Edward Rees

Computer Science

CS 333 Introduction to Database Systems

Spring 2022

Design, Load, and Explore a Movies database

Chapter 1: Project Description

The Goal of this Project

The goal of this project is to better understand the process of creating and working with a database. This will be done by understanding various components of the database design, such as: loading data from a file, designing and building a database based on information provided, testing the database with sample queries, exploring the database, querying the database created, and optimizing queries for the database. In doing these, a stronger foundation and understanding of database design will be gained.

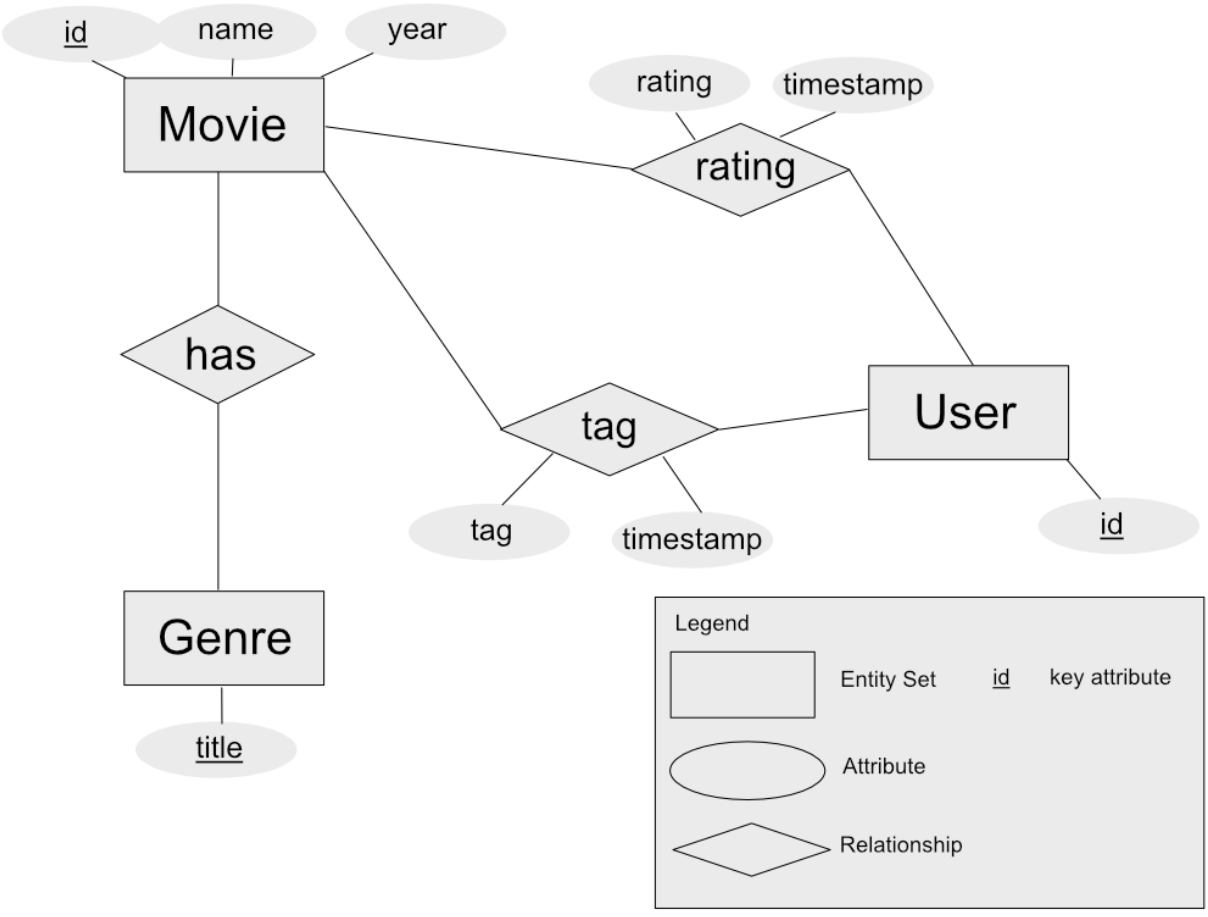
Data Exploration

The dataset contains three txt files. The first is the movies.txt file, which contains a list of movies with each row containing the movie id, the movie name with the year released included, and a list of genres. The movie id is an integer, while the movie name and the genres are all strings. The second is the ratings.txt file, which contains a list of ratings from a given user id, movie id, rating, and the timestamp for the rating. All of the attributes from the ratings.txt are integers, as they are all represented as numbers. The third file is the tags.txt file, which contains the user id, the movie id, the tag, and the timestamp. The user id, movie id, and timestamp are all integers, while the tag itself is a string.

Chapter 2: Database Design

E/R Diagram

When looking at the data, I came up with the E/R diagram below. I saw that Movie would have to be its own entity set, with the attributes of id and name. I initially thought of Rating and Tag as another entity set, until noticing that userId is common in both, which made me think that User can be its own entity set, with an ID attribute. This leads to Rating and Tag both being relationships between Movie and User, with Rating have rating and timestamp as additional attributes, and Tag having tag and timestamp as additional attributes. I then considered Genre as its own entity set connecting with Movie, with genre being a primary key, with Movie and Genre having a many-to-many relationship.



Logical Schema

Movie (id: int, name: text, year: integer)
User (id: int)

Rating (userId: int, movieId: int, rating: text, timestamp: timestamp)
Tag (userId: int, movieId:int, tag: text, timestamp: timestamp)
Genre (gen_title: text)

Chapter 3: Load Data and Database Testing

Section A: Load Data

Repository: https://github.com/EdwardRees/Movie-Database

SQL Code:
db.sql

```
CREATE DATABASE moviesdb;
DROP TABLE IF EXISTS movies CASCADE;
DROP TABLE IF EXISTS users CASCADE;
DROP TABLE IF EXISTS ratings;
DROP TABLE IF EXISTS tags;
DROP TABLE IF EXISTS genres;
DROP TABLE IF EXISTS has_genres;
CREATE TABLE movies(
 id SERIAL PRIMARY KEY,
 title TEXT NOT NULL,
 year INTEGER NOT NULL
);
CREATE TABLE users(id SERIAL PRIMARY KEY);
CREATE TABLE ratings(
 movieId INTEGER NOT NULL,
 userId INTEGER NOT NULL,
 rating FLOAT NOT NULL,
 ratingTime INTEGER,
 FOREIGN KEY (movieId) REFERENCES movies(id),
 FOREIGN KEY (userId) REFERENCES users(id)
);
CREATE TABLE tags(
 movieId INTEGER NOT NULL,
 userId INTEGER NOT NULL,
 tag TEXT NOT NULL,
 tagTime INTEGER,
 FOREIGN KEY(movieId) REFERENCES movies(id),
 FOREIGN KEY(userId) REFERENCES users(id)
);
CREATE TABLE genres(
 genreTitle TEXT NOT NULL
);
CREATE TABLE has_genre (
 movieId INTEGER NOT NULL,
```

```
 genreTitle TEXT NOT NULL
);

\copy movies(id, title, year) FROM './out/movies.txt' DELIMITER ';';
\copy genres(genreTitle) FROM './out/genres.txt' DELIMITER ';';
\copy has_genre(movieId, genreTitle) FROM './out/has_genre.txt' DELIMITER ';';
\copy users(id) FROM './out/users.txt' DELIMITER ',';
\copy ratings(userId, movieId, rating, ratingTime) FROM './out/ratings.txt' DELIMITER
',';
\copy tags(userId, movieId, tag, tagTime) FROM './out/tags.txt' DELIMITER ',';
```

File Editing Code / Supporting Code:

Constants.py

```
DELIM1 = ';'
DELIM2 = ','
```

Download.py

```
from shutil import move
from os import mkdir, path
import wget
import zipfile

def prep():
 if path.exists("./movies"):
   print("movies directory already exists")
   print("Skipping setup process")
   return
 url = "https://www.dropbox.com/s/2rn7qc51yvmb766/movies.zip?dl=1"
 movieszip = wget.download(url, 'movies.zip')
 print(movieszip)

 path.isdir('./out') or mkdir('./out')
 path.isdir("./movies") or mkdir("./movies")

 move(movieszip, "./movies/movies.zip")

 print("Extracting movies.zip...")
 with zipfile.ZipFile("./movies/movies.zip", 'r') as zip_ref:
   zip_ref.extractall("./movies")
```

```
 print("Done!")

if __name__ == '__main__':
 prep()
```

Files.py

```
from sys import argv
from movies import parseMovies, validateMovies, outputMovies
from genres import parseGenres, outputGenres
from tags import parseTags, validateTags, outputTags
from ratings import parseRatings, outputRatings
from download import prep
from users import Users


def main():
   prep()
   if(len(argv) > 1):
     if(argv[1] == '-h'):
       print("Usage: python3 files.py [-h,-d]\n-h: help\n-d: debug. Will print out the values of the files parsed as the original list and dictionary values.\nIf no flag is passed, the program will continue with the default behavior of outputting the files to the output directory.")
       return
     elif(argv[1] == '-d'):
         movies = parseMovies()
         invalidMovies = validateMovies(movies)
         if(len(invalidMovies) > 0):
             print("Invalid movies found. Update parser to handle the following cases:")
             print("Invalid movies:")
             for movie in invalidMovies:
                 print(f"{movie['id']}:{movie['name']}")
         else:
             print(movies)
         genres = parseGenres()
         outputGenres(genres)
         tags = parseTags()
         invalidTags = validateTags(tags)
         if(len(invalidTags) > 0):
             print("Invalid tags found. Update parser to handle the following cases:")
             print("Invalid tags:")
```

```
            for tag in invalidTags:
                print(tag)
        else:
            print(tags)
        ratings = parseRatings()
        print(ratings)
    else:
        movies = parseMovies()
        invalidMovies = validateMovies(movies)
        if(len(invalidMovies) > 0):
            print("Invalid movies found. Update parser to handle the following cases:")
            print("Invalid movies:")
            for movie in invalidMovies:
                print(f"{movie['id']}:{movie['name']}")
        else:
            outputMovies(movies)
        genres = parseGenres()
        outputGenres(genres)
        tags = parseTags()
        invalidTags = validateTags(tags)
        if(len(invalidTags) > 0):
            print("Invalid tags found. Update parser to handle the following cases:")
            print("Invalid tags:")
            for tag in invalidTags:
                print(tag)
        else:
            outputTags(tags)
        ratings = parseRatings()
        outputRatings(ratings)
        Users.outputUsers()
    print("Done!")

if __name__ == '__main__':
    main()
```

Genres.py

```
from util import readFile
from constants import DELIM1, DELIM2

def parseGenres():
 print("Parsing genres...")
```

```
 genreList = []
 moviesFile = readFile('./movies/movies.txt')
 count = 1
 # Format: movieId:movieName (year):genre1|genre2|genre3
 for line in moviesFile.split("\n"):
   if line == '':
     continue
   line = line.split(":")
   movieId = line[0]
   genres = line[-1].split("|")
   for genre in genres:
     genreList.append({
       'movieId': movieId,
       'genre': genre
     })
 return genreList

def getGenreList(genres):
 genreList = []
 for genre in genres:
   if genre['genre'] not in genreList:
     genreList.append(genre['genre'])
 return genreList


def outputGenres(genres):
 print("Writing genres...")
 with open('./out/genres.txt', 'w') as f:
   genreList = getGenreList(genres)
   for genre in genreList:
     f.write(f"{genre}\n")
 with open('./out/has_genre.txt', 'w') as f:
   for genre in genres:
     f.write(f"{genre['movieId']}{DELIM1}{genre['genre']}\n")
```

Movies.py

```
from util import readFile
```

```python
from constants import DELIM1, DELIM2

def parseMovies():
 print("Parsing movies...")
 movies = []
 genres = []
 moviesFile = readFile('./movies/movies.txt')
 for line in moviesFile.split('\n'):
   if line == '':
     continue
   line = line.split(":")
   movieId = line[0]
   genres = line[-1].split("|")
   movieName = ':'.join(line[1:-1])
   movieYear = movieName.split("(")[-1].split(")")[0]
   movieName = movieName.replace(f"({movieYear})", "")
   movieName = movieName.strip()
   movies.append({
     'id': movieId,
     'name': movieName,
     'year': movieYear,
   })
 return movies

def validateMovies(movies):
 print("Validating movies...")
 invalidMovies = []
 for movie in movies:
   if movie['year'] == '':
     print(f"{movie['id']}:{movie['name']}")
     invalidMovies.append(movie)
   if f"({movie['year']})" in movie['name']:
     print(f"{movie['id']}:{movie['name']}")
     invalidMovies.append(movie)
   try:
     int(movie['year'])
   except ValueError:
     invalidMovies.append(movie)
     print(movie['name'])
 return invalidMovies
```

```
def outputMovies(movies):
 print("Writing movies...")
 with open('./out/movies.txt', 'w') as f:
   for movie in movies:
     f.write(f"{movie['id']}{DELIM1}{movie['name']}{DELIM1}{movie['year']}\n")
```

Ratings.py

```
from util import readFile
from constants import DELIM1, DELIM2
from users import Users

def parseRatings():
 print("Parsing ratings...")
 ratings = []
 ratingsFile = readFile('./movies/ratings.txt')
 for line in ratingsFile.split('\n'):
   if line == '':
     continue
   line = line.split(":")
   Users.addToUsers(line[0])
   userId = line[0]
   movieId = line[1]
   rating = line[2]
   timestamp = line[3]
   ratings.append({
     'userId': userId,
     'movieId': movieId,
     'rating': rating,
     'timestamp': timestamp,
   })
 return ratings

def outputRatings(ratings):
 print("Writing ratings...")
 with open('./out/ratings.txt', 'w') as f:
   for rating in ratings:

f.write(f"{rating['userId']}{DELIM2}{rating['movieId']}{DELIM2}{rating['rating']}{DELI
M2}{rating['timestamp']}\n")
```

Tags.py

```
from util import readFile
from constants import DELIM1, DELIM2
from users import Users

def parseTags():
 print("Parsing tags...")
 tags = []
 tagsFile = readFile("./movies/tags.txt")
 for line in tagsFile.split("\n"):
   if line == '':
     continue
   line = line.split(":")
   userId = line[0]
   Users.addToUsers(userId)
   movieId = line[1]
   timestamp = line[-1]
   tag = ':'.join(line[2:-1])
   tags.append({
     'userId': userId,
     'movieId': movieId,
     'timestamp': timestamp,
     'tag': tag
   })
 return tags

def validateTags(tags):
 print("Validating tags...")
 invalidTags = []
 for line in tags:
   if len(line) != 4:
     print("Invalid tag:", line)
     invalidTags.append(line)
 return invalidTags

def outputTags(tags):
 print("Writing tags...")
 with open("./out/tags.txt", "w") as f:
   for line in tags:

f.write(f"{line['userId']}{DELIM2}{line['movieId']}{DELIM2}{line['tag']}{DELIM2}{line[
'timestamp']}\n")
```

Util.py

```
def readFile(file):
 with open(file, 'r') as f:
   return f.read()
```

Users.py

```
from util import readFile
from constants import DELIM1, DELIM2



class Users:
   _users = list()

   @classmethod
   def addToUsers(cls, userId):
       cls._users.append(userId)

   @classmethod
   def __sortUsers(cls):
       print("Sorting users...")
       cls._users = list(set(cls._users))
       cls._users = [int(userId) for userId in cls._users]
       cls._users.sort()

   @classmethod
   def outputUsers(cls):
       cls.__sortUsers()
       print("Writing users...")
       with open("./out/users.txt", "w") as f:
           for user in cls._users:
               f.write(f"{user}\n")
```

Run.sh

```
#!/bin/bash

# This script is used to run the application.

# Check for dependencies
```

```
echo "Checking dependencies..."

# Check if wget is installed
which wget >/dev/null
if [ $? -ne 0 ]; then
 echo "wget is not installed. Please install it and try again."
 exit 1
fi

# Check if python3 is installed
which python3 >/dev/null
if [ $? -ne 0 ]; then
 echo "python3 is not installed. Please install it and try again."
 exit 1
fi

# Check if pip3 is installed
which pip3 >/dev/null
if [ $? -ne 0 ]; then
 echo "pip3 is not installed. Please install it and try again."
 exit 1
fi

# Check if wget is installed in pip3
pip3 show wget >/dev/null
if [ $? -ne 0 ]; then
 pip3 install wget
 echo "wget is not installed in pip3. Please install it and try again."
 exit 1
fi

# Check if postgres is installed
which psql >/dev/null
if [ $? -ne 0 ]; then
 echo "postgres is not installed. Please install it and try again."
 exit 1
fi

echo "No dependencies missing, continuing..."
echo ""
echo "Running files program..."
echo ""
```

```
python3 files.py

echo "Finished running files program..."
echo ""

echo "Checking for database ..."
echo ""

psql -U postgres -c "SELECT 1 FROM pg_database WHERE datname = 'moviesdb'" | grep -q 1
if [ $? -ne 0 ]; then
 echo "Database does not exist. Creating database..."
 psql -U postgres -c "CREATE DATABASE moviesdb;"
 echo "Database created."
 echo ""
else
 echo "Database exists."
 echo ""
fi

echo "Running database script..."
echo ""
psql moviesdb < db.sql

echo "Finished running database script..."
echo ""

echo "Done..."
```

Section B: Database Testing

A. List your tables:

```
moviesdb=# \d
              List of relations
 Schema |     Name      |   Type   |   Owner
--------+---------------+----------+------------
 public | genres        | table    | edwardrees
 public | has_genre     | table    | edwardrees
 public | movies        | table    | edwardrees
 public | movies_id_seq | sequence | edwardrees
```

```
 public | ratings       | table    | edwardrees
 public | tags          | table    | edwardrees
 public | users         | table    | edwardrees
 public | users_id_seq  | sequence | edwardrees
(8 rows)
```

B. Data types of your tables

```
moviesdb=# \d genres
              Table "public.genres"
   Column   | Type | Collation | Nullable | Default
------------+------+-----------+----------+---------
 genretitle | text |           | not null |
```

```
moviesdb=# \d movies
                        Table "public.movies"
 Column |  Type   | Collation | Nullable |              Default
--------+---------+-----------+----------+-----------------------------------
 id     | integer |           | not null | nextval('movies_id_seq'::regclass)
 title  | text    |           | not null |
 year   | integer |           | not null |
Indexes:
    "movies_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "genres" CONSTRAINT "genres_movieid_fkey" FOREIGN KEY (movieid)
REFERENCES movies(id)
    TABLE "ratings" CONSTRAINT "ratings_movieid_fkey" FOREIGN KEY (movieid)
REFERENCES movies(id)
    TABLE "tags" CONSTRAINT "tags_movieid_fkey" FOREIGN KEY (movieid) REFERENCES
movies(id)
```

```
moviesdb=# \d ratings
                   Table "public.ratings"
   Column   |       Type       | Collation | Nullable | Default
------------+------------------+-----------+----------+---------
 movieid    | integer          |           | not null |
 userid     | integer          |           | not null |
 rating     | double precision |           | not null |
 ratingtime | integer          |           |          |
Foreign-key constraints:
    "ratings_movieid_fkey" FOREIGN KEY (movieid) REFERENCES movies(id)
```

```
    "ratings_userid_fkey" FOREIGN KEY (userid) REFERENCES users(id)

moviesdb=# \d tags
                Table "public.tags"
 Column  |  Type   | Collation | Nullable | Default
---------+---------+-----------+----------+---------
 movieid | integer |           | not null |
 userid  | integer |           | not null |
 tag     | text    |           | not null |
 tagtime | integer |           |          |
Foreign-key constraints:
    "tags_movieid_fkey" FOREIGN KEY (movieid) REFERENCES movies(id)
    "tags_userid_fkey" FOREIGN KEY (userid) REFERENCES users(id)

moviesdb=# \d has_genre
              Table "public.has_genre"
   Column   |  Type   | Collation | Nullable | Default
------------+---------+-----------+----------+---------
 movieid    | integer |           | not null |
 genretitle | text    |           | not null |


C. Size of your tables
moviesdb=# select count(*) from genres;
 count
-------
    20
(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;
 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=# select count(title) from movies;
 count
-------
 10681
(1 row)

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

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


moviesdb=# select count(movieid) from has_genre where genretitle='(no genre listed)';
 count
-------
     0
(1 row)
```

E. Extra queries

```
1. moviesdb=# select * from movies where not (movies is not null);
 id | title | year
----+-------+------
(0 rows)

moviesdb=# select * from ratings where not (ratings is not null);
 movieid | userid | rating | ratingtime
---------+--------+--------+------------
(0 rows)

moviesdb=# select * from tags where not (tags is not null);
 movieid | userid | tag | tagtime
---------+--------+-----+---------
(0 rows)

moviesdb=# select * from users where not (users is not null);
```

```
 id
----
(0 rows)

moviesdb=# select * from genres where not (genres is not null);
 genretitle
------------
(0 rows)
```

2. Select year, count(title) 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. moviesdb=# WITH series AS (SELECT generate_series(1910, 2000, 10) AS r_from), range AS (SELECT r_from, (r_from + 9) AS r_to FROM series) SELECT r_from as decade, (SELECT count(title) FROM movies WHERE year BETWEEN r_from AND r_to) as count FROM range;

```
 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. moviesdb=# select genretitle as genre, count(movieid) from has_genre 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
 (no genres listed) |     1
 Thriller           |  1706
 Western            |   275
 Sci-Fi             |   754
 Comedy             |  3703
 Adventure          |  1025
 War                |   511
(20 rows)
```

5. moviesdb=# select rating, count(movieid) 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 movies that have:

i. No tags, but have ratings

```
SELECT count(*) FROM movies WHERE id IN (SELECT DISTINCT movieid FROM ratings)
AND id NOT IN (SELECT DISTINCT movieid FROM tags);
 count
-------
  3080
(1 row)
```

ii. no ratings, but have tags

```
moviesdb=# select count(*) from movies where id in (select distinct movieid from tags) and id
not in (select distinct movieid from ratings);
 count
-------
     4
(1 row)
```

iii. No tags and no ratings

```
SELECT count(*) FROM movies WHERE id NOT IN (SELECT DISTINCT movieid FROM tags)
AND id NOT IN (SELECT DISTINCT movieid FROM ratings);
 count
-------
     0
(1 row)
```

iv. both tags and ratings

```
SELECT count(*) FROM movies WHERE id IN (SELECT DISTINCT movieid FROM tags) AND
id IN (SELECT DISTINCT movieid FROM ratings); count
-------
  7597
(1 row)
```