DBM1 Project Report - Letterboxd

By Fiona Eguare & Erica Guardamagna

GENERAL INFORMATION AND DOMAIN

The aim of this report is to show the process of creating and querying a database, using data from Kaggle (https://www.kaggle.com/datasets/samlearner/letterboxd-movie-ratings-data). The data is based on Letterboxd, a social platform for sharing your reviews of films. The steps that led to the final work include the pre-processing of the data, the creation of the database, the formulation of the queries, the design of the ER schema and the presentation of results.

The domain we took into consideration corresponds to 4,885 Letterboxd movie data entries and 11,078,167 entries of ratings made by Letterboxd users. We had to truncate the movies data as the file was too large for us to perform the necessary preprocessing.

PRE-PROCESSING

The preparation of the dataset included different steps, such as the normalization of redundant data, the removal of unusably incomplete entries, the selection of interesting data, the renaming of poorly named attributes and the correction of data formatting.

The removal of incomplete entries was a necessary step, given that some of the entries had null values in most of the important attributes. We noticed that entries with null values in the original_language attribute had null values in at least 3 other columns and we agreed on deleting them from the dataset. Keeping the entries would have resulted in inefficient storage use and inaccurate query results.

The selection of interesting data included the removal of specific attributes that povided uninteresting, redundant data that would lead to longer query execution times, with no real gain. For example, the raw data contained multiple links to access each movie, and we did not think that any interesting queries could arise from the inclusion of this data.

The renaming of poorly named attributes was an important step, since the original names led to confusion and did not give a clear description of the value contained (eg. Ratings._ID -> rid).

The correction of the data implied the removal of strange terminator characters from the data, since it could not be imported into the dataset.

After the above pre-processing steps, certain redundant data became visible. To better handle this redundancy, we normalised the redundant data into another table (Movie_ids).

There was a problem concerning the access of the multi-valued attributes (genres and overview from Movies). For this reason we agreed on creating one more entity, Movie_genres (weak entity) to be able to use the data in a more efficient way. Our database is not in Normal form as we made the choice not to splot the multivalued ‘overview’ attribute as we felt that it was valuable enough as it was and would add a lot of overhead if we were to split it up like we did with the genres.

ER DIAGRAM

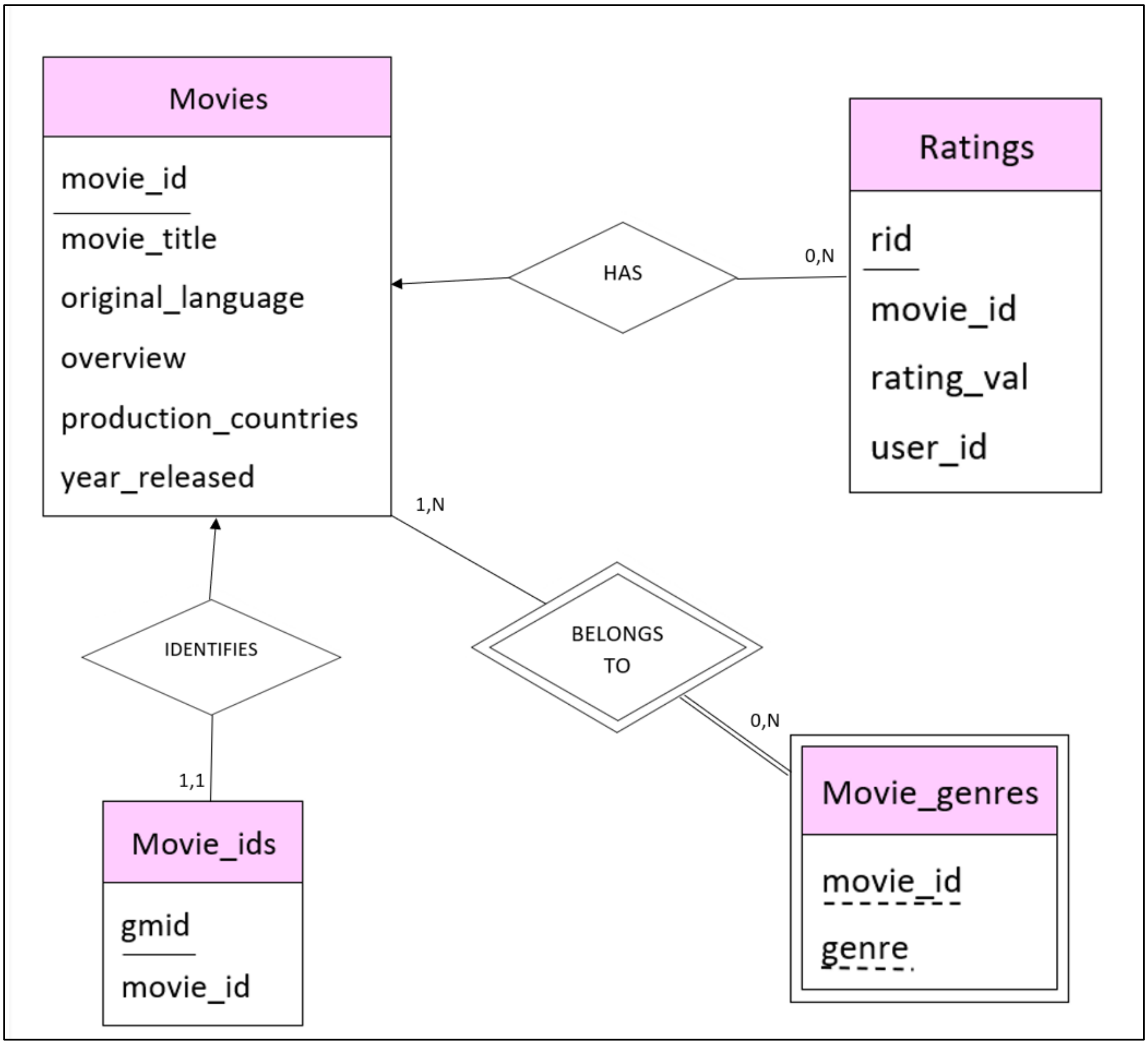


Figure 1: Final Entity Relational Schema

FORMULATION OF THE QUERIES

While formulating the queries, we looked at metadata such as the amount of a certain attribute or the average, since it would be interesting to see, for example, the average of all the ratings a certain film got, as this is value showed also on the Letterboxd platform or how many movies we can find grouped by their original language.

The queries in SQL are as following:

1) List all original languages and number of movies with each.

```
SELECT DISTINCT original_language, COUNT(original_language)
FROM movies
GROUP BY original_language
ORDER BY original_language;
```

2) List title and number or ratings for each movie rated, in descending order.

```
WITH
t1 AS
     (SELECT movie_title, movie_id
      FROM movies),
t2 AS
     (SELECT movie_id, COUNT(movie_id)
      FROM ratings
      GROUP BY movie_id),
t3 AS
     (SELECT movie_title, COUNT
      FROM t1
      JOIN t2
      ON t1.movie_id = t2.movie_id
     )
SELECT * FROM t3
ORDER BY COUNT DESC;
```

3) Count ratings with each rating value.

```
SELECT DISTINCT rating_val, COUNT(rating_val)
FROM ratings
GROUP BY rating_val
ORDER BY count;
```

4) Give the title, original language and release year for each movie that received a rating of 10.

```
WITH
t1 AS
     (SELECT DISTINCT movie_id
      FROM ratings
      WHERE rating_val = '10')
SELECT movie_title, original_language, year_released
FROM t1
JOIN movies
ON t1.movie_id = movies.movie_id
ORDER BY year_released;
```

5) List title of every movie rated by a given user, and the rating they gave.

```sql
WITH
t1 AS
    (select movie_title, movie_id
     FROM movies),
t2 AS
    (SELECT movie_id, rating_val
     FROM ratings
     WHERE user_id = 'deathproof'),
t3 AS
    (SELECT movie_title, rating_val
     FROM t1
     JOIN t2
     ON t1.movie_id = t2.movie_id
    )
SELECT * FROM t3
ORDER BY CAST(rating_val AS int) DESC;
```

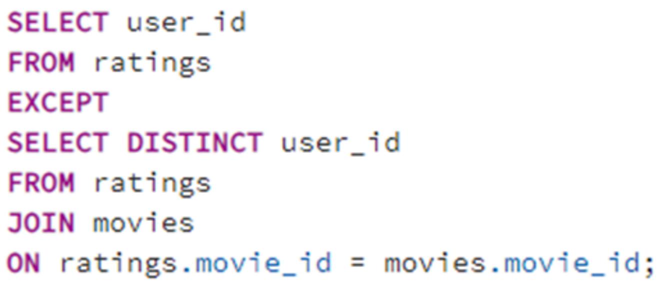
6) List ID and average rating, and number of ratings for movies with a given title.

```sql
WITH
t1 AS
    (SELECT movie_id, AVG(CAST(rating_val AS int))::numeric(10,2)
     FROM ratings
     GROUP BY movie_id),
t2 AS
    (SELECT movie_title, t1.movie_id, AVG
     FROM t1
     JOIN movies
     ON movies.movie_id = t1.movie_id)
SELECT *
FROM t2
WHERE movie_title = 'Aftermath';
```

7) List number of ratings, and average rating given by each user.

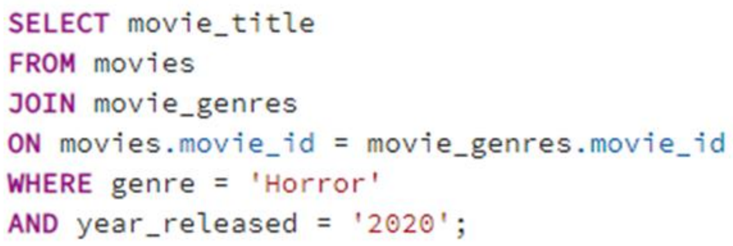
```sql
SELECT user_id, COUNT(user_id), AVG(CAST(rating_val AS int))::numeric(10,2)
FROM ratings
GROUP BY user_id
ORDER BY AVG;
```

8) List all users who didn’t rate any of the movies in the movies table.



```sql
SELECT user_id
FROM ratings
EXCEPT
SELECT DISTINCT user_id
FROM ratings
JOIN movies
ON ratings.movie_id = movies.movie_id;
```

9) List the titles of all horror movies released in 2020.



```sql
SELECT movie_title
FROM movies
JOIN movie_genres
ON movies.movie_id = movie_genres.movie_id
WHERE genre = 'Horror'
AND year_released = '2020';
```

ER DIAGRAM

The ER Schema changed a lot during the the whole process. At the beginning we had a 2 entity schema (Movies and Ratings), where the first had 8 attributes and the second 2. This was a result of the presentation of the data at the time of downloading. The particulatity of this diagram was that the Movies entity was considered the strong entity and the Ratings one a weak entity, dependent on Movies.

As we preprocessed the data, removing irrelevant and redundat attributes, and normalizing the data, we decided it would be better to create a different diagram, representing the data as 4 entities (Movies, Ratings, Movie_IDs and Movie_Genres) where the last is a weak entity of the first.

RELATIONAL ALGEBRA

We have included 2 examples of our queries under form of relational algebra.

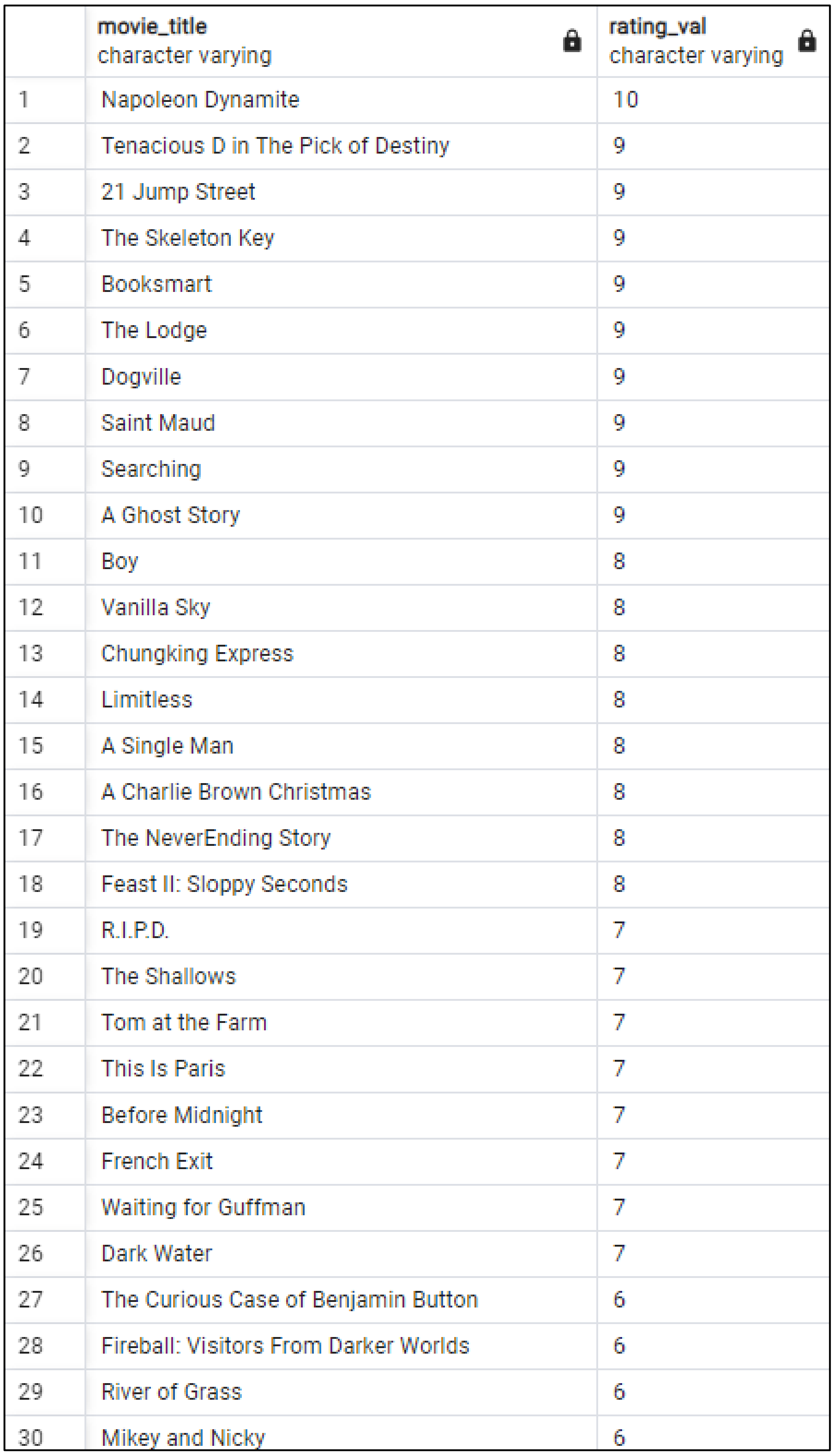
The 5th and 8th queries are as follows.

5) $\prod_{movie\_id,rating\_val}(\vartheta_{user\_id='deathproof'}(Movies \bowtie_{Movies.movie\_id=Ratinigs.movie\_id} Ratings)$

8) $\prod_{user\_id}(Ratings) - \prod_{user\_id}(Ratings \bowtie_{ratings.movie\_id\,=\,movies.movie\_id} Movies)$

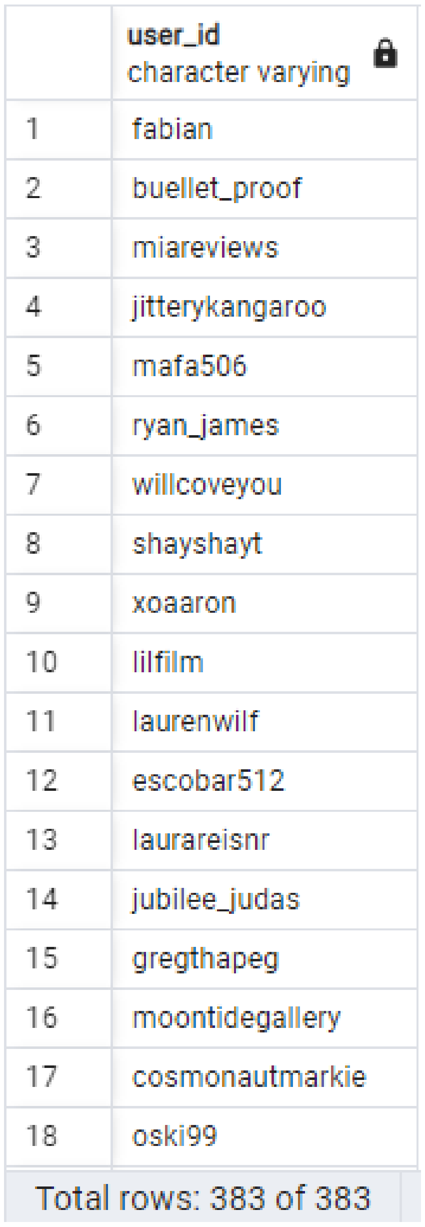
EXAMPLE OUTPUT

We have included 2 examples of our output, from query 5 and 6 as follows.



	movie_title character varying	rating_val character varying
1	Napoleon Dynamite	10
2	Tenacious D in The Pick of Destiny	9
3	21 Jump Street	9
4	The Skeleton Key	9
5	Booksmart	9
6	The Lodge	9
7	Dogville	9
8	Saint Maud	9
9	Searching	9
10	A Ghost Story	9
11	Boy	8
12	Vanilla Sky	8
13	Chungking Express	8
14	Limitless	8
15	A Single Man	8
16	A Charlie Brown Christmas	8
17	The NeverEnding Story	8
18	Feast II: Sloppy Seconds	8
19	R.I.P.D.	7
20	The Shallows	7
21	Tom at the Farm	7
22	This Is Paris	7
23	Before Midnight	7
24	French Exit	7
25	Waiting for Guffman	7
26	Dark Water	7
27	The Curious Case of Benjamin Button	6
28	Fireball: Visitors From Darker Worlds	6
29	River of Grass	6
30	Mikey and Nicky	6

Figure 2: Query 5 output



	user_id character varying
1	fabian
2	buellet_proof
3	miareviews
4	jitterykangaroo
5	mafa506
6	ryan_james
7	willcoveyou
8	shayshayt
9	xoaaron
10	lilfilm
11	laurenwilf
12	escobar512
13	laurareisnr
14	jubilee_judas
15	gregthapeg
16	moontidegallery
17	cosmonautmarkie
18	oski99

Total rows: 383 of 383

Figure 3: Query 6 output

CONCLUSION

We familiarised ourselves with the workings of SQL, as well as with the platform PGAdmin, and the library SQLite over the course of this project. We were able to successfully preprocess our chosen data into a form suitable for our database, create a database, and query the database to extract interesting data.