Dominic Fernandez

Computer Science

CS 333 - Intro to Database Systems

Spring 2021

Design, Load, and Explore a Movies Database

Chapter 1: Project Description

A) Goal of the Project

The purpose of this project is to logically and efficiently implement a database that houses a data set pertaining to movies from the online movie recommender, MovieLens. This project consists of 3 main phases:

1) Exploring the data and designing the database
2) Building the database
3) Using the database with efficient and optimized SQL queries

In the first phase, the data set provided from MovieLens is to be examined. The data is not going to be in a format that can simply be imported into the database, so some data processing or ETL must be performed prior to passing the data to SQL queries and being persisted in the database. For this project, a python script will be used to translate the data to be readable by SQL. In addition to processing the data, the requirements for the database are to be examined in the first phase. These requirements are to provide a high level understanding and shed light on concerns such as why the movie data is being stored, how the movie data is being stored, and how the data is to be used. Once the requirements have been established, the database will be ready to be designed. All of the necessary entities, tables, attributes, cardinalities, and relationships will be defined and visually depicted through the use of an E/R diagram.

In the second phase, the database will be built. Initially, the DDL subset of SQL statements will be used to establish the schema of the database that was determined from the design portion of the first phase. Once these empty tables have been created, the DML portion of SQL will be used to populate the data into the tables. After all the data has been loaded into the database, a series of simple SQL statements will be performed on the tables to ensure that the data was loaded correctly and corresponds to the E/R model from the first phase.

In the third phase, the database will be further tested and used. High level, English queries and scenarios will be provided and translated into efficient SQL queries that will perform successful queries with optimal runtime and minimal memory usage. In addition, a mixture of schema changes, computations, and new relationships will be performed to experiment other possible database designs and further test the efficiency of the queries.

B) Data Exploration

The raw data provided for this project are three .txt files: ‘movies.txt’, ‘ratings.txt’, and ‘tags.txt’, each of which are described below.

‘movies.txt’ contains a large list of movies with each line representing a single movie. Each movie in the .txt file is defined with the syntax: *MovieId: Title: Genres*, where *Genres* is a pipe-separated list that has a limit of a fixed number of genres. Thus, ‘movies.txt’ can be considered to be its own entity set that houses data pertaining to movies that are uniquely identified by MovieId.

‘ratings.txt’ contains 10,000,054 ratings that are applied to 10,681 movies by 71,567 users. Each rating in the .txt file is defined with the syntax: *UserId: MovieId: Rating: Timestamp*, where *Rating* is 5-star based and *Timestamp* is the number of seconds since 1/1/70, UTC. However, ‘ratings.txt’ cannot necessarily be considered to stand as its own entity set, for it does not have primary keys that can uniquely identify each entity. Though it may seem that UserId and MovieId can provide a unique identification for rating entities, ratings are dependent on those attributes and likely inherits it from other entity sets. Thus, ‘ratings.txt’ should not be considered an entity set, but a relationship set.

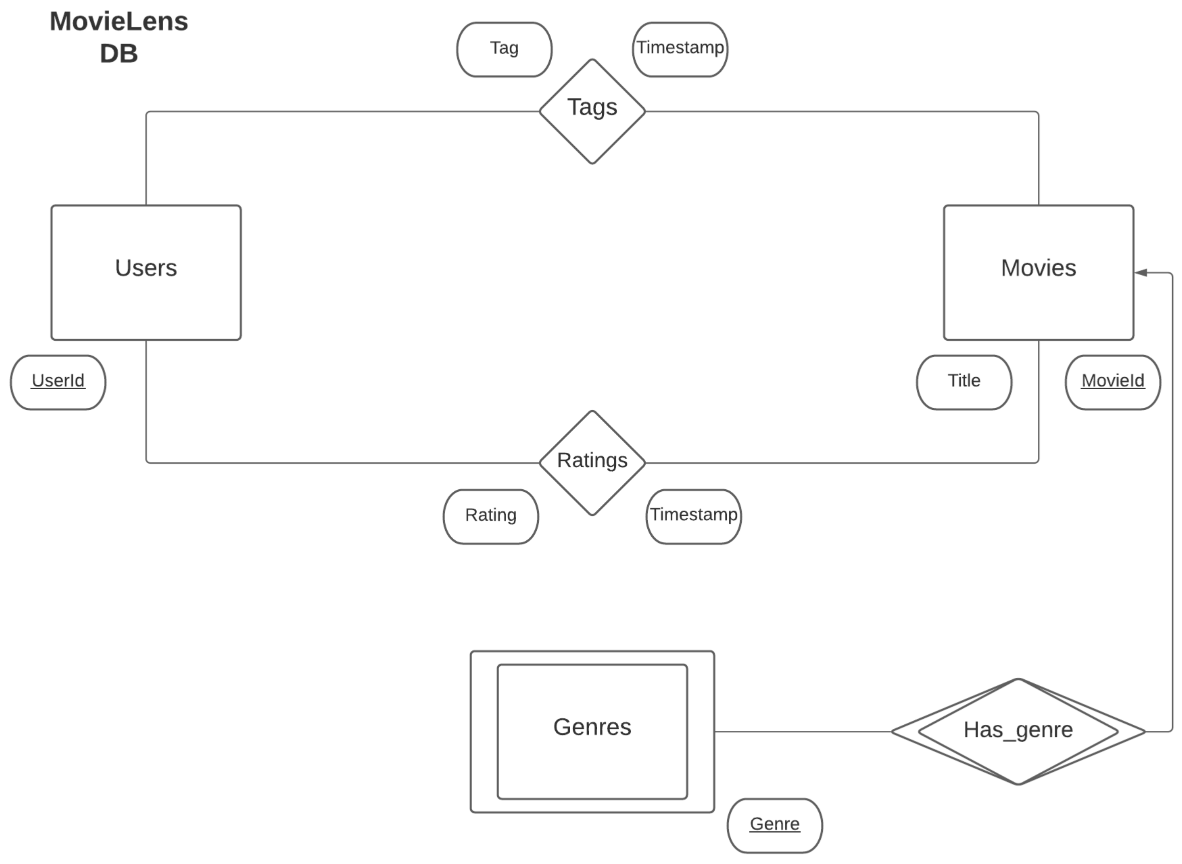
‘tags.txt’ contains 95,580 tags that are also applied to 10,681 movies by 71,567 users. Each tag in the .txt file is defined with the syntax: *UserId: MovieId: Tag: Timestamp*, where *Tag* is a single user-determined word or phrase and *Timestamp* is the number of seconds since 1/1/70, UTC. Similar to ‘ratings.txt’, tag entities are not to be considered unique, for the attributes UserId and MovieId are inherited from other entity sets. Thus, ‘tags.txt’ should not be considered an entity set, but a relationship set.

With ratings, tags, and movies defined, this leaves two other pieces of data from the data set that have not yet been classified: genres and users. Genres are found in ‘movies.txt’, and each movie can be associated with one or more genres. Users are found in every ‘ratings.txt’ and ‘tags.txt’, though the only identifying piece of information is a UserId.

Since one or many genres are to be associated with each movie, in ‘movies.txt’, genres should be stored in its own entity set, and not with the entity set of movies, as that would make the data structure more nested and therefore harder and less efficient to perform queries on. Therefore, a genres entity set can be defined, but it will have to be a weak entity set supported by the Movies entity set, since genres are generic to movies. In regard to users, since users have a unique UserId attribute, users can be stored in its own entity set, with its primary key being UserId

Chapter 2: Database Design

A) <u>E/R Diagram</u>



B) <u>Logical Schema</u>

The E/R diagram above depicts the way the data from MovieLens will be loaded and related in the database. The entity sets are Users and Movies. The relationship sets between Users and Movies are Tags and Ratings. Genres is a weak entity set, and is supported by Movies through the supporting relationship set, Has_genre. The cardinality between the entity sets Users and Movies are many-to-many through both Ratings and Tags, and many-to at least one from Genres to Movies through Has_genre. In other words, this schema will allow many users to tag and/or rate many movies, and will enforce that many genres to be associated with at least one movie. From a high-level perspective, this seems to be logical, as it makes sense to allow many users to rate or tag many movies, and it would not make sense for a movie to not have a genre, which is why the constraint, "many to at least one" is set. In addition, the data seems to support this database schema, as it suggests that many tags and ratings are applied to many movies by users. Therefore, each user can be expected to have one or many ratings and tags on one or more movies.

In terms of the movies and genres, the data suggests that one movie can have multiple genres from the 19 that are available from the data set. Since it does not really make sense to have a movie without a genre, the database will enforce the constraint that a movie must have at least one genre. A more in depth look at each table and their attributes is shown below.

Users(UserId)

Movies(MovieId, Title)

Tags(UserId, MovieId, Tag, Timestamp)

Ratings(UserId, MovieId, Rating, Timestamp)

Genres(Genre)

Has_genre(MovieId, Genre)

The Users table seems very empty, as it is just holding a UserId as an attribute. However, the Users table will remain as an entity set because the UserId attribute is unique, which allows each entity in the table to be uniquely identified. In addition, the Users table can be expanded on in the future and accept more attributes such as an email address, home address, telephone number, and can be linked with other entity sets through relationship sets due to its uniqueness.

Chapter 3: Load Data and Test the Database

A) Load Data

There are two steps to load the data into the database:

1. A series of python scripts are used to parse the data from each of the .txt files and save them as CSV files, where the order of the values in each CSV file matches the schema of the corresponding table that the CSV file will be loaded into. The python scripts responsible for this parsing can be found in the parsing_scripts directory of the project's root folder.
2. Predefined SQL statements from the dcfernandez-code-phase2.sql file in the sql_code directory under the project's root folder are executed to create the tables and then load the data from the CSV files into each of those tables. The COPY statement is used to read the CSV files by the "," delimiter and load the values into their respective tables.

*See the README.md in the project's root folder for detailed information at https://github.com/dcfernandez1023/MoviesDB

B) Test the Database

Now that the data has been loaded, the next step is to test that the data was loaded as expected. The queries that were used to test the data can be found in the Github repository https://github.com/dcfernandez1023/MoviesDB in the root folder of the project that is titled

dcfernandez-code-results-phase2.pdf.