Paige Rosynek

CS 3860

Dr. Magana

10.6.2022

Lab 3 – Data Modelling & SQL

Part 1 – Create a Video Database

1. Review videodb-readme.txt
   After reviewing the videodb-readme.txt that contains the fields for the other data files: Video_Categories.txt, Video_Actors.txt, and Video_Recordings.txt. While looking at this file, I observed that there are a lot of fields in the Video_Recordings.txt file which could be something we have to fix later in the lab. In addition, I noticed that the Video_Actors file has a recording_id which indicates that there is a relationship between video actors and video recordings.

2. Create a Video database
   Proof and verification of the created database is shown below.

```
mysql> CREATE DATABASE Video
    -> ;
Query OK, 1 row affected (0.01 sec)

mysql> use Video
Database changed
```

3. Create an SQL script to create import tables for importing the data as tab-delimited text files. Note: There are differences between importing csv- and tab-delimited files. Why would you select one format over another?

   One reason you may want to have a .csv file (which can be delimited with a ‘,’) is if you don’t know what type of system a tab-delimited file was made on, then you may have issues loading in the file because different systems have different encodings for how large a tab is. The screenshots below show the results of creating each of the tables: video_actors, video_categories, and video_recordings.

```
mysql> CREATE TABLE video_recordings (
    -> recording_id int,
    -> director varchar(100),
    -> title varchar(200),
    -> category varchar(100),
    -> image_name varchar(100),
    -> duration float,
    -> rating varchar(10),
    -> year_released int,
    -> price float,
    -> stock_count float
    -> );
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> CREATE TABLE video_categories (
    -> id int,
    -> name varchar(100)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> CREATE TABLE video_actors (
    -> id int,
    -> name varchar(100),
    -> recording_id int)
    -> CHARACTER SET latin1;
Query OK, 0 rows affected (0.03 sec)
```

4. Load the tables from the data files, verify the data has been successfully imported.
   The screenshots below show the 'Load Data infile' successful load of each of the data files. In addition, I included the 'update' commands that was used to clean up punctuation in the data files.

```
mysql> Load Data local Infile 'C:/Users/rosynekp/OneDrive - Milwaukee School of Engineering/Desktop/cs 3860/labs/lab03/v
ideodb2022/Video_Categories.txt'
    -> into table video.video_categories fields terminated by '\t'
    -> lines terminated by '\r\n';
Query OK, 6 rows affected (0.01 sec)
Records: 6  Deleted: 0  Skipped: 0  Warnings: 0
```

```
mysql> Load Data local Infile 'C:/Users/rosynekp/OneDrive - Milwaukee School of Engineering/Desktop/cs 3860/labs/lab03/v
ideodb2022/Video_Recordings.txt'
    -> into table video.video_recordings fields terminated by '\t'
    -> lines terminated by '\r\n';
Query OK, 55 rows affected (0.02 sec)
Records: 55  Deleted: 0  Skipped: 0  Warnings: 0
```

```
mysql> Load Data local Infile 'C:/Users/rosynekp/OneDrive - Milwaukee School of Engineering/Desktop/cs 3860/labs/lab03/
ideodb2022/Video_Actors.txt'
    -> into table video.video_actors
    -> CHARACTER SET latin1
    -> fields terminated by '\t'
    -> lines terminated by '\r\n';
Query OK, 372 rows affected (0.02 sec)
Records: 372  Deleted: 0  Skipped: 0  Warnings: 0
```

```
mysql> update video_categories set name = replace(trim(name),'"','');
Query OK, 6 rows affected (0.01 sec)
Rows matched: 6  Changed: 6  Warnings: 0

mysql> update video_actors set name = replace(trim(name),'"','');
Query OK, 372 rows affected (0.02 sec)
Rows matched: 372  Changed: 372  Warnings: 0

mysql> update video_recordings set director = replace(trim(director),'"','');
Query OK, 55 rows affected (0.01 sec)
Rows matched: 55  Changed: 55  Warnings: 0

mysql> update video_recordings set title = replace(trim(title),'"','');
Query OK, 55 rows affected (0.01 sec)
Rows matched: 55  Changed: 55  Warnings: 0

mysql> update video_recordings set category = replace(trim(category),'"','');
Query OK, 55 rows affected (0.01 sec)
Rows matched: 55  Changed: 55  Warnings: 0

mysql> update video_recordings set image_name = replace(trim(image_name),'"','');
Query OK, 55 rows affected (0.00 sec)
Rows matched: 55  Changed: 55  Warnings: 0

mysql> update video_recordings set rating = replace(trim(rating),'"','');
Query OK, 55 rows affected (0.01 sec)
Rows matched: 55  Changed: 55  Warnings: 0
```

Trimming the columns of the tables and replacing "".

The screenshots below display the data loaded into each of the tables, verifying the data was loaded in correctly.

```
mysql> SELECT * FROM video.video_recordings;
+--------------+--------------------+-----------------------------+-----------------+-------------------------
--+----------+--------+---------------+-------+-------------+
| recording_id | director           | title                       | category        | image_name
  | duration | rating | year_released | price | stock_count |
+--------------+--------------------+-----------------------------+-----------------+-------------------------
--+----------+--------+---------------+-------+-------------+
|         3000 | Francis Ford Coppola | Apocalypse Now              | Action & Adventur | apocalypse_now.gif
  |     9180 | R      |          1979 | 22.99 |           0 |
|         3001 | Michael Bay          | Bad Boys                    | Action & Adventur | badboys.gif
  |     7140 | R      |          1995 | 15.99 |         782 |
|         3002 | Mel Gibson           | Braveheart                  | Action & Adventur | braveheart.gif
  |    10620 | R      |          1995 | 14.99 |         582 |
|         3003 | Mimi Leder           | Deep Impact                 | Action & Adventur | deep_impact.gif
  |     5700 | PG-1   |          1998 | 11.99 |         501 |
|         3004 | J. Robert Wagoner    | Disco Godfather             | Action & Adventur | disco_godfather.gif
  |     5580 | R      |          1993 | 10.99 |         872 |
|         3005 | Stanley Kubrick      | Full Metal Jacket           | Action & Adventur | full_metal_jacket.gif
  |     7020 | R      |          1987 | 16.99 |         872 |
|         3006 | Roland Emmerich      | Independence Day            | Action & Adventur | independence_day.gif
  |     8700 | PG-1   |          1996 | 16.99 |           0 |
|         3007 | John McTiernan       | The Hunt for Red October    | Action & Adventur | hunt_for_red_october.gif
```

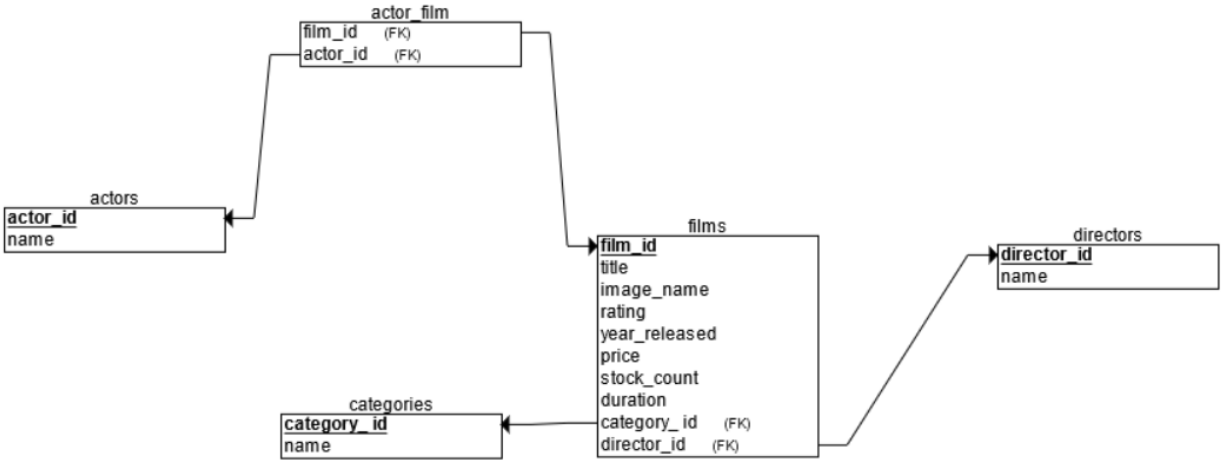
```
mysql> SELECT * FROM video.video_categories;
+------+--------------------+
| id   | name               |
+------+--------------------+
|    0 | Action & Adventure |
|    1 | Comedy             |
|    2 | Drama              |
|    3 | Horror             |
|    4 | Science Fiction    |
|    5 | Suspense           |
+------+--------------------+
6 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM video.video_actors;
+------+----------------------+--------------+
| id   | name                 | recording_id |
+------+----------------------+--------------+
|    0 | Marlon Brando        |         3000 |
|    1 | Martin Sheen         |         3000 |
|    2 | Robert Duvall        |         3000 |
|    3 | Frederic Forrest     |         3000 |
|    4 | Dennis Hopper        |         3000 |
|    5 | Scott Glenn          |         3000 |
|    6 | Harrison Ford        |         3000 |
|    7 | Laurence Fishburne   |         3000 |
|    8 | Sam Bottoms          |         3000 |
|    9 | Will Smith           |         3001 |
|   10 | Martin Lawrence      |         3001 |
```

Part 2 – Design Your Database Schema

5. Relational Data Model for Database Schema

Below is the relational data model for my database schema. One thing I changed from the original data is I replaced the term ‘video_recording’ with ‘film’ for better readability. In addition, I created an actor_film table which relates film_ids and actor_ids which are both foreign keys from their corresponding tables, films and actors. The actor table has an actor_id as the primary key and then a name field. The films table has the film_id as the primary key, the category_id and director_id as the foreign keys, and the following fields: title, image_name, rating, year_released, price, stock_count, and duration. The films table is related to the categories table which has the category_id as the primary key and a name field. The films and directors tables are related through the directors table. The directors table has the director_id as the primary key and a name field.



Relational data model for the video database.

6. Generate SQL Script to Define/Create Tables for Schema

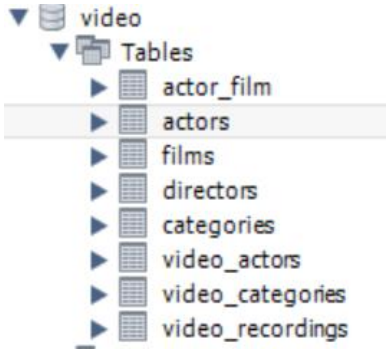
The text boxes below is the SQL script created by ERDplus from the relational model above.

```
CREATE TABLE categories
(
  category__id INT NOT NULL,
  name VARCHAR(100) NOT NULL,
  PRIMARY KEY (category__id)
);

CREATE TABLE actors
(
  actor_id INT NOT NULL,
  name VARCHAR(100) NOT NULL,
  PRIMARY KEY (actor_id)
);

CREATE TABLE directors
(
  director_id INT NOT NULL,
  name VARCHAR(100) NOT NULL,
  PRIMARY KEY (director_id)
);
);
```

```
CREATE TABLE films
(
  film_id INT NOT NULL,
  title VARCHAR(200) NOT NULL,
  image_name VARCHAR(100) NOT NULL,
  rating VARCHAR(10) NOT NULL,
  year_released INT NOT NULL,
  price FLOAT NOT NULL,
  stock_count FLOAT NOT NULL,
  duration FLOAT NOT NULL,
  category__id INT NOT NULL,
  director_id INT NOT NULL,
  PRIMARY KEY (film_id),
  FOREIGN KEY (category__id) REFERENCES
categories(category__id),
  FOREIGN KEY (director_id) REFERENCES
directors(director_id)
);

CREATE TABLE actor_film
(
  film_id INT NOT NULL,
  actor_id INT NOT NULL,
  FOREIGN KEY (film_id) REFERENCES
films(film_id),
  FOREIGN KEY (actor_id) REFERENCES
actors(actor_id)
);
```

7. Load Data from Imported Tables to Final Schema

The screenshot below shows the successful creation of the tables for the new schema using the generated script above. The created tables are: actor_film, actors, films, directors, and categories.



8. Verify Primary & Foreign Key Constraints. Why would I create the primary key index after the table has been created and the data imported versus defining the primary key in the table definition?

Defining the primary key after the table has been created makes inserting data into the tables easier because you won't have to worry about getting any warnings for key constraints when loading data into the table. It can allow you to load in certain columns of data at a time as well as allow you to load tables in any order because there won't be any key constraint errors. For my schema, I defined the key constraints of each table in the 'CREATE TABLE' query, however with this structure I ran into the problems described above which I solved by temporarily turning off foreign key checks which is not best practice. The screenshots below describe the primary and foreign keys of each table. The actors table has the primary key actor_id and the film_actors table has two foreign keys: film_id and actor_id. The films table has the primary key, film_id, and two foreign keys: category_id and director_id. The last two tables, directors and categories, each have a single primary key, named director_id and category_id respectively.

```
mysql> desc actor_film;
+----------+------+------+-----+---------+-------+
| Field    | Type | Null | Key | Default | Extra |
+----------+------+------+-----+---------+-------+
| film_id  | int  | NO   | MUL | NULL    |       |
| actor_id | int  | NO   | MUL | NULL    |       |
+----------+------+------+-----+---------+-------+
```

```
mysql> desc actors;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| actor_id | int          | NO   | PRI | NULL    |       |
| name     | varchar(100) | NO   |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
```

```
mysql> desc directors;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| director_id | int          | NO   | PRI | NULL    | auto_increment |
| name        | varchar(100) | NO   |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
```

```
mysql> desc categories;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| category_id | int          | NO   | PRI | NULL    |       |
| name        | varchar(100) | NO   |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
```

```
mysql> desc films;
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| film_id       | int          | NO   | PRI | NULL    |       |
| title         | varchar(200) | NO   |     | NULL    |       |
| image_name    | varchar(100) | NO   |     | NULL    |       |
| rating        | varchar(10)  | NO   |     | NULL    |       |
| year_released | int          | NO   |     | NULL    |       |
| price         | float        | NO   |     | NULL    |       |
| stock_count   | float        | NO   |     | NULL    |       |
| duration      | float        | NO   |     | NULL    |       |
| category_id   | int          | NO   | MUL | NULL    |       |
| director_id   | int          | NO   | MUL | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
```

9. Verify Tables Loaded

In the screenshots below, I included the top results from a 'select *' query for each of the tables, as well as the results of a count of how many rows are in each table.

```
mysql> select * from actor_film;
+---------+----------+
| film_id | actor_id |
+---------+----------+
|    3000 |        0 |
|    3000 |        1 |
|    3000 |        2 |
|    3000 |        3 |
|    3000 |        4 |
|    3000 |        5 |
|    3000 |        6 |
|    3000 |        7 |
|    3000 |        8 |
|    3001 |        9 |
|    3001 |       10 |
```

```
mysql> select * from actors;
+----------+----------------------+
| actor_id | name                 |
+----------+----------------------+
|        0 | Marlon Brando        |
|        1 | Martin Sheen         |
|        2 | Robert Duvall        |
|        3 | Frederic Forrest     |
|        4 | Dennis Hopper        |
|        5 | Scott Glenn          |
|        6 | Harrison Ford        |
|        7 | Laurence Fishburne   |
|        8 | Sam Bottoms          |
|        9 | Will Smith           |
|       10 | Martin Lawrence      |
```

```
mysql> select * from directors;
+-------------+----------------------+
| director_id | name                 |
+-------------+----------------------+
|           1 | Francis Ford Coppola |
|           2 | Michael Bay          |
|           3 | Mel Gibson           |
|           4 | Mimi Leder           |
|           5 | J. Robert Wagoner    |
|           6 | Stanley Kubrick      |
|           7 | Roland Emmerich      |
|           8 | John McTiernan       |
|           9 | Tony Scott           |
|          10 | Jay Roach            |
```

```
mysql> select count(*) from actor_film;
+----------+
| count(*) |
+----------+
|      372 |
+----------+
```

```
mysql> select count(*) from actors;
+----------+
| count(*) |
+----------+
|      335 |
+----------+
```

```
mysql> select count(*) from directors;
+----------+
| count(*) |
+----------+
|       51 |
+----------+
```

```
mysql> select * from films;
+---------+-------------------------------+-------------------------+--------+---------------+-------+-------------+----------+-------------+-------------+
| film_id | title                         | image_name              | rating | year_released | price | stock_count | duration | category_id | director_id |
+---------+-------------------------------+-------------------------+--------+---------------+-------+-------------+----------+-------------+-------------+
|    3000 | Apocalypse Now                | apocalypse_now.gif      | R      |          1979 | 22.99 |           0 |     9180 |           0 |           1 |
|    3001 | Bad Boys                      | badboys.gif             | R      |          1995 | 15.99 |         782 |     7140 |           0 |           2 |
|    3002 | Braveheart                    | braveheart.gif          | R      |          1995 | 14.99 |         582 |    10620 |           0 |           3 |
|    3003 | Deep Impact                   | deep_impact.gif         | PG-13  |          1998 | 11.99 |         501 |     5700 |           0 |           4 |
|    3004 | Disco Godfather               | disco_godfather.gif     | R      |          1993 | 10.99 |         872 |     5580 |           0 |           5 |
|    3005 | Full Metal Jacket             | full_metal_jacket.gif   | R      |          1987 | 16.99 |         872 |     7020 |           0 |           6 |
|    3006 | Independence Day              | independence_day.gif    | PG-13  |          1996 | 16.99 |           0 |     8700 |           0 |           7 |
|    3007 | The Hunt for Red October      | hunt_for_red_october.gif| PG     |          1990 | 10.99 |         618 |     8100 |           0 |           8 |
|    3008 | The Rock                      | the_rock.gif            | R      |          1996 | 20.99 |         514 |     8160 |           0 |           2 |
|    3009 | Top Gun                       | top_gun.gif             | PG     |          1986 | 11.99 |         499 |     6600 |           0 |           9 |
|    3010 | Austin Powers                 | austin_powers.gif       | PG-13  |          1997 | 17.99 |         688 |     5220 |           1 |          10 |
|    3011 | Caddyshack                    | caddyshack.gif          | R      |          1980 | 18.99 |         938 |     5880 |           1 |          11 |
|    3012 | Car Wash                      | car_wash.gif            | PG     |          1976 | 24.99 |           0 |     5820 |           1 |          12 |
|    3013 | Crooklyn                      | crooklyn.gif            | PG-13  |          1994 |  9.99 |         651 |     6840 |           1 |          13 |
|    3014 | Dolemite                      | dolemite.gif            | R      |          1975 | 19.99 |         489 |     5460 |           1 |          14 |
|    3015 | Dumb and Dumber               | dumb_and_dumber.gif     | PG-13  |          1994 | 23.99 |         973 |     6600 |           1 |          15 |
|    3016 | Monty Python and The Holy Grail | monty_python_holy_grail.gif | PG | 1975 | 17.99 | 980 | 5400 | 1 | 16 |
|    3017 | Mrs. Doubtfire                | mrs_doubtfire.gif       | PG-13  |          1993 | 21.99 |         772 |     7500 |           1 |          17 |
```

```
mysql> select count(*) from films;
+----------+
| count(*) |
+----------+
|       55 |
+----------+
```

```
mysql> select * from categories;
+-------------+--------------------+
| category_id | name               |
+-------------+--------------------+
|           0 | Action & Adventure |
|           1 | Comedy             |
|           2 | Drama              |
|           3 | Horror             |
|           4 | Science Fiction    |
|           5 | Suspense           |
+-------------+--------------------+
6 rows in set (0.00 sec)
```

Part 3 – SQL

Use the original tables imported from the files for the first two questions. Include your observations and explanations along with the question, SQL, and your query results.

1. Execute:

```
SELECT * FROM video_recordings, video_categories;
```

Note the cross-product effect of joining two tables. Record the number of rows generated. Do all permutations of Video Recordings × Video Categories make sense? Explain

The first ten results of the above query are shown in the screenshot below; the total amount of rows returned was 330. The video_recordings table has 55 rows and the video_categories table has 6 rows. The above query takes each row of the video_recordings table and joins it with each row of the video_categories table, so each of the 55 rows will be printed 6 times, thus resulting in 55 * 6 = 330 rows total. From inspection, the permutations of video_recordings x video_categories does not make sense. For each of the rows in video_recordings, each row of video_categories is tacked on the end. This does not make sense because a single movie cannot belong to every category as shown in the first 6 rows of the screenshot below.

```
mysql> SELECT * FROM video_recordings, video_categories;
+--------------+--------------------+-------------------------------+-----------------+-----------------------+----------+--------+---------------+-------+-------------+----+--------------------+
| recording_id | director           | title                         | category        | image_name            | duration | rating | year_released | price | stock_count | id | name               |
+--------------+--------------------+-------------------------------+-----------------+-----------------------+----------+--------+---------------+-------+-------------+----+--------------------+
|         3000 | Francis Ford Coppola | Apocalypse Now              | Action & Adventure | apocalypse_now.gif |     9180 | R      |          1979 | 22.99 |           0 |  5 | Suspense           |
|         3000 | Francis Ford Coppola | Apocalypse Now              | Action & Adventure | apocalypse_now.gif |     9180 | R      |          1979 | 22.99 |           0 |  4 | Science Fiction    |
|         3000 | Francis Ford Coppola | Apocalypse Now              | Action & Adventure | apocalypse_now.gif |     9180 | R      |          1979 | 22.99 |           0 |  3 | Horror             |
|         3000 | Francis Ford Coppola | Apocalypse Now              | Action & Adventure | apocalypse_now.gif |     9180 | R      |          1979 | 22.99 |           0 |  2 | Drama              |
|         3000 | Francis Ford Coppola | Apocalypse Now              | Action & Adventure | apocalypse_now.gif |     9180 | R      |          1979 | 22.99 |           0 |  1 | Comedy             |
|         3000 | Francis Ford Coppola | Apocalypse Now              | Action & Adventure | apocalypse_now.gif |     9180 | R      |          1979 | 22.99 |           0 |  0 | Action & Adventure |
|         3001 | Michael Bay          | Bad Boys                    | Action & Adventure | badboys.gif        |     7140 | R      |          1995 | 15.99 |         782 |  5 | Suspense           |
|         3001 | Michael Bay          | Bad Boys                    | Action & Adventure | badboys.gif        |     7140 | R      |          1995 | 15.99 |         782 |  4 | Science Fiction    |
|         3001 | Michael Bay          | Bad Boys                    | Action & Adventure | badboys.gif        |     7140 | R      |          1995 | 15.99 |         782 |  3 | Horror             |
|         3001 | Michael Bay          | Bad Boys                    | Action & Adventure | badboys.gif        |     7140 | R      |          1995 | 15.99 |         782 |  2 | Drama              |
```

2. Execute:

```
SELECT *
FROM video_recordings vr, video_categories vc
WHERE vr.category = vc.name;
```

Note the cross-product effect of joining two tables when restricted on the appropriate keys. Record the number of rows generated. Explain the purpose of the join.

The first ten results of the above query are shown in the screenshot below; the total amount of rows returned was 55 which is equal to number of rows in the video_recordings table. The purpose of the above query is to join the two tables where the category column of video_recordings is matches the name column of video_categories.

```
mysql> SELECT * FROM video_recordings vr, video_categories vc WHERE vr.category=vc.name;
+--------------+--------------------+-------------------------------+-----------------+-----------------------+----------+--------+---------------+-------+-------------+----+--------------------+
| recording_id | director           | title                         | category        | image_name            | duration | rating | year_released | price | stock_count | id | name               |
+--------------+--------------------+-------------------------------+-----------------+-----------------------+----------+--------+---------------+-------+-------------+----+--------------------+
|         3000 | Francis Ford Coppola | Apocalypse Now              | Action & Adventure | apocalypse_now.gif |     9180 | R      |          1979 | 22.99 |           0 |  0 | Action & Adventure |
|         3001 | Michael Bay          | Bad Boys                    | Action & Adventure | badboys.gif        |     7140 | R      |          1995 | 15.99 |         782 |  0 | Action & Adventure |
|         3002 | Mel Gibson           | Braveheart                  | Action & Adventure | braveheart.gif     |    10620 | R      |          1995 | 14.99 |         582 |  0 | Action & Adventure |
|         3003 | Mimi Leder           | Deep Impact                 | Action & Adventure | deep_impact.gif    |     5700 | PG-13  |          1998 | 11.99 |         501 |  0 | Action & Adventure |
|         3004 | J. Robert Wagoner    | Disco Godfather             | Action & Adventure | disco_godfather.gif |    5580 | R      |          1993 | 10.99 |         872 |  0 | Action & Adventure |
|         3005 | Stanley Kubrick      | Full Metal Jacket           | Action & Adventure | full_metal_jacket.gif |  7020 | R      |          1987 | 16.99 |         872 |  0 | Action & Adventure |
|         3006 | Roland Emmerich      | Independence Day            | Action & Adventure | independence_day.gif |   8700 | PG-13  |          1996 | 16.99 |           0 |  0 | Action & Adventure |
|         3007 | John McTiernan       | The Hunt for Red October    | Action & Adventure | hunt_for_red_october.gif | 8100 | PG     |          1990 | 10.99 |         618 |  0 | Action & Adventure |
|         3008 | Michael Bay          | The Rock                    | Action & Adventure | the_rock.gif       |     8160 | R      |          1996 | 20.99 |         514 |  0 | Action & Adventure |
|         3009 | Tony Scott           | Top Gun                     | Action & Adventure | top_gun.gif        |     6600 | PG     |          1986 | 11.99 |         499 |  0 | Action & Adventure |
```

For the remaining questions, use your relational schema.

3. List the number of videos for each video category.

   Query:

```
SELECT c.name, f.category_id, COUNT(f.film_id) AS num_videos
FROM films f, categories c
WHERE c.category_id = f.category_id
GROUP BY f.category_id;
```

   Results:

```
mysql> SELECT c.name, f.category_id, COUNT(f.film_id) AS num_videos
    -> FROM films f, categories c
    -> WHERE c.category_id = f.category_id
    -> GROUP BY f.category_id;
+--------------------+-------------+------------+
| name               | category_id | num_videos |
+--------------------+-------------+------------+
| Action & Adventure |           0 |         10 |
| Comedy             |           1 |         10 |
| Drama              |           2 |         10 |
| Horror             |           3 |          8 |
| Science Fiction    |           4 |          9 |
| Suspense           |           5 |          8 |
+--------------------+-------------+------------+
6 rows in set (0.00 sec)
```

   The above query uses the films table by grouping by each category_id and then counting the number of film_ids in each group. The category table is used to get the category name that corresponds with each category_id. The query returned a total of six rows.

4. List the number of videos for each video category where the inventory is non-zero

   Query:

```
SELECT c.name, f.category_id, COUNT(f.film_id) AS num_videos
FROM films f, categories c
WHERE c.category_id = f.category_id AND f.stock_count != 0
GROUP BY f.category_id;
```

   Results:

```
mysql> SELECT c.name, f.category_id, COUNT(f.film_id) AS num_videos
    -> FROM films f, categories c
    -> WHERE c.category_id = f.category_id AND f.stock_count != 0
    -> GROUP BY f.category_id;
+--------------------+-------------+------------+
| name               | category_id | num_videos |
+--------------------+-------------+------------+
| Action & Adventure |           0 |          8 |
| Comedy             |           1 |          8 |
| Drama              |           2 |          9 |
| Horror             |           3 |          6 |
| Science Fiction    |           4 |          8 |
| Suspense           |           5 |          6 |
+--------------------+-------------+------------+
6 rows in set (0.00 sec)
```

The above query is the doing the same thing as the query in #3, however, the above query adds another constraint to the WHERE clause to filter the data such that the stock_count for the film was nonzero. These results make sense because the number of videos for each categories is less than the total film counts in #3. The query returned a total of six rows.

5. For each actor, list the video categories that actor has appeared in.

Query:

```
SELECT a.name, GROUP_CONCAT(c.name)
FROM actors a, films f, categories c, actor_film af
WHERE af.actor_id = a.actor_id
AND af.film_id = f.film_id
AND f.category_id = c.category_id
GROUP BY a.name;
```

Results:

name	GROUP_CONCAT(c.name)
Adam Baldwin	Action & Adventure
Adrian Pasdar	Action & Adventure
Adrienne Corri	Science Fiction
Adrienne King	Horror
Alec Baldwin	Action & Adventure
Alec Guinness	Science Fiction
Alfre Woodard	Comedy
Alice Krige	Science Fiction
Amy Irving	Horror
Angela Bassett	Science Fiction

The screenshot above shows the first ten results of the query; the query returned a total of 335 rows which makes sense because that is the number of distinct actors in the database. The above query joins the tables: actors, films, actor_film, and categories, then it filters the data to get the records in with matching actor_ids, film_ids, and category_ids to get the categories of each of the films that each actor has been in. Then it concatenates the different categories each actor has been in.

6. Which actors have appeared in movies in different video categories?

Query:

```
SELECT a.name, c.name
FROM actors a, films f, categories c, actor_film af
WHERE af.actor_id = a.actor_id
AND af.film_id = f.film_id
AND f.category_id = c.category_id
GROUP BY a.name
HAVING COUNT(*) > 1;
```

Results:

I spent a lot of time trying to find a query that would answer the question, but the closest thing I could come up with (it does not return any results. I attempted to filter the data to find all the categories that each actgor has been in and then group those results by each actor and then only choose the actors that had more than one category.

7. Which actors have not appeared in a comedy?

Query:

```
SELECT a.name, c.name
FROM actors a, films f, categories c, actor_film af
WHERE a.actor_id = af.actor_id
AND af.film_id = f.film_id
AND f.category_id = c.category_id
HAVING c.name != "Comedy";
```

Results:

name	name
Marlon Brando	Action & Adventure
Martin Sheen	Action & Adventure
Robert Duvall	Action & Adventure
Frederic Forrest	Action & Adventure
Dennis Hopper	Action & Adventure
Scott Glenn	Action & Adventure
Harrison Ford	Action & Adventure
Laurence Fishburne	Action & Adventure
Sam Bottoms	Action & Adventure
Will Smith	Action & Adventure

The screenshot above shows the first ten results of the above query; the total number of row returned by the query was 269. The query first gets the corresponding records from the actors, actor_film, films, and categories tables to get the categories that each actor has been in. Then it filters the table further to only get the actors that have corresponding categories that are not 'Comedy'.

8. Which actors have appeared in both a comedy and an action adventure movie?

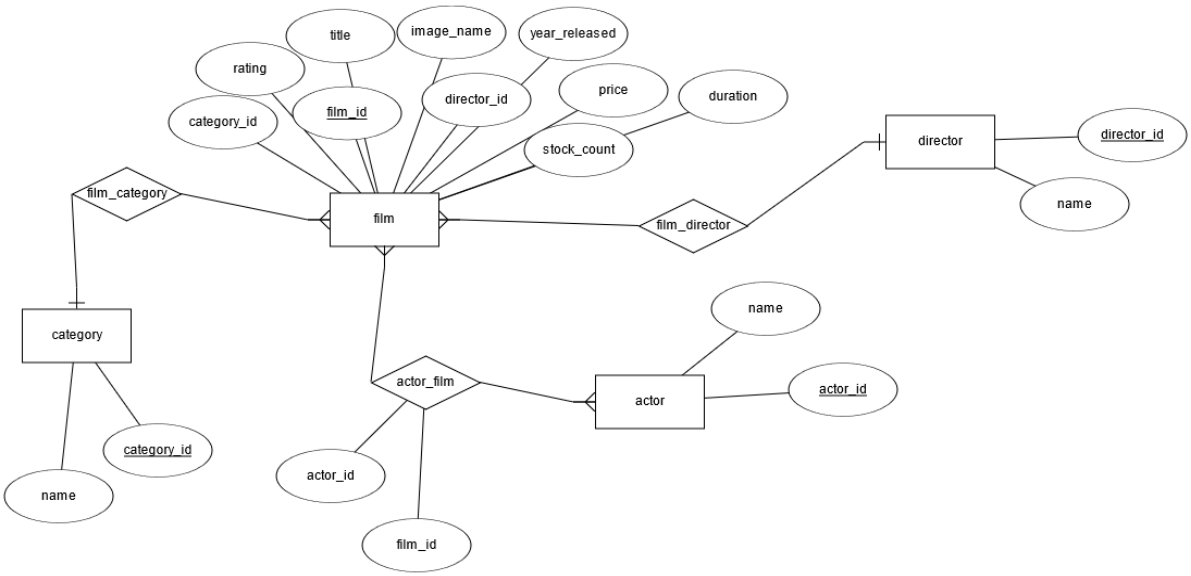
Query:

```
SELECT a.name
FROM actors a, films f, categories c, actor_film af
WHERE (a.actor_id = af.actor_id
AND af.film_id = f.film_id
AND f.category_id = c.category_id)
AND (c.name = "Comedy" OR c.name = "Action & Adventure")
GROUP BY a.actor_id
HAVING COUNT(*) > 1;
```

Results:

Unfortunately, I was unable to write a query to answer this question. The query above was as close as I could get but the query itself does not return anything. In theory, this query should filter all of the tables to match actors to the categories they have been in, and also filter for only the actors that have been in a "Comedy" or an "Action & Adventure". Then it would group the actors by their ids and then only take the actors that had a count of greater than 1 with would mean that the actor was in both a comedy and an action & adventure film.

Final ERD:



Note to the professor

I apologize for how late I turned this lab in. Between studying for the exam and studying for other classes I couldn't get it done during the week. Also, some of my queries don't work and I spent as much time as I could trying to get them to work without luck; I planned on having more time to work on it on Saturday but there was a scheduling error with my job and I ended up having to work for much longer than anticipated. I know this doesn't excuse my work, but I would appreciate it if I could meet about where I went wrong in the lab so I can understand my mistakes.