Group Coursework Submission Form

Specialist Masters Programme

Please list all names of group members: (Surname, first name) 1.Hamade, Ramez 2.Gawlowski, Karol 3.Karwa, Radhika	4. Kalochoritou, Phoebe 5. 6. 7. **GROUP NUMBER: 15**
MSc in: Business Analytics	
Module Code: SMM695	
Module Title: Data Management Systems	
Lecturer: Matteo Devigili	18/06/2021

Declaration:

By submitting this work, we declare that this work is entirely our own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the coursework instructions and any other relevant programme and module documentation. In submitting this work we acknowledge that we have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. We also acknowledge that this work will be subject to a variety of checks for academic misconduct.

We acknowledge that work submitted late without a granted extension will be subject to penalties, as outlined in the Programme Handbook. Penalties will be applied for a maximum of five days lateness, after which a mark of zero will be awarded.

Marker's Comments (if not being marked on-line):

Deduction for Late Submission: | **Final Mark:** **%**

Table of Contents

1. Introduction

Database management has quickly become a key process that is undertaken in everyday activities. It is a fundamental component of the digital era and understanding how it works is vital for every organisation.

Hence, the purpose of this project is to use the Internet Movie Database (IMDb) dataset to build five meaningful views on PostgreSQL and create a maximally efficient relational database. The underlying learning objectives here are to:

- Learn about SQL and PgAdmin
- Learn the essentials of database design, preparing the Data, logical schema, and Indexing.
- Practice database querying by posing basic and more advanced queries using Pgadmin.

2. IMDB Dataset Preparation

The first step we undertook was to create the same tables with our data set under a new schema. l The data was imported in its original tsv form in the PgAdmin and to avoid any issues we decided to set the initial data types as text. The next step was to alter the data types and set them in the appropriate form for each attribute. (Table 1)

DataType	Variables Used	Why?
Integer	Numvotes, birth and death year, ordering, runtime minutes, start and end year, season and episode number	These variables were given the data type integer as they are whole numbers without any fractional component.
Text	Types, attributes, title, genres, job, characters, category, primaryName, original title, primary title	Columns which contained text data were given the data type text despite their length.
Text Array	Primary profession, genres, attributes, types	Any text data that included more than one option in each row was set as a text array.
Boolean	Is adult, isorginaltitle	This data type has either a truth or false value.
Character Varying	Tconst, title type,, nconst, knownfortitles, language, region, titled, region, directors, writers, parenttconst	This data type stores a string of letters and symbols of different lengths. We have set the length to be 15 as no character exceeded this amount.

Numeric	Average rating	Average rating was set to the data type of numeric as the numbers were in decimal.

Table 1: DataTypes used and why

While undertaking this task it became evident that there are a few issues with the dataset that have to be resolved before any sort of analysis can be done. These mainly included:

1. There were many tables that had null values in the columns that affected the way we added constraints to the database.
2. The title_id(tconst) in many of the tables didn't match up with the primary key title _id's(tconst) so we had to delete the ones that weren't matching.
3. Added constraints like Primary Key, Foreign Key and applied indexes to enhance the performance of queries.

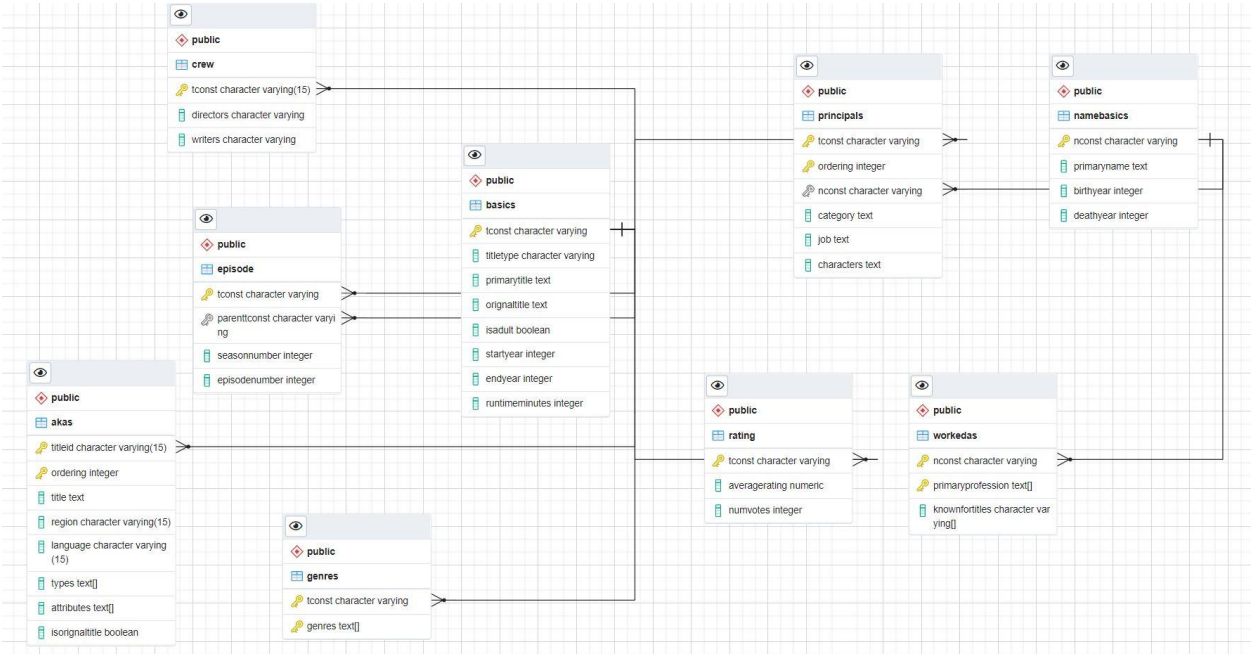
Furthermore, we decided to normalise the dataset in order to minimise the redundancy from a relation or set of relations. Normalization refers to the process of structuring the database columns and tables so that their dependencies are correctly enforced by database integrity constraints. Redundancy in relations may cause insertion, deletion and updating anomalies. So, it helps to minimise the redundancy in relations.

Although some of the missing data could have been scraped from the internet we decided that deleting them would be sufficient to reach the learning objectives of the project.

3. Logical Schema

Modeled the database using an Entity-Relationship (ER) diagram which includes entities (tables), attributes (columns) and relationships (keys).

New tables were created for multi-valued attributes, such as genres, workedas.

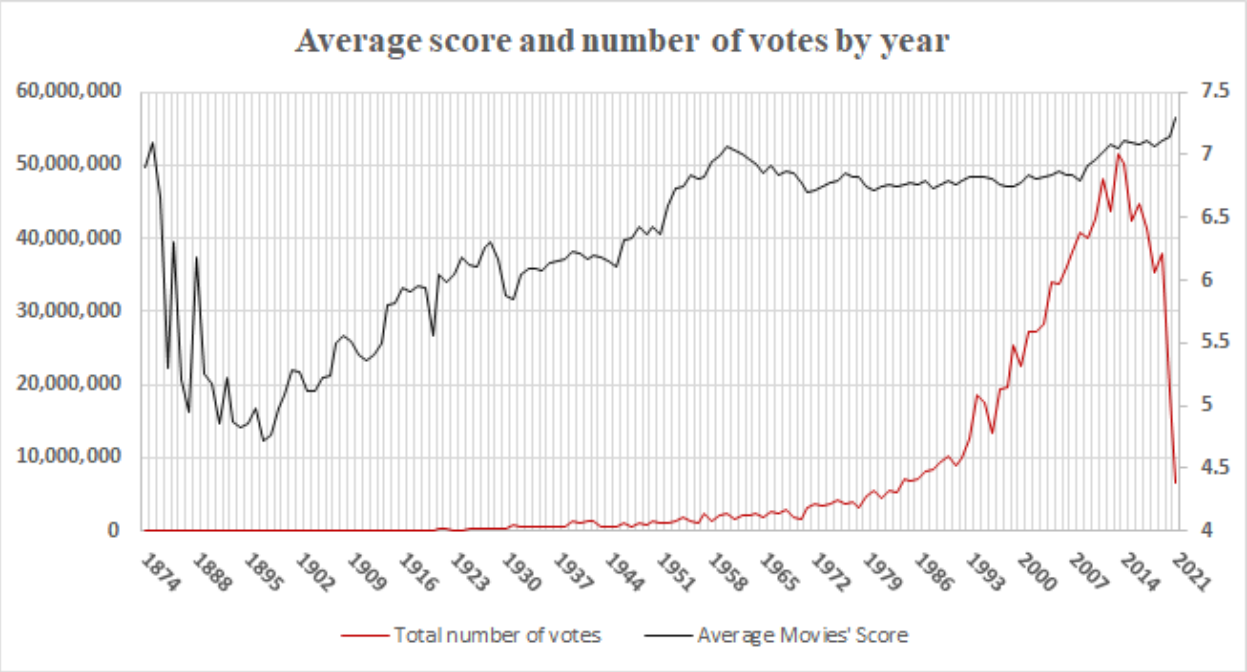


4. Meaningful Views

Query 1: Average Rating per year

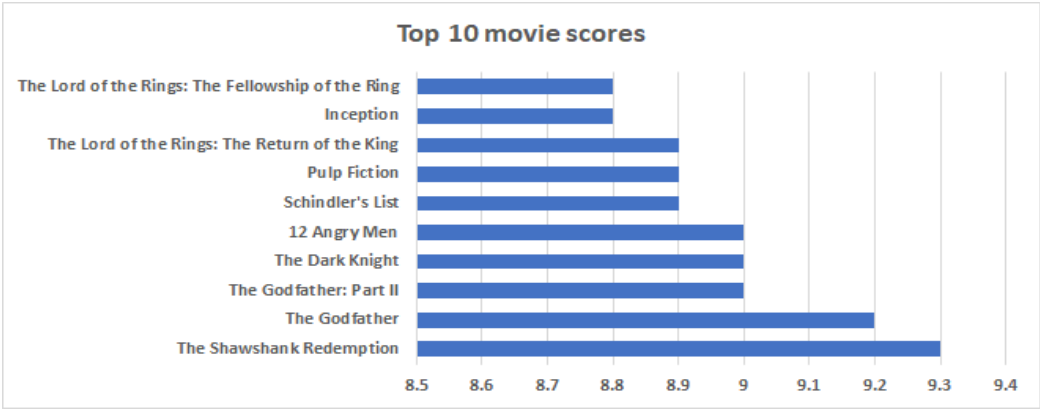
This query outputs a summary of rating characteristics per movie release date. It displays the average score assigned to the movies from a certain year and also the number of votes given by IMDB users. From this table we can learn that the scores seem to increase in time, with the exception of the earliest production years. It would be interesting to statistically explain this phenomenon, which is out of scope for this report. Presumably though, the score spike in 1877, might be caused by curiosity and unfamiliarity with over 100 year old motion pictures. It should also be considered in

conjunction with the number of votes. productions premiering earlier than 1990 account for just 15% of total votes. Clearly, IMDB users are fans of more recent cinematography but also not all of them are completely up to date with the newest releases, as after 2013 there is a significant drop in vote numbers. This particular summary is worth visualizing - see figure 2.



Query 2: Top 10 Movies According to Rating

For the second query we created a table of the top 10 movies ordered by their respective average rating. We only included the titletype ‘movie’, year released and movies that received over 200,000 votes. We did this to filter out movies that don't have enough ratings making our list more accurate.



Query 3: *Friends* Average Rating Per Season and Episode

For the third query we decided to show the data related to a specific TV show. This query shows the various episodes that are in each season of the TV Show “Friends” along with each episode's average rating from the ratings table. The reason for choosing this approach is to show how our database retrieves specific information .

seasonnumber integer	episodenumber integer	primarytitle text	averagerating numeric
1	1	The One Where Monica Gets a Roommate	8.3
1	2	The One with the Sonogram at the End	8.0
1	3	The One with the Thumb	8.1
1	4	The One with George Stephanopoulos	8.1
1	5	The One with the East German Laundry Detergent	8.5
1	6	The One with the Butt	8.1
1	7	The One with the Blackout	9.0
1	8	The One Where Nana Dies Twice	8.1
1	9	The One Where Underdog Gets Away	8.2
1	10	The One with the Monkey	8.1

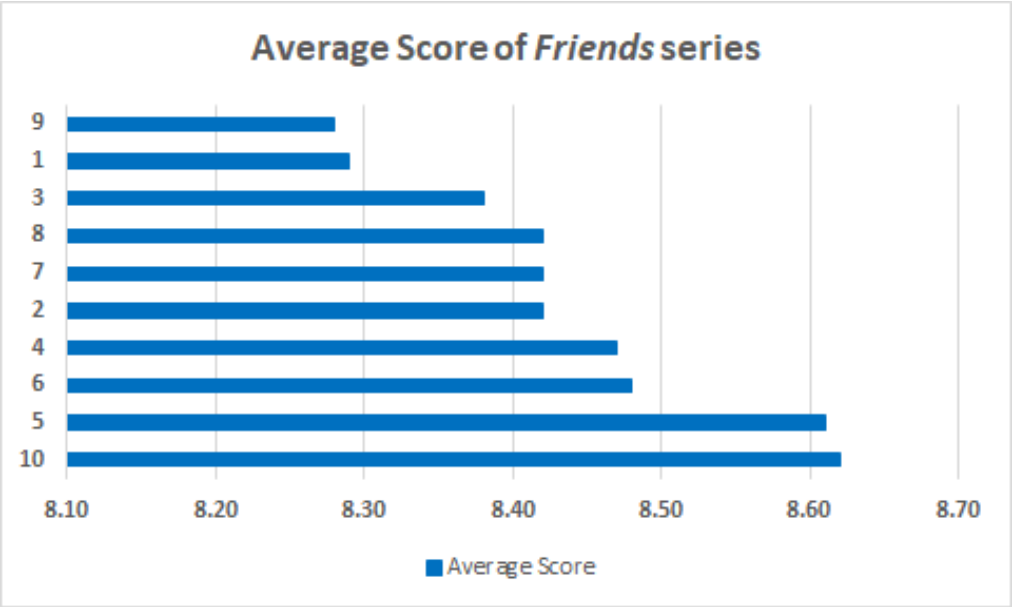
Query 4: The most popular episode of Friends.

To investigate the friends show further we found the best rated episodes across all 10 seasons (see Table below). This is a subset of aforementioned Query 3 with applied max by averagerating and got two largest observations.

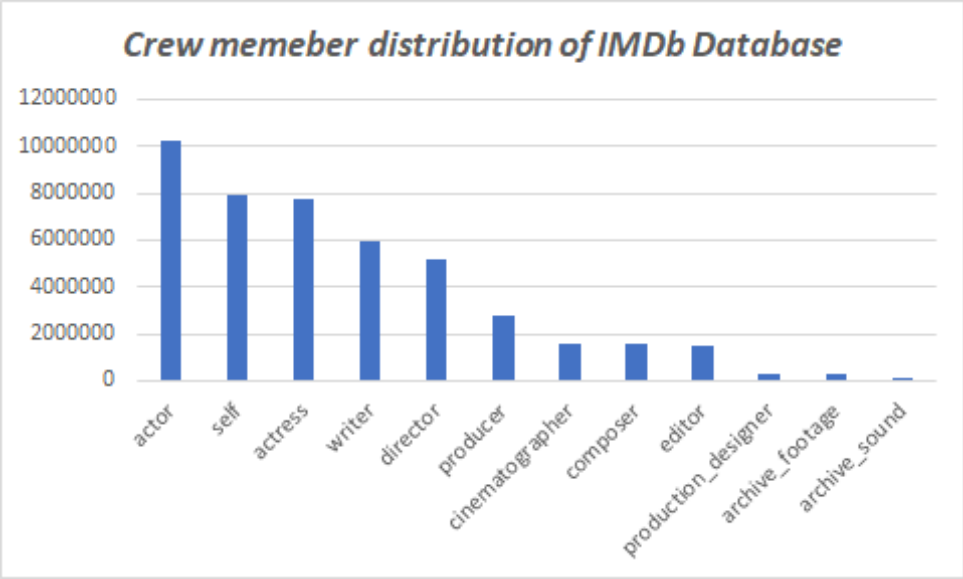
seasonnumber integer	episodenumber integer	primarytitle text	averagerating numeric
5	14	The One Where Everybody Finds Out	9.7
10	17	The Last One	9.7

Query 5: This query summarizes *Friends* series seasons by their average rating.

For the fifth query we decided to find out which are the most famous and top rated season among all seasons of Friends. From this view we might learn that the production team involved in creating *Friends* managed to conclude the series successfully.



Query 6: Professions in IMDB Database



The last Query we did was to check the number of people working in different professions in the data set. Unsurprisingly, actors are the dominant group, but interestingly, the second largest - *self* - might suggest that there is a large number of movies, with actors either playing themselves, or documentary productions with no acting involved.

5. Indexing

Indexes in PostgreSQL, simply put, are a replica of the data on the column(s) that is/are indexed. The only difference is in the data order - the replica of the data is sorted, which allows PostgreSQL to quickly find and retrieve the data.

The aim of using indexes was to enhance the database performance. In this way, the database server will be able to find and retrieve specific relations much faster. We applied the default index type- b-tree which is the most suitable to fit different situations like the IS NULL condition. From the table below we can see that the cost of the query has decreased after applying indexing. When applying indexing we have to be careful as if it is of no use, it can come at a cost with storage and maintenance.

Indexing analysis on query 2

To show the plan for a simple query on a table with a top rated movies based on their average rating and number of votes:

EXPLAIN SELECT * FROM topmovies;

	QUERY PLAN text
1	Limit (cost=201755.27..201766.94 rows=100 width=88)
2	-> Gather Merge (cost=201755.27..201943.58 rows=1614 width=88)
3	Workers Planned: 2
4	-> Sort (cost=200755.25..200757.26 rows=807 width=88)
5	Sort Key: r.averagerating DESC
6	-> Parallel Hash Join (cost=186841.35..200724.40 rows=807 width=88)
7	Hash Cond: ((r.tconst)::text = (b.tconst)::text)
8	-> Parallel Seq Scan on rating r (cost=0.00..13459.26 rows=161447 width=64)
9	Filter: (numvotes > 200000)
10	-> Parallel Hash (cost=186633.00..186633.00 rows=16668 width=56)
11	-> Parallel Seq Scan on basics b (cost=0.00..186633.00 rows=16668 width=56)
12	Filter: ((titletype)::text = 'movie'::text)

Indexing analysis on query 6

To show the plan for a simple query on a table with a all crew member based on their count in IMDb database in descending order:

EXPLAIN SELECT * FROM cat;

	QUERY PLAN text
1	Sort (cost=703980.99..703981.02 rows=12 width=15)
2	Sort Key: (count(*)) DESC
3	-> Finalize GroupAggregate (cost=703977.73..703980.77 rows=12 width=15)
4	Group Key: principals.category
5	-> Gather Merge (cost=703977.73..703980.53 rows=24 width=15)
6	Workers Planned: 2
7	-> Sort (cost=702977.71..702977.74 rows=12 width=15)
8	Sort Key: principals.category
9	-> Partial HashAggregate (cost=702977.38..702977.49 rows=12 width=...
10	Group Key: principals.category
11	-> Parallel Seq Scan on principals (cost=0.00..608761.25 rows=188...

Indexing analysis on query 3

EXPLAIN (COSTS FALSE) SELECT * FROM Q3 WHERE episodenumber = 5;

Here is the plan with cost estimates suppressed:

	QUERY PLAN text
7	Join Filter: ((e.tconst)::text = (t2.tconst)::text)
8	-> Parallel Hash Join
9	Hash Cond: ((r.tconst)::text = (e.tconst)::text)
10	-> Parallel Seq Scan on rating r
11	-> Parallel Hash
12	-> Parallel Seq Scan on episode e
13	Filter: (episodenumber = 5)
14	-> Index Scan using basics_attributes_index on basics t2
15	Index Cond: ((tconst)::text = (r.tconst)::text)
16	Filter: ((titletype)::text = 'tvEpisode'::text)
17	-> Index Scan using basics_attributes_index on basics t1
18	Index Cond: ((tconst)::text = (e.parenttconst)::text)
19	Filter: ((primarytitle = 'Friends'::text) AND ((titletype)::text = 'tvSeries'::text))

6. Conclusion

To conclude, we were successfully able to fetch the data from the IMDb dataset by importing the data from tsv files into postgresql. We segregated the data further by redefining the relationships among entities to create structured data which can fetch us useful results without data anomalies. Based on structured data we created some interesting meaningful views. The data set had some very interesting attributes that can be used to further explore more meaningful relationships.