Front page

Full Name: Cara Lu Hui Cao

Your Major: Computer Science

Course No and Title: CS333 - Introduction to Database Systems

Semester and Year: Spring 2022

Title of the project: Design, Load and Explore a Movies database

Chapter 1: Project Description

a) Goal of the project

Write a paragraph to describe the several parts of the project: loading data, designing and building a database, testing the database with sample queries, exploring the database, querying the database and optimizing queries.

The dataset provided by the professor is loaded from https://movielens.org/. It consists of three files in .txt: movies, ratings and tags. The goal of this project is to design, build and make use of a database that is made up of the data provided in this dataset. The design of the database should include an E/R diagram, several logical schemas that clearly define the relationship between each data table and the (key) attributes in each table. The database design should minimize the use of weak entity sets, but make good use of entity sets and relationship sets. Overall, the database design should reduce redundancy, minimize the chance of having inconsistencies in the database. At the end, users should be able to test the database with queries.

b) Data Exploration

PROJECT DATASET

Describe your dataset: input files, file size, types of records (.txt), attributes identified, and more.

There are three input files: movies.txt, ratings.txt and tags.txt.

- Movies.txt (489.1KB):
    - Each line of this file represents one movie
    - Attributes: MovieID, Title, Genres
        - MovieID is the real MovieLens id
        - Movie titles is the title from IMDB

        - Genres examples are Action, Adventure, Animation, etc.
    - Format: `MovieID::Title::Genres`
- Ratings.txt (224.21 MB):
    - Each line of this file represents one rating of one movie by one user
    - Attributes: UserID, MovieID, Rating, Timestamp
    - Format: `UserID::MovieID::Rating::Timestamp`
    - The lines within this file are ordered first by UserID, then within user, by MovieID
    - Ratings are made on a 5-star scale, with half-star increments
    - Timestamps represents seconds since midnight UTC od Jan 1st, 1970
- Tags.txt (3.14 MB):
    - Each line of this file represents one tag applied to one movie by one user
    - Attributes: UserID, MovieID, Tag, Timestamp
    - Format: `UserID::MovieID::Tag::Timestamp`
    - The lines within this file are ordered first by UserID, then within user, by MovieID
    - Tags are generated metadata about movies by each user
    - Timestamps' definition is the same as Timestamps in Ratings.txt

Chapter 2: Database Design

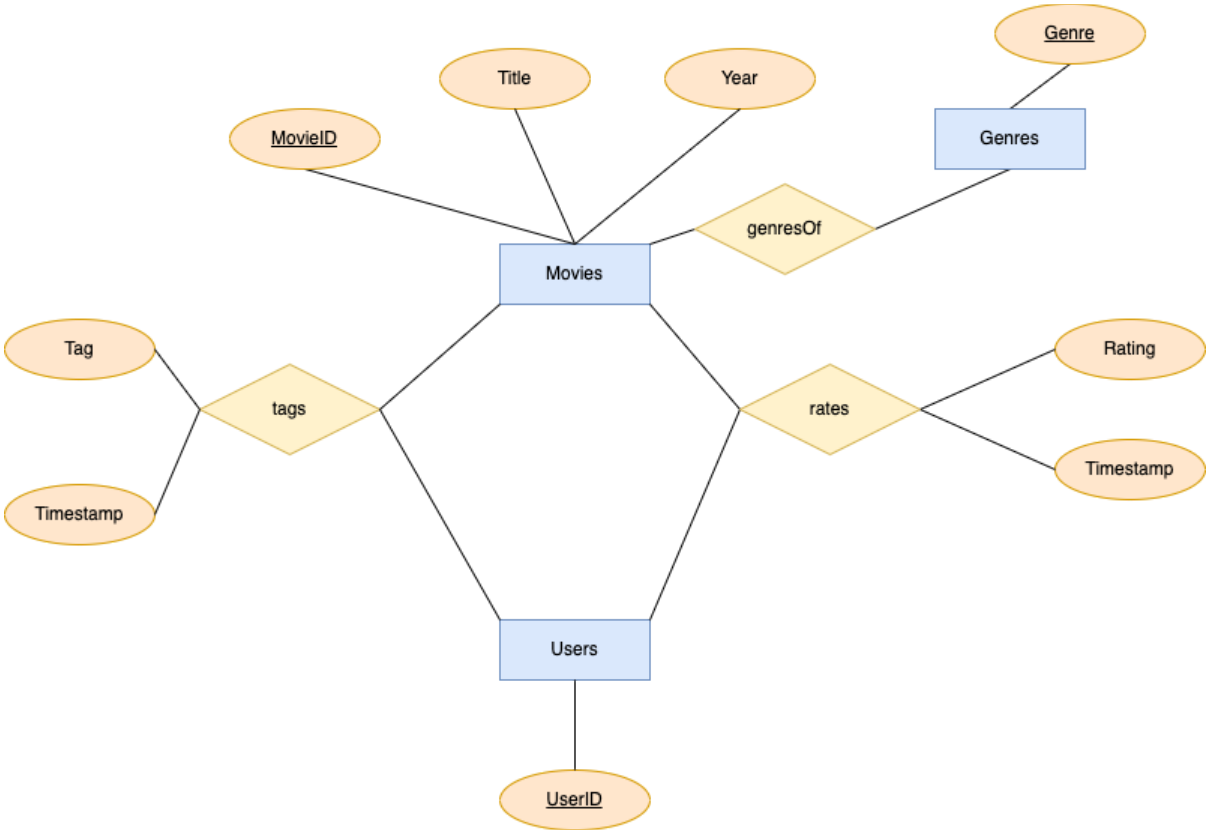
a) E/R Diagram

Describe the entities and relationships you discovered during data exploration. Give details about their attributes, how they are related (relationship types), keys.

There should be two entities: Movies and Users. The key to those entities should be movieID and UserID respectively.

The relationship between those two entities should be “tags” and “rates”, since the user tags the movies, and the user rates the movies. The “tags” relationship should have MovieID, UserID, tag and Timestamp as attributes, where MovieID and UserID combined is the primary key for this relation. The “rates” relationship should have MovieID, UserID, ratings and Timestamp as attributes, where MovieID and UserID combined is the primary key for this relation.

The relationship between Movies entity and Users entity is a many-to-many relationship. Since one user can rate or tag many movies, and one movie can be tagged or rated by many users.



b) Logical Schema

List the tables that are created from the E/R diagram and provide their detailed schemata: table names, attributes, keys.

Movies(MovieID, Title, Year)

Genres(Genre)

genresOf(MovieID, Genre)

Users(UserID)

tags(MovieID, UserID, tag, timestamp)

rates(MovieID, UserID, rating, timestamp)

Chapter 3: Load Data and test your Database

a) Load Data

EDIT YOUR INPUT FILES

Write SQL queries to:

i) create new database: "moviesdb".

```
create database moviesdb;
```

ii) create the tables of your logical schema.

```
CREATE TABLE movies (
        id integer PRIMARY KEY,
        title varchar(200),
        year integer
);


CREATE TABLE genres (
        genre varchar(60),
        PRIMARY KEY(genre)
);
```

```
CREATE TABLE users (
        id integer PRIMARY KEY
);

CREATE TABLE ratings (
        userid integer,
        movieid integer,
        rating real,
        time bigint,
        PRIMARY KEY(userid, movieid)
);

CREATE TABLE tags (
        userid integer,
        movieid integer,
        tag varchar(255),
        time bigint,
        PRIMARY KEY(userid, movieid, tag)
```

```
);

CREATE TABLE has_genre (
        movieid integer,
        gen_title varchar(100),
        PRIMARY KEY(movieid, gen_title)
);
```

iii) load the data from your input (.txt) files into your tables. Edit your input files so that they are readable in Postgres. Read these notes on file editing.

```
\copy ratings FROM 'ratings.txt' (DELIMITER(':'));

\COPY movies FROM 'moviesTable.csv' (DELIMITER(':'));

\COPY genres FROM 'genresTable.csv';

\COPY tags FROM 'tags.csv' (DELIMITER(':'));
```

```
\COPY has_genre FROM 'has_genre.csv' (DELIMITER(':'));
```

b) Test your database

TEST YOUR DATABASE

Verify that your data were successfully loaded into your tables. Follow the instructions on this page.

```
moviesdb_test=# \d
          List of relations
 Schema |   Name    | Type  |  Owner
--------+-----------+-------+----------
 public | genres    | table | postgres
 public | has_genre | table | postgres
 public | movies    | table | postgres
 public | ratings   | table | postgres
 public | tags      | table | caracao
 public | users     | table | postgres
(6 rows)

moviesdb_test=# \d genres
                  Table "public.genres"
 Column |          Type          | Collation | Nullable | Default
--------+------------------------+-----------+----------+---------
 title  | character varying(200) |           | not null |
Indexes:
    "genres_pkey" PRIMARY KEY, btree (title)

moviesdb_test=# \d movies
                  Table "public.movies"
 Column |          Type          | Collation | Nullable | Default
--------+------------------------+-----------+----------+---------
 id     | integer                |           | not null |
 title  | character varying(200) |           |          |
 year   | integer                |           |          |
Indexes:
    "movies_pkey" PRIMARY KEY, btree (id)

moviesdb_test=# \d ratings
              Table "public.ratings"
 Column  |  Type   | Collation | Nullable | Default
---------+---------+-----------+----------+---------
 userid  | integer |           | not null |
 movieid | integer |           | not null |
```

```
 rating  | real    |           |          |
 time    | bigint  |           |          |
Indexes:
    "ratings_pkey" PRIMARY KEY, btree (userid, movieid)

moviesdb_test=# \d tags
                      Table "public.tags"
 Column  |          Type          | Collation | Nullable | Default
---------+------------------------+-----------+----------+---------
 userid  | integer                |           | not null |
 movieid | integer                |           | not null |
 tag     | character varying(255) |           | not null |
 time    | bigint                 |           |          |
Indexes:
    "tags_pkey" PRIMARY KEY, btree (userid, movieid, tag)

moviesdb_test=# \d users
               Table "public.users"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
Indexes:
    "users_pkey" PRIMARY KEY, btree (id)

moviesdb_test=# \d has_genre
                     Table "public.has_genre"
  Column   |          Type          | Collation | Nullable | Default
-----------+------------------------+-----------+----------+---------
 movieid   | integer                |           | not null |
 gen_title | character varying(100) |           | not null |
Indexes:
    "has_genre_pkey" PRIMARY KEY, btree (movieid, gen_title)

moviesdb_test=# select count(*) from genres;
 count
-------
    18
(1 row)

moviesdb_test=# select count(*) from movies;
 count
-------
 10681
(1 row)

moviesdb_test=# select count(*) from ratings;
  count
----------
 10000054
(1 row)

moviesdb_test=# select count(*) from tags;
 count
-------
 95580
(1 row)
```

```
moviesdb_test=# select count(*) from users;
 count
-------
 71567
(1 row)

moviesdb_test=# select count(*) from has_genre;
 count
-------
 21564
(1 row)

moviesdb_test=# select * from movies limit 5;
 id |            title            | year
----+-----------------------------+------
  1 | Toy Story                   | 1995
  2 | Jumanji                     | 1995
  3 | Grumpier Old Men            | 1995
  4 | Waiting to Exhale           | 1995
  5 | Father of the Bride Part II | 1995
(5 rows)

moviesdb_test=# select count(title) from movies;
 count
-------
 10681
(1 row)

moviesdb_test=# select * from movies order by year desc limit 5;
  id   |                   title                    | year
-------+--------------------------------------------+------
 55830 | Be Kind Rewind                             | 2008
 56949 | 27 Dresses                                 | 2008
 53207 | 88 Minutes                                 | 2008
 55603 | My Mom's New Boyfriend                     | 2008
 57326 | In the Name of the King A Dungeon Siege Tale | 2008
(5 rows)

moviesdb_test=# select * from movies order by year limit 5;
  id   |            title             | year
-------+------------------------------+------
  7065 | Birth of a Nation, The       | 1915
  7243 | Intolerance                  | 1916
 62383 | 20,000 Leagues Under the Sea | 1916
 48374 | Father Sergius (Otets Sergiy) | 1917
  8511 | Immigrant, The               | 1917
(5 rows)

moviesdb_test=# select count(year) from movies;
 count
-------
 10681
(1 row)

moviesdb_test=# select count(year) from movies where year = 0;
 count
-------
```

```
    0
(1 row)

moviesdb_test=# select count(year) from movies where year > 1500;
 count
-------
 10681
(1 row)
```

Can you test the cases where there is no genre associated with a movie (no genres listed case)?

```
moviesdb_test=# select count(*) from has_genre where gen_title = null;
 count
-------
    0
(1 row)
```

Continue to check the values of the other attributes in a similar way and extract statistics. In particular, ***write SQL queries*** for the questions below, and ***attach your results***:

1) Find unknown or invalid data in any of the attributes for all of the tables, movies, ratings, tags, users, genres.

```
moviesdb_test=# select count(*) from movies where title = null or year = null;
 count
-------
    0
(1 row)

moviesdb_test=# select count(*) from ratings where rating = null or time = null;
 count
-------
    0
(1 row)

moviesdb_test=# select count(*) from tags where tag = null or time = null;
 count
-------
    0
(1 row)

moviesdb_test=# select count(*) from genres where title = null;
 count
-------
    0
```

```
(1 row)

moviesdb_test=# select count(*) from users where id = null;
 count
-------
     0
(1 row)
```

2) Find the distribution of the values for attribute "year" of table "movies".

```
moviesdb_test=# select year, count(*) from movies group by year order by year;
 year | count
------+-------
 1915 |     1
 1916 |     2
 1917 |     2
 1918 |     2
 1919 |     4
 1920 |     5
 1921 |     3
 1922 |     7
 1923 |     6
 1924 |     6
 1925 |    10
 1926 |    10
 1927 |    19
 1928 |    10
 1929 |     7
 1930 |    15
:
```

3) Find the distribution of the movies across different decades.

```
moviesdb_test=# select decade, count(*) from (select year, year/10*10 as decade from movies) as foo group by decade
order by decade;
 decade | count
--------+-------
   1910 |    11
   1920 |    83
   1930 |   230
   1940 |   379
   1950 |   521
   1960 |   690
   1970 |   784
   1980 |  1712
   1990 |  3022
   2000 |  3249
(10 rows)
```

4) Find the distribution of the genres across the movies.

```
moviesdb_test=# select gen_title as genre, count(*) from has_genre group by gen_title;
       genre        | count
```

```
--------------------+-------
 IMAX               |    29
 Crime              |  1118
 Animation          |   286
 Documentary        |   482
 Romance            |  1685
 Mystery            |   509
 Children           |   528
 Musical            |   436
 Film-Noir          |   148
 Fantasy            |   543
 Horror             |  1013
 Drama              |  5339
 Action             |  1473
 (no genres listed) |     1
 Thriller           |  1706
 Western            |   275
 Sci-Fi             |   754
 Comedy             |  3703
 Adventure          |  1025
 War                |   511
(20 rows)
```

5) Find the distribution of the ratings values (how many movies were rated with 5, how many with 4, etc.).

```
moviesdb_test=# select rating, count(*) from ratings group by rating order by rating;
 rating |  count
--------+---------
    0.5 |   94988
      1 |  384180
    1.5 |  118278
      2 |  790306
    2.5 |  370178
      3 | 2356676
    3.5 |  879764
      4 | 2875850
    4.5 |  585022
      5 | 1544812
(10 rows)
```

6) Find how many movies have:

i. no tags, but they have ratings

```
 moviesdb_test=# select count(*) from (select movieid from ratings except select movieid from tags) as foo;

 count
-------
  3080
```

(1 row)

ii. no ratings, but they have tags

```
moviesdb_test=# select count(*) from (select movieid from tags except select movieid from ratings) as foo;
 count
-------
     4
(1 row)
```

iii. no tags and no ratings

```
moviesdb_test=# select count(*) from (select distinct id from movies) as foo except
(select count(*) from (select movieid from ratings union select movieid from tags) as fo
)
;
 count
-------
(0 rows)
```

iv. both tags and ratings

```
  moviesdb_test=# select count(*) from (select distinct movieid from ratings intersect select distinct movieid from tags) as
foo;

 count
-------
  7597
(1 row)
```

Note: The above numbers in the 4 cases should sum up to the total number of movies, that is. 10681.

Chapter 4: Query the database and Optimize the Queries

Run these QUERIES.

Use any indexes you may find useful and report:

a) why you chose these indexes and

b) how did they help with the running time of your queries

Chapter 5: Discussion

Discuss your choices and observations in every step of the project:

- any assumptions you made in your E/R design

- any constraints you discovered (e.g. types of relationships)

- errors/duplicates/redundancy you encountered when testing your database

- the percentage of unknown values in your attributes

- any benefit from using indexes and in what cases they helped

- alternatives that you considered to improve the run time of your operations

- challenges in implementing any of the project parts