Front page

Xavier Davis
Data Science
333 Intro to Database Systems
Spring 2021
Title of the project: Design, Load and Explore a Movies database

Chapter 1: Project Description

a) Goal of the project

The goal of the project is to take a dataset with information regarding movies and translate it into a database, Movies, that allows for data exploration and querying. To start, you must load the data using the descriptions from the dataset. That means studying the information and organizing it in the most logical manner. The next step is to take the dataset on paper and input the information into a console to build a database. The database is dependent on the creation of an E/R Diagram to have a physical illustration of the entity sets and relationships. Then, translate the information into logical schemas for table set up and proper data types (int, float, string, date, time, etc.). Utilizing the information from the logical schemas makes it easier to input the data and create tables in SQL. To test the database design, you run tests with sample queries. Optimizing queries gives insight into how easy it is to explore data and implement Relational Algebra.

b) Data Exploration

count(ratings): 10000054
count(tags) :95580
count(movies): 10681
count(users): 71567

Three text files:

movies.txt (489.1 KB):

- MovieID, Title, and Genre
- Title includes the name of the film and year it was created
- Identical to the title policy used by IMDB
- There could be some manual errors with the title

- Genre is selected from a pipe-separated list of popular genres (18 in total)

ratings.txt (224.21 MB)

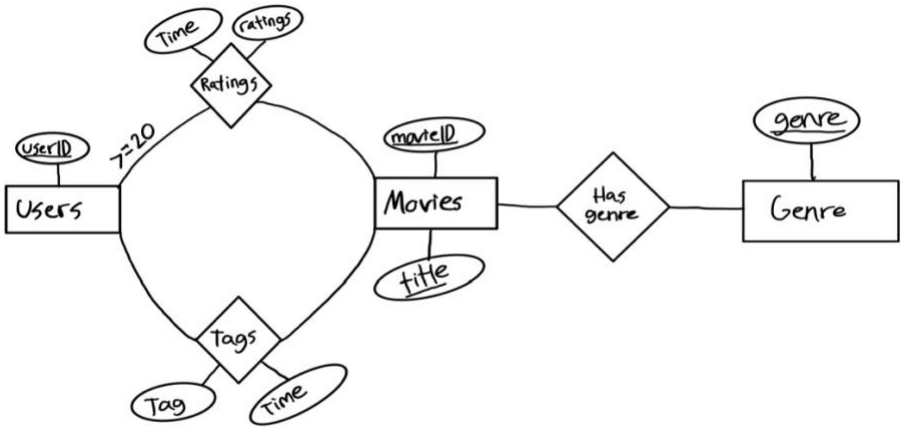
- UserID is the first column representing the ID of each user
- MovieID is the movie that user rated. Each user rated at least 20 movies
- Rating is on a scale of 1-5, similar to the number of stars a movie can receive. Given in multiples of (1/2)
- Timestamp is the time represented in seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970

tags.txt (3.14 MB)

- First column is the UserID
- Second column is the MovieID, the movie the user tagged
- Tags are defined by the user; some are descriptions of the movie and others are verbal ratings. Tags are only one word or short phrase
- Timestamp represents in seconds since midnight Coordinated Universal

Chapter 2: Database Design

a) E/R Diagram



b) Logical Schema

Ratings(userID, movieID, rating, time)

Tags(userID, movieID, tag, time)

Users(userID)

Movies(movieID, title)

Genre(genre)

Has_genre(movieID, title, genre)

Chapter 3: Load Data and test your Database

a) Load Data

Ratings

create table ratings(userid int, movieid int, rating double precision, time bigint, PRIMARY KEY (userid, movieid, rating));

copy ratings from '/Users/xavier.davis/Documents/Database_Systems/movies/ratings.txt' with delimiter ':';

COPY 10000054

Tags

create table tags(userid int, movieid int, tag text, time bigint, PRIMARY KEY (userid, movieid, tag));

copy tags from '/Users/xavier.davis/Documents/Database_Systems/tags.csv' with delimiter '|';

COPY 95580

Has_genre

create table has_genre(movieid int, title varchar(50), primary key(movieid, title));

copy has_genre from '/Users/xavier.davis/Documents/Database_Systems/Has_Genre.csv' with delimiter '~';

COPY 21564

Genre

```
create table genres(genre text primary key);

copy genres from '/Users/xavier.davis/Documents/Database_Systems/movies/genre.csv' with delimiter ' ';

COPY 19
```

Movie

```
create table movie(id int primary key, title text, year int);

copy movie from '/Users/xavier.davis/Documents/Database_Systems/movies.csv' with delimiter '~';

COPY 10681
```

Users

```
create table users(id int);

insert into users(id) select distinct userid from ratings;

INSERT 0 69878

insert into users(id) select distinct userid from tags;

INSERT 0 4009

create table temp(id int);

insert into temp(id) select distinct id from users;

INSERT 0 71567
```

```
drop table users;

create table users(id int primary key);

insert into users(id) select id from temp;

INSERT 0 71567
```

b) Test your database

A. List your tables.

```
\d+
                              List of relations
 Schema |   Name    | Type  |    Owner     | Persistence |  Size   | Description
--------+-----------+-------+--------------+-------------+---------+-------------
 public | genres    | table | xavier.davis | permanent   | 16 kB   |
 public | has_genre | table | xavier.davis | permanent   | 960 kB  |
 public | movie     | table | xavier.davis | permanent   | 672 kB  |
 public | ratings   | table | xavier.davis | permanent   | 498 MB  |
 public | tags      | table | xavier.davis | permanent   | 8112 kB |
 public | users     | table | xavier.davis | permanent   | 2568 kB |
(6 rows)
```

B. Data types of your tables.

```
\d genres
            Table "public.genres"
 Column | Type | Collation | Nullable | Default
--------+------+-----------+----------+---------
 genre  | text |           | not null |
Indexes:
    "genres_pkey" PRIMARY KEY, btree (genre)
```

```
\d movie
              Table "public.movie"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 title  | text    |           |          |
 year   | integer |           |          |
Indexes:
    "movie_pkey" PRIMARY KEY, btree (id)



\d ratings
                 Table "public.ratings"
 Column  |       Type       | Collation | Nullable | Default
---------+------------------+-----------+----------+---------
 userid  | integer          |           | not null |
 movieid | integer          |           | not null |
 rating  | double precision |           | not null |
 time    | bigint           |           |          |
Indexes:

    "ratings_pkey" PRIMARY KEY, btree (userid, movieid, rating)



\d tags
               Table "public.tags"
 Column  |  Type   | Collation | Nullable | Default
---------+---------+-----------+----------+---------
```

```
 userid  | integer |           | not null |
 movieid | integer |           | not null |
 tag     | text    |           | not null |
 time    | bigint  |           |          |
Indexes:
    "tags_pkey" PRIMARY KEY, btree (userid, movieid, tag)



\d users
              Table "public.users"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
Indexes:

    "users_pkey" PRIMARY KEY, btree (id)



\d has_genre
                  Table "public.has_genre"
 Column  |         Type          | Collation | Nullable | Default
---------+-----------------------+-----------+----------+---------
 movieid | integer               |           | not null |
 title   | character varying(50) |           | not null |
Indexes:
    "has_genre_pkey" PRIMARY KEY, btree (movieid, title)
```

C. Sizes of your tables.

```
select * from genres;
```

```
   genre
-------------
 IMAX
 Action
 Adventure
 Animation
 Children's
 Comedy
 Crime
 Documentary
 Drama
 Fantasy
 Film-Noir
 Horror
 Musical
 Mystery
 Romance
 Sci-Fi
 Thriller
 War
 Western
(19 rows)

select count(*) from movie;
 count
-------
 10681
(1 row)

select count(*) from ratings;
```

```
  count
----------
 10000054
(1 row)

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

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

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

D. Data values

```
select * from movie 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)

select count(title) from movie;
 count
-------
 10681
(1 row)

select * from movie 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)

select * from movie where year = 0;
 id | title | year
----+-------+------
(0 rows)

select count(year) from movie;
 count
-------
 10681
(1 row)
```

```
select count(year) from movie where year = 0;
 count
-------
     0
(1 row)

select count(year) from movie where year > 1500;
 count
-------
 10681
(1 row)
```

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

```
select year, count(year) from movie 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)
```

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

```
select title, count(title) from has_genre group by title;
      title        | 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.).

```
select rating, count(rating) from ratings group 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

```
select count(distinct movieid) from ratings where movieid not in (select distinct movieid from
tags);
 count
-------
  3080
(1 row)
```

ii. no ratings, but they have tags

```
select count(distinct movieid) from tags where movieid not in (select distinct movieid from
ratings);
 count
-------
     4
(1 row)
```

iii. no tags and no ratings

```
select id from movie where id not in (select movieid from ratings union select movieid from
tags);
 id
----
```

(0 rows)

iv. both tags and ratings

```
select count(id) from movie where id in (select movieid from ratings intersect  select movieid from tags);
 count
-------
  7597
(1 row)
```

Chapter 4: Query the database and Optimize the Queries

1. **Find the most reviewed movie** (that is, the movie with the highest number of reviews). Show the movie id, movie title and the number of reviews.

```
select movies.movieid, movies.title, count(rating) from ratings, movies where movies.movieid = ratings.movieid group by movies.title, movies.movieid order by count(rating) desc limit 1;
 movieid |    title     | count
---------+--------------+-------
     296 | Pulp Fiction | 34864
(1 row)
```

2. **Find the highest reviewed movie** (movie with the most 5-star reviews). Show the movie id, movie title and the number of reviews.

select movies.movieid, movies.title, count(movies.movieid) from movies where movieid in (select movieid from ratings where rating = 5 group by movieid order by count(*) desc limit 1) group by movies.movieid, movies.title;

```
 id  |          title          | count
-----+-------------------------+-------
 318 | Shawshank Redemption, The |     1
(1 row)
```

3. **Find the number of movies that are associated with at least 4 different genres**.

```
select count(*) from (select count(*) from has_genre group by movieid having count(movieid) >=
4) dt;
 count
-------
   968
(1 row)
```

4. **Find the most popular genre across all movies** (genre associated with the highest number of movies).

select genre, count(movieid) from has_genre group by genre order by count(movieid) desc limit 1;

```
 title | count
-------+-------
```

Drama | 5339

(1 row)

5.

a. **Find the genres that are associated with the best reviews** (genres of movies that have more high ratings than low ratings). Display the genre, the number of high ratings (>=4.0) and the number of low ratings (< 4.0).

```
select list.genre, high, low from( select list.genre, count(rate.rating) as high from has_genre list, ratings rate where list.movieid = rate.movieid and rate.rating >= 4 group by list.genre) list join (select list.genre, count(rate.rating) as low from has_genre list, ratings rate where list.movieid = rate.movieid and rating < 4 group by list.genre) total on list.genre = total.genre where high > low;
```

```
   genre    | high   |   low
------------+---------+---------
 Animation   |  275590 |  243522
 Crime       |  826375 |  648582
 Documentary |   64986 |   38468
 Drama       | 2455297 | 1888901
 Film-Noir   |   94675 |   36917
 IMAX        |    5501 |    3579
 Musical     |  250613 |  230561
 Mystery     |  356799 |  274145
 Romance     |  977944 |  923939
 War         |  348331 |  219732
 Western     |  108365 |  102094
(11 rows)
```

b. **Find the genres that are associated with the most recent movies** (genres that have more recent movies than old movies). Display the genre, the number of recent movies (>=2000) and the number of old movies (< 2000).

```
select list.genre, high, low from (select has.genre, count(title) as high from has_genre has join movies film on has.movieid = film.movieid where film.year >= 2000 group by 1) list join (select has.genre, count(title) as low from has_genre has join movies film on has.movieid = film.movieid where film.year < 2000 group by 1) total on list.genre = total.genre where high > low;
```

```
   genre    | high | low
------------+------+-----
 Documentary |  252 | 230
(1 row)
```

Chapter 5: Discussion

When creating the E/R diagrams I made the assumptions that ratings, tags, and has_genres were relationships while movies, users, and genres are entity sets. Ratings, tags, and has_genre are relationships because they contain the primary keys from the entity sets. Genres is an entity set because it works out better when working with queries. By having a list of all the genres in one place set as a primary key then I can easily check other tables to see if they match certain genres.
A constraint I encountered was with creating the entity set users. In result of userid being a primary key I had to take extra steps to insure no redundancy. To do so I inserted the distinct values from tags and ratings into users. Next, I created a temporary table to take in distinct values from the newly filled users table. After I emptied users and inserted the values from the temporary table, users had the right entries.