Full Name: Jenny Tang

Your Major: CS

Course No and Title: CS 333 Intro to Database Systems

Semester and Year: Spring 2023

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 Goal is to build a database from scratch by creating tables, cleaning txt files, populating extra tables from the given table and loading input txt files into these empty tables. We then draw relationships between these tables and build a schema. Eventually we want to use the tables to answer any questions that users are asking, and time these queries to see how to optimize the queries.
Loading data: check the size of the data,

Cleaning data: clean null values, duplicates and overwritten data(ex: user rate the same movie twice)

Design database: use an E/R diagram to understand the relationship between tables and which tables are entities and which are relationships.

Create tables: based on the E/R diagram, we decide which tables we need in the database and create them, and copy data over to the tables

Populate extra tables: some tables need to be created to be queried easily later.

Test tables: query tables to test if the numbers are expected

Optimize queries: based on the query time, we can write better queries to save time

b) Data Exploration

Input file, size, type, attributes and number of records:

File name	File size	File type	Attributes	Number of records
movies.txt	501 KB	.txt	movieid, title, year, genres	10681
ratings.txt	235.1MB	.txt	movieid, userid, rating, timestamp	10000054
tags.txt	3.3 MB	.txt	movieid, userid, tag, timestamp	95580

Other insights:

there are 71567 distinct users from both ratings table and tags table

there are 10681 distinct movies from movies table

there are 19 unique genres

All users selected had rated at least 20 movies

Ratings are made on a 5-star scale, with half-star increments.

Chapter 2: Database Design

a) E/R Diagram

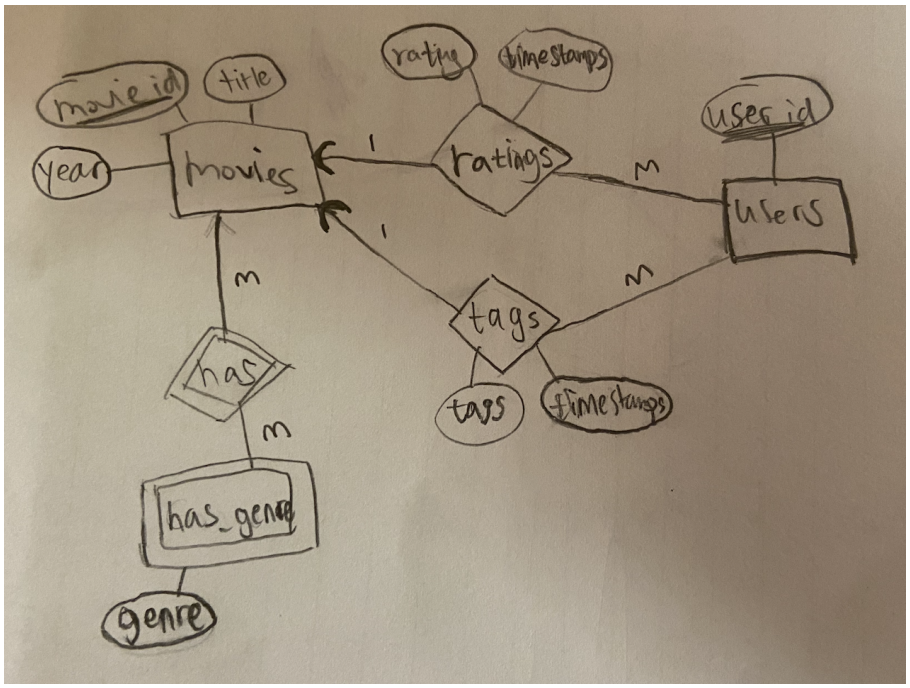
Describe the entities and relationships you discovered during data exploration. Give details about their attributes, how they are related (relationship types), keys.

Entities: movies, users, has_genre(weak entity)

Relationship: ratings, tags

some relationships:

- every movie can be rated exact once by 1 user, but every user can rate many movies
- every movie can be tagged exact once by 1 user, but every user can tag many movies
- has_genre is a weak entity set and movies is strong entity set, so has_genre has to borrow movies's key 'movieid' as its own key
- Every movie can have multiple genres



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)

Genre(movieid, genre)

Ratings(userid, movieid, ratings, timestamps)

Tags(userid, movieid, tags, timestamps)

Users(userid)

Chapter 3: Load Data and test your Database

a) Load Data

-- Database creation

```
CREATE TABLE Ratings(

userid numeric,
movieid numeric,
rating double precision,
timestamp numeric);
CREATE TABLE

CREATE TABLE Tags(
userid numeric,
movieid numeric,
tag text,
timestamp numeric);
CREATE TABLE

CREATE TABLE Movies(
movieid numeric PRIMARY KEY,
title text,
year numeric);
CREATE TABLE

CREATE TABLE Users(
userid numeric PRIMARY KEY);
CREATE TABLE

CREATE TABLE genres(
title text);
CREATE TABLE


CREATE TABLE has_genre(
movieid numeric,
genre text);
CREATE TABLE
```

-- Relations creation

SQL to create tables according to the logical schema of this **E/R DESIGN**

```
ALTER TABLE ratings add constraint uc foreign key(userid) references users(userid);
ALTER TABLE
```

(Note: The above link shows an E/R diagram of a database moviesdb which is a sample solution of phase 1. This diagram is provided to ensure that performance in phase 2 won't be affected by performance in phase 1.)

-- Relations data population

SQL to load the data from the input (.txt) files into the database relations

```
\copy ratings from 'ratings.txt' (DELIMITER(":"));
COPY 10000054


\copy tags from 'tags.csv' (DELIMITER(":"));
COPY 95580


\copy movies from ‘movies.csv' (DELIMITER(":"));
COPY 10681

\copy genres from 'genres.txt'
COPY 19

\copy has_genre from 'has_genre.txt' (DELIMITER(":"));
COPY 21564
```

**movies.csv and tag.csv were generated by two python scripts; genres.txt and has_genre.txt were generated by sql statements.*

b) Test your database

Add a snapshot of the SQL queries of part 3 along with the Postgres response.

A. List your tables.

```
moviesdb=# \dt
          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 | postgres
 public | users     | table | postgres
(6 rows)
```

B. Data types of your tables.

```
moviesdb=# \d genres
            Table "public.genres"
 Column | Type | Collation | Nullable | Default
--------+------+-----------+----------+---------
 title  | text |           |          |
```

```
moviesdb=# \d movies
              Table "public.movies"
 Column  |  Type   | Collation | Nullable | Default
---------+---------+-----------+----------+---------
 movieid | numeric |           | not null |
 title   | text    |           |          |
 year    | numeric |           |          |
Indexes:
    "movies_pkey" PRIMARY KEY, btree (movieid)
```

```
moviesdb=# \d ratings
                  Table "public.ratings"
  Column   |       Type       | Collation | Nullable | Default
```

```
-----------+-----------------+-----------+----------+---------
 userid    | numeric         |           |          |
 movieid   | numeric         |           |          |
 rating    | double precision |           |          |
 timestamp | numeric         |           |          |
```

```
moviesdb=# \d tags
                Table "public.tags"
  Column   |  Type   | Collation | Nullable | Default
-----------+---------+-----------+----------+---------
 userid    | numeric |           |          |
 movieid   | numeric |           |          |
 tag       | text    |           |          |
 timestamp | numeric |           |          |
```

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

```
moviesdb=#  \d has_genre
             Table "public.has_genre"
 Column  |  Type   | Collation | Nullable | Default
---------+---------+-----------+----------+---------
 movieid | numeric |           |          |
 genre   | text    |           |          |
```

C. Sizes of your tables.

```
moviesdb=# select count(*) from genres ;
 count
-------
    19
(1 row)
```

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


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


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


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


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

D. Data values.

```
moviesdb=# select * from movies limit 5;
 movieid |            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=# select count(title) from movies;
 count
-------
 10681
(1 row)


moviesdb=#  select * from movies order by year desc limit 5;
 movieid |                  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=# select * from movies order by year limit 5;
 movieid |            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=# select * from movies order by year limit 5;
 movieid |             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=#  select count(year) from movies;
 count
-------
 10681
(1 row)


moviesdb=# select count(year) from movies where year = 0;
 count
-------
     0
(1 row)


moviesdb=# 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=# Select movieid from has_genre where genre is null;
 movieid
---------
(0 rows)
```

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

```
moviesdb=# select genre, count(genre) from has_genre where genre not in (select title from genres)
group by genre;
       genre        | count
--------------------+-------
 (no genres listed) |     1
(1 row)
```

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

```
moviesdb=# select year, count(year) from movies group by year order by year asc;
 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
 1931 |    16
 1932 |    22
 1933 |    23
 1934 |    18
 1935 |    18
 1936 |    32
 1937 |    30
 1938 |    19
 1939 |    37
 1940 |    40
 1941 |    28
 1942 |    38
```

1943 | 40
1944 | 37
1945 | 36
1946 | 38
1947 | 39
1948 | 46
1949 | 37
1950 | 44
1951 | 44
1952 | 40
1953 | 55
1954 | 43
1955 | 57
1956 | 53
1957 | 62
1958 | 62
1959 | 61
1960 | 66
1961 | 57
1962 | 69
1963 | 63
1964 | 72
1965 | 72
1966 | 87
1967 | 68
1968 | 72
1969 | 64
1970 | 71
1971 | 73
1972 | 83
1973 | 81
1974 | 75
1975 | 74
1976 | 75
1977 | 83
1978 | 82
1979 | 87
1980 | 161
1981 | 178
1982 | 170
1983 | 111
1984 | 137
1985 | 158
1986 | 166
1987 | 205
1988 | 214

```
 1989 |   212
 1990 |   200
 1991 |   188
 1992 |   212
 1993 |   258
 1994 |   307
 1995 |   362
 1996 |   384
 1997 |   370
 1998 |   384
 1999 |   357
 2000 |   405
 2001 |   403
 2002 |   441
 2003 |   366
 2004 |   342
 2005 |   332
 2006 |   345
 2007 |   364
 2008 |   251
(94 rows)
```

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

moviesdb=# select decade, count(*) as count from (select floor(year/10) * 10 as decade from movies) as t group by decade order by decade asc;

```
 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=# select genre, count(genre) from has_genre where genre in (select title from genres)
group by genre;
    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
 Thriller    |  1706
 Western     |   275
 Sci-Fi      |   754
 Comedy      |  3703
 Adventure   |  1025
 War         |   511
(19 rows)
```

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

```
moviesdb=# select rating, count(rating) from ratings group by rating order by rating asc;
 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=# select count(distinct(r.movieid)) from ratings r left join tags t on r.movieid = t.movieid
where t.movieid is null;
 count
-------
  3080
(1 row)
```

ii. no ratings, but they have tags

```
moviesdb=# select count(distinct(t.movieid)) from ratings r right join tags t on r.movieid = t.movieid
where r.movieid is null;
 count
-------
     4
(1 row)
```

iii. no tags and no ratings

```
moviesdb=# select count(distinct(t.movieid)) from ratings r full outer join tags t on r.movieid =
t.movieid where t.movieid not in (select movieid from movies);
 count
-------
     0
(1 row)
```

iv. both tags and ratings

```
moviesdb=# select count(distinct(t.movieid)) from ratings r inner join tags t on r.movieid = t.movieid;

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

5. Add your code from steps 2 and 3 to a code file. Name it "<username>-code.sql"

6. Create a test database, movies_test, to test your code. Run your SQL code on the database from bash:

```
createdb moviesdb_test
```

```
psql -d moviesdb_test -a -f <username>-code.sql
```

This should run your SQL code on an empty database and finish with no errors. Report the results of the screen after you run this command. Save these results in a file "<username>-code-results.pdf".

7. Create a README file. Name it "<username>-README.txt". Add your name, date, course number and project title. Add details about:

i) what each file includes

ii) instructions on how to use each of your code files, including your SQL code and any scripts.