues			Extraction Sums of Squared Loadings			Rotation Sums of Squared Loadings		
	Total	% of Variance	Cumulative %	Total	% of Variance	Cumulative %	Total	% of Variance	Cumulative %
1	5,249	29,159	29,159	5,249	29,159	29,159	3,697	20,540	20,540
2	2,187	12,150	41,309	2,187	12,150	41,309	2,512	13,957	34,498
3	1,600	8,887	50,195	1,600	8,887	50,195	1,910	10,611	45,109
4	1,157	6,427	56,623	1,157	6,427	56,623	1,843	10,241	55,349
5	,916	5,087	61,710	,916	5,087	61,710	1,145	6,360	61,710
6	,824	4,578	66,287						
7	,765	4,252	70,539						
8	,728	4,046	74,585						
9	,667	3,705	78,291						
10	,617	3,430	81,720						
11	,584	3,242	84,962						
12	,561	3,116	88,078						
13	,519	2,885	90,963						
14	,423	2,348	93,312						
15	,404	2,247	95,558						
16	,306	1,701	97,260						
17	,270	1,501	98,761						
18	,223	1,239	100,000						

Rotated Component Matrix[a]

	Component				
	1	2	3	4	5
Action	,606	,066	,552	,116	-,190
Adventure	,339	,238	,718	-,046	,071
Animation	,126	,769	,158	-,050	,071
Children	,180	,748	,194	-,159	,010
Comedy	,746	,130	,071	-,009	,097
Crime	,465	-,157	,090	,304	,430
Documentary	,008	,222	,008	,067	,839
Drama	,857	,054	,076	,056	,100
Fantasy	-,057	,642	,181	,232	,084
Film_Noir	-,020	,068	-,049	,722	,290
Horror	-,002	,135	,353	,601	-,073
Musical	,209	,636	-,135	,260	-,027
Mystery	,368	,091	-,082	,608	-,041
Romance	,828	,118	,137	-,016	-,059
Sci_Fi	,174	,139	,818	,103	,048
Thriller	,619	,039	,259	,392	-,159
War	,615	,211	,248	,091	,074
Western	,029	,529	,088	,393	,235

By interpreting the results of the factor analysis, we can see the weight of each genre to the average of the corresponding factor. The factors derived from the PCA are:

- Factor1: **Commercial**
- Factor2: **Children**
- Factor3: **Adventure_SciFI**
- Factor4: **Horror**
- Factor5: **Documentary**

The next step is to perform the cluster analysis based on the five factors above. Here are the results.

Final Cluster Centers

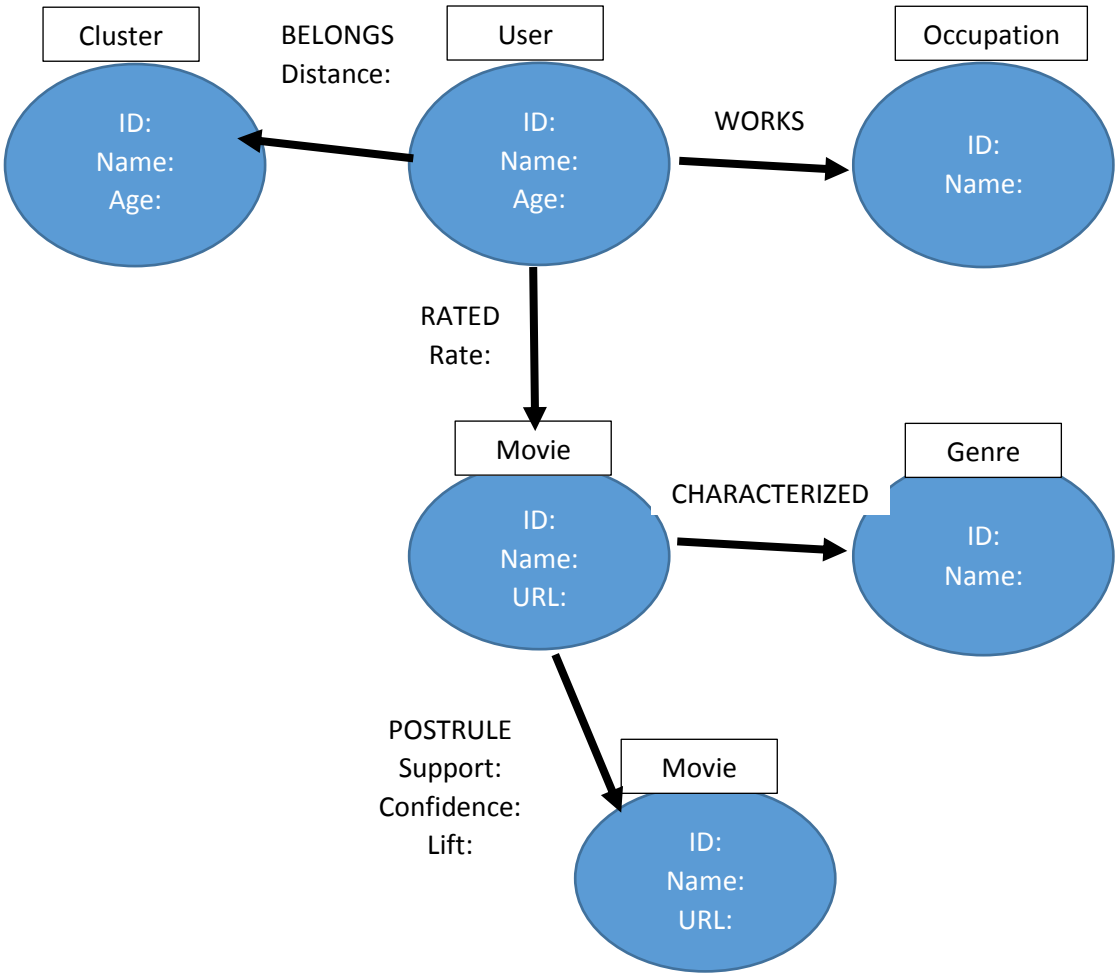
	Cluster				
	1	2	3	4	5
Commercial	,81776	,23804	-1,22116	,41348	,33843
Childer	-,55942	,74725	,00354	-,14358	-1,37639
SCI_FI	-2,58793	,11648	-,13607	,39197	,45343
Horror	-,31489	,49532	-,10071	-1,49798	,55061
Documentary	-,52670	,32035	-,39852	,11407	-,14186

So after the execution, we have for each user, the cluster to which he belongs and the distance to the corresponding cluster. For example:

USER	Cluster	Distance
1	2	1,35046
10	2	1,37181
100	3	1,17450
101	3	,60903

We export the csv in order to import it to neo4j.

5. Final Model in Neo4j



According to the new model, we added the **Cluster nodes** and the relationship **belongs** that connects the users with the clusters. The second thing that we added is the relationship **postrule** that connects the movies with each other based on the association rules.

6. The recommendation queries

The first thing you will need to do is to either open the database which has already imported the data, or open the default neo4j database and import the data yourself. In the following chapters, both alternatives are described

6.1 Open the neo4j Database

In this option, the database has already imported the csv files, and you can execute the recommendation queries directly.

- Download the zip file from here. Extract the *project4_neo4j.rar* .
- Open neo4j and click *choose* in the database location field. Navigate to the *project4_neo4j* folder (that you just extracted), choose the folder *movieLens.graphdb* and click *open*. Then in the neo4j dialogue box click start and head over to the link listed in the box.
- The password is *movielens*

Now you have successfully opened the database in your browser. Skip to chapter 6.3 for the recommendation queries.

6.2 Import the csv in the default.graphdb

In this option you will have to import all the csv files in order for the queries to retrieve the desirable results.

- Download the zip file from here. Extract the *project4_neo4j.rar*
- Head over to your documents folder and choose the *neo4j* folder. Inside that folder, there is another folder called *default.graphdb.* Open that folder and **create** a new folder called *import*.
- In the *project4_neo4j* folder, open the folder *csvImport.* Copy all the files into the folder *import* that you created in the previous step.
- In the *project4_neo4j* folder, open the folder *queries.* Execute all the cypher queries in the order listed.

Now, you should have the database ready for the recommendation queries.

6.3 Execute the recommendation queries

In the *project4_neo4j* folder, open the folder *RecommendQry.* Execute the queries in the neo4j browser.

Let us take a closer look into what each query does:

1. **Most Popular**

```
MATCH (u:User)-[r:RATED]-(m:Movie)

WITH m,count(r) AS views

RETURN m.name AS Name,m.DateToDVD AS ReleaseDate,views

ORDER BY  views DESC LIMIT 250
```

This is query is pretty simple, as it recommends the movies that have the most incoming relationships from users, in other words the most ratings (and hypothetically the most views)

2. **Apriori**

```
MATCH (u:User {id:"1"})-[r:RATED]->(m1:Movie)-[p:POSTRULE]->(m2:Movie)<-[r2:RATED]-(u1:User)

WHERE NOT (u)-[r]->(m2)

RETURN  m2.name,avg(toInt(r2.rate)),  count(r2) as count

ORDER BY count DESC

LIMIT 100
```

This query finds all the movies that user 1 has watched, and matches all the outgoing **POSTRULE** relationships, according to the association rules (Chapter 4.1). These postrules – movies are sorted and presented according to how many views they have from other users**.**

3. **RecCluster**

```
match (u1:User {id:"1"})-[b1:BELONGS]->(c:Cluster)<-[b2:BELONGS]-(u2:User)-[r:RATED]->(m:Movie)

WITH m,avg(toInt(r.rate)) AS avgRate, count(r) AS count

WHERE  avgRate>4

return   m.name,count,avgRate

ORDER BY count DESC

LIMIT 200
```

This query finds the cluster the user 1 belongs to, and all the other users that belong to the same cluster. For those users, the query matches the movies that they have watched and recommends them to the user 1 sorted by their views and filtered with their average rating > 4

4. **moviesFromFavoriteGenre**

```
MATCH (u:User {id:"1"})-[r1:RATED]->(m:Movie)-[c:CHARACTERIZED]->(g:Genre)

WITH g,avg(toInt(r1.rate)) AS avgGenre

ORDER BY avgGenre DESC

LIMIT 2

MATCH (g)<-[c2:CHARACTERIZED]-(m2:Movie)<-[r2:RATED]-(u2:User)

WITH m2,avg(toInt(r2.rate)) AS avgMovie

return m2.name as Title,avgMovie

ORDER BY avgMovie DESC
```

This query finds the favorite 2 genres that the user have, according to the biggest 2 average ratings for each genre. For those 2 genres, the query finds all the movies that belong to each one of them and recommends them to the user 1 sorted by their average rating.