FINAL PROJECT REPORT

By
Fardokhtsadat Mohammadi

Table of Contents

Introduction

The aim of this project is to choose an appropriate database to store and handle big data. In this study, SQL databases are used to create a database for movies. In this paper, firstly, I describe the raw data, the data-transformation, and the tables of the database. Next, I show some queries made to extract some information. Finally, I explain why I chose MySQL databases to store the data.

Material and methods

For this project, the data is obtained from Grouplens. This dataset stores the 25,000,095 ratings and 1,093,360 tags made by 162,541 users for 62,423 movies. The data is provided in csv files. In this section, I shortly describe each datafile.

- The data file 'movies.csv' contains the columns movieId, title, genres.
- The data file 'tags.csv' contains userId, movieId, tag, timestamp. A tag is a single word or a phrase made by a user for a movie. The purpose of the tag is determined by the user. The column 'timestamp' shows the date and the time of the creation of the tag. The timestamp represents seconds in regard to midnight coordinated universal time of January 1, 1970.
- The data file 'ratings.csv' contains userId, movieId, rating, and timestamp. Ratings are on a scale of 0 to 5 with the incrementation of 0.5. A user can rate a movie based on this scale.
- The 'links.csv' includes the identifiers used by movielens.org, imdb.com, and themoviedb.org presented in three columns of movieId, imdbId, tmdbId respectively.
- Tag-genome is provided in two files of 'genome-scores.csv' and 'genome-tags.csv'. Tag-genome is computed using a machine-learning algorithm and it shows to what degree movies show particular properties represented by tags. The data file 'genome-scores.csv' contains three columns of movieId, tagId, and relevance. The file 'genome-tags.csv' contains tagId and tag.

In this project, I use MySQL to store the data. The schema of the database is shown below:

```
+--------------------+
| Tables_in_moviesdb |
+--------------------+
| genome_scores      |
| genome_tags        |
| genres             |
| links              |
| ratings            |
| tags               |
| titles             |
| users              |
+--------------------+
```

To follow the ACID properties of SQL databases, I split the datafile 'movies.csv' into two tables of 'titles' and 'genres'. The table 'titles' has two columns of 'movieId', 'title', and year_of_movie. 'movieId' is the primary-key of the table and it is auto-increment. The table 'titles' is presented in figure.1.

Field	Type	Null	Key	Default	Extra
movieId	int	NO	PRI	NULL	auto_increment
title	varchar(300)	NO		NULL	
year_of_movie	int	YES		NULL	

Figure 1

The table 'genres' has three columns of 'id', 'movieId', and 'genres'. 'id' is the primary-key of this table and it is auto-incremented. 'movieId' is a foreign-key referring to the 'movieId' of table 'titles'. The table 'genres' is shown in figure.2.

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
movieId	int	NO	MUL	NULL	
genres	varchar(30)	NO		NULL	

Figure 2

The data file 'links.csv' is stored as the table 'links' in the database. This table has four columns of 'id', 'movieId', 'imdbId', and 'tmdbId'. 'id' is the primary-key of this table and it is auto-incremented. 'movieId' is a foreign-key referring to the 'movieId' of table 'titles'. The table 'links' is described in figure.3.

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
movieId	int	NO	MUL	NULL	
imdbId	int	YES		NULL	
tmdbId	int	YES		NULL	

Figure 3

The file 'tags.csv' is imported as the table 'tags'. Tags contains five columns of 'id', 'userId', 'movieId', 'tag', and 'timestamp'. 'id' is the primary-key of this table and it is auto-incremented. 'movieId' and 'userId' are foreign-keys referring to the 'movieId' of table 'titles', and 'userId' of table 'users' respectively. The table 'tags' is described in figure.4.

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
userId	int	NO	MUL	NULL	
movieId	int	NO	MUL	NULL	
tag	varchar(60)	NO		NULL	
timestamp	int	NO		NULL	

Figure 4

The data file 'ratings.csv' is imported as the table 'ratings' to the SQL database. This table has five columns of 'id', 'userId', 'movieId', 'ratings', and 'timestamp'. 'id' is the primary-key of this table and it is auto-incremented. 'movieId' and 'userId' are foreign-keys referring to the 'movieId' of table 'titles', and 'userId' of table 'users' respectively. The table 'ratings' is described in figure.5.

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
userId	int	NO	MUL	NULL	
movieId	int	NO	MUL	NULL	
rating	decimal(2,1)	NO		NULL	
timestamp	int	NO		NULL	

Figure 5

The data files 'genome_tags.csv' and 'genome_scores.csv' are imported as tables 'genome_tags' and 'genome_scores'. The table 'genome_tags' has two columns of 'tagId' and 'tag' with 'tagId' being the primary-key of the table. The table 'genome_tags' is shown in figure.6.

Field	Type	Null	Key	Default	Extra
tagId	int	NO	PRI	NULL	auto_increment
tag	varchar(100)	NO		NULL	

Figure 6

The file 'genome_scores.csv' is stored as the table 'genome_scores' in the database. This table includes four columns of 'id', 'movieId', 'tagId', and 'relevance'. 'id' is the primary-key of the table, and it is auto-incremented. 'movieId' and 'tagId' are foreign-keys referring to the 'movieId' of table 'titles' and 'tagId' of the table genome_tags' respectively. The table 'genome_scores' is described in figure.7.

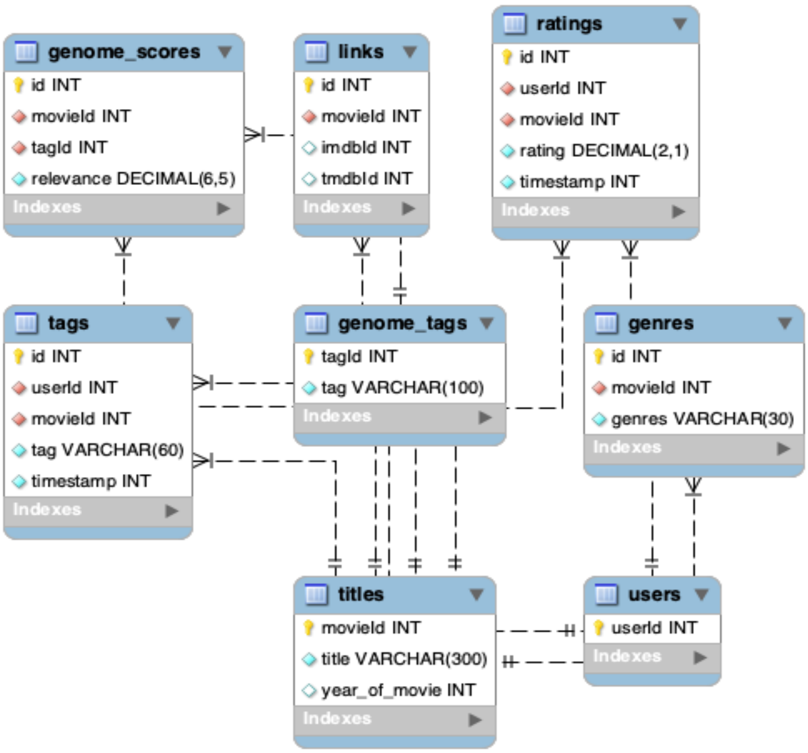
Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
movieId	int	NO	MUL	NULL	
tagId	int	NO	MUL	NULL	
relevance	decimal(6,5)	NO		NULL	

Figure 7

All the data-transformation is done using the programming language R, and it can be found in the following GitHub repository https://github.com/fardokhtsadat/Advanced-Database-Systems . In addition, the SQL commands used to create the database and import the data can be found in the above-mentioned repository.

ER-diagram of the database:

The database 'movies DB' is described using an ER-diagram. In the ER-diagram below, the primary-keys are shown with the logo , and the foreign-keys are shown with the logo . In addition, the data type for each column is mentioned.



Example Queries

In this section, I make some queries to extract some information from the database. The time required for each query is also recorded.

1. Make a query to find movies titles of movies under genres comedy with rating higher than 4:

```
SELECT DISTINCT titles.title
FROM moviesDB.titles
INNER JOIN genres ON titles.movieId = genres.movieId
INNER JOIN ratings ON ratings.movieId = genres.movieId
WHERE genres.genres = 'Comedy' AND ratings.rating > 4;
```

8925 rows are returned and the time taken is 4,164 seconds. The first 5 rows of the result are shown below:

```
+------------------------------------------+
| title                                    |
+------------------------------------------+
| Toy Story (1995)                         |
| Grumpier Old Men (1995)                  |
| Waiting to Exhale (1995)                 |
| Father of the Bride Part II (1995)       |
| Sabrina (1995)                           |
```

2. Under which genres is the movie Toy Story 1995 categorized?

```
SELECT DISTINCT genres.genres
FROM moviesDB.titles
INNER JOIN moviesDB.genres ON titles.movieId = genres.movieId
WHERE titles.title = 'Toy Story (1995)';
```

5 rows are returned and the time taken is 0.015 seconds. The full result is shown below:

```
+-----------+
| genres    |
+-----------+
| Adventure |
| Animation |
| Children  |
| Comedy    |
| Fantasy   |
+-----------+
```

3. make a query to obtain titles of movies with the tag epic.

```
SELECT titles.title, tags.tag
FROM moviesDB.titles
INNER JOIN tags ON titles.movieId = tags.movieId
WHERE tags.tag = 'epic';
```

1054 rows are returned and the time taken is 0.129 seconds. The first 5 rows of the result are shown below:

```
+-----------------------------------------------------+------+
| title                                               | tag  |
+-----------------------------------------------------+------+
| Braveheart (1995)                                   | epic |
| Troy (2004)                                         | Epic |
| Hero (Ying xiong) (2002)                            | Epic |
| Lord of the Rings: The Fellowship of the Ring, The (2001) | epic |
| Lord of the Rings: The Two Towers, The (2002)       | Epic |
```

4. Make a query to find movieIds which do not have a tmdbId link:

SELECT movieId, tmdbId
FROM moviesDB.links
WHERE tmdbId IS NULL;

181 rows are returned and the time taken is 0.0096 seconds. The first 5 rows of the result are shown below:

movieId	tmdbId
721	NULL
730	NULL
769	NULL
770	NULL
791	NULL

5. Make a query to retrieve the movie-titles which were tagged between '2015-01-01' and '2015-01-31' and order the result by the date ascending:

SELECT userId, movieId, tag, TIME(from_unixtime(tags.timestamp)) AS creation_time, DATE(from_unixtime(tags.timestamp)) AS creation_date
FROM moviesDB.tags
WHERE from_unixtime(tags.timestamp) between '2015-01-01' and '2015-01-31'
ORDER BY creation_date ASC;

7741 rows are returned and the time taken is 0.74 seconds. The first 5 rows of the result are shown below:

userId	movieId	tag	creation_time	creation_date
222759	86815	inspiring	00:42:15	2015-01-01
222759	86815	must see!	00:42:15	2015-01-01
222759	86815	true story	00:42:15	2015-01-01
99921	74541	Canada	00:43:17	2015-01-01
140201	104239	documentary	00:19:31	2015-01-01

6. Get the tmdbId and the imdbId link for the movie 'Troublemaker (1988)':

SELECT title, CONCAT('https://movielens.org/movies/', links.tmdbId) as tmdbId_link, CONCAT('https://movielens.org/movies/', links.imdbId) as imdbId_link
FROM moviesDB.links
INNER JOIN titles ON titles.movieId = links.movieId
WHERE titles.title = 'Troublemaker (1988)';

1 row is returned and the time taken is 0.014 seconds. The result is shown below:

title	tmdbId_link	imdbId_link
Troublemaker (1988)	https://movielens.org/movies/286545	https://movielens.org/movies/179493

7. Make a query to retrieve the average rating for the movie ‘White Dog (1982)’:

```sql
SELECT titles.title, AVG(rating) 'Average rating'
FROM moviesDB.ratings
INNER JOIN titles ON titles.movieId = ratings.movieId
WHERE titles.title = 'White Dog (1982)';
```

1 row is returned and the time taken is 0.016 seconds. The result is shown below:

```
+------------------+----------------+
| title            | Average rating |
+------------------+----------------+
| White Dog (1982) |        3.35357 |
+------------------+----------------+
```

Why SQL databases

SQL databases have many features which make it advantageous over other database types and makes it a good option for my database.

SQL servers use a concept abbreviated as ACID when evaluating databases. ACID is an acronym for Atomicity, Consistency, Isolation, and Durability. **Atomicity** guarantees that each transaction is treated as a single "unit", which either succeeds completely, or fails completely. **Consistency** means that integrity constraints must be maintained so that the database is consistent before and after the transaction. **Isolation** ensures that multiple transactions can occur concurrently without leading to the inconsistency of database state. **Durability** ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. The **ACID** properties provide a mechanism to ensure correctness, consistency, and avoid duplication of data in a database.

Moreover, I chose SQL databases because it is very easy to import, manage, and make queries. In addition, users can quickly and efficiently retrieve a large amount of records from a database.

Besides, choosing SQL databases gives me an opportunity to practice my SQL knowledge and helps me to put my knowledge into practice.

Acknowledgment

I would like to express my special thanks and gratitude to my teachers Mr. PhDr. Prokýšek, and Mr. Mgr. Geyer. I enjoyed the course 'Advanced Database Systems' very much, and I believe this course helped me to broaden my knowledge in the area of database systems.

References

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19. https://doi.org/10.1145/2827872