The movie database

Team Members:

Sree Vandana Nadipalli; PSU ID: 961365217

Shashank Shekhar; PSU ID: 949523554

Introduction:

The movie database TMDB, which is biggest source of metadata is a community-built movie and tv show database created to answer many questions related to movie industry.

In our movie database domain, we have taken subset of this huge metadata (around 5000 movies) which contains information about Movie Names, Lead Actors Names, Genre, Budget of that Movie, Revenue, Release date, Rating given to that movie, Run Time of the movie etc.

For our project we are considering dataset from *www.kaggles.com*, the link to dataset that we are using is *https://www.kaggle.com/tmdb/tmdb-movie-metadata*

The ER Diagram for the domain (The Movie Database):

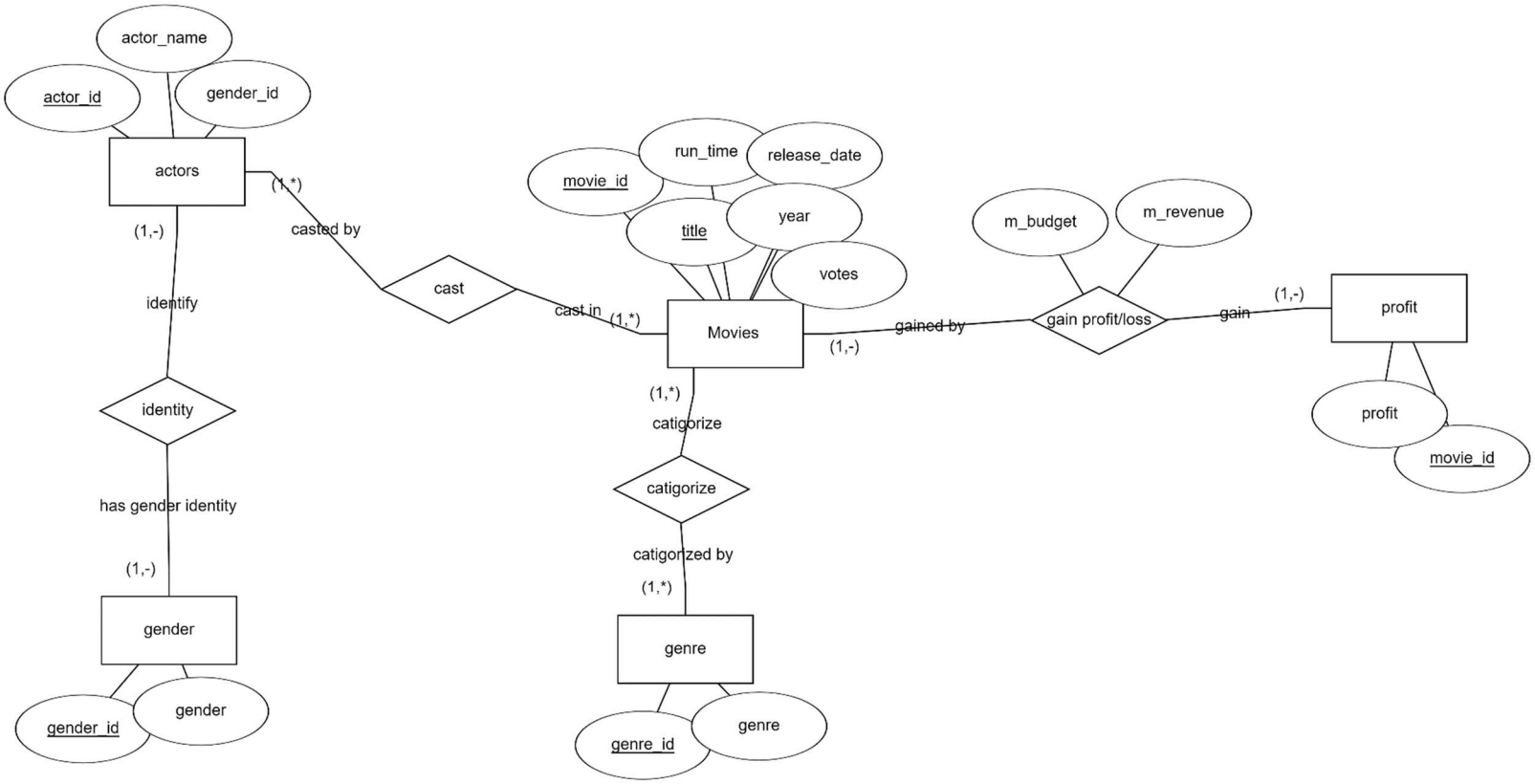


Fig 1: ER Diagram of "The Movie Database"

Relational Schema:

Using the "tmdb-movie-metadata" data source, we are able to generate 8 relational database tables containing different datatypes, attributes, relations and cardinalities. The relational schema is described below.

i. **Movies (<u>movie_id</u>, title, release_date, year, run_time, votes)**
   Movies.title has UNIQUE constraint on it

ii. **Genre (<u>genre_id,</u> genre_name)**

iii. **Movie_Genre (movie_id, genre_id)**
   movie_id is a foreign key referring to Movies.movie_id
   genre_id is a foreign key referring to Genre.genre_id

iv. **Budget_Revenue (movie_id, m_budget, m_revenue)**
   movie_id is a foreign key referring to Movies.movie_id

v. **Profit (<u>movie_id</u>, profit)**
   movie_id is a foreign key referring to Movies.movie_id
   (Here movie_id is both primary key and a foreign key; Profit entity has one-to-one relation with Movies entity)

vi. **Gender(<u>gender_id</u>, gender)**

vii. **Actors (<u>actor_id</u>, actor_name, gender_id)**
   gender_id is a foreign key references to Gender.gender_id

viii. **Movie_Actors (<u>movie_id, actor_id</u>)**
   movie_id is a foreign key references to Movies.movie_id
   actor_id is a foreign key references to Actors.actor_id

Creating Database Tables and Populating them with data:

a. Creating Tables

- In this step all tables are created, attributes are assigned with keys, Primary Keys and Foreign Keys, and also UNIQUE constraint on few attributes.

- Below are the SQL commands for creating tables, keys and Constraints.

i. Movies (movie_id, title, release_date, year, run_time, votes)

```
postgres=# CREATE TABLE movies(
postgres(# movie_id int NOT NULL,
postgres(# title varchar(80),
postgres(# release_date date,
postgres(# year int,
postgres(# run_time int,
postgres(# votes float,
postgres(# PRIMARY KEY (movie_id)
postgres(# );
CREATE TABLE
```

- Adding UNIQUE constraint on movies.title

```
postgres=# alter table movies
postgres-# ADD UNIQUE(title);
ALTER TABLE
```

ii. Genre (genre_id, genre_name)

```
postgres=# CREATE TABLE genre(
postgres(# genre_id int NOT NULL,
postgres(# genre_name varchar(20),
postgres(# PRIMARY KEY (genre_id)
postgres(# );
CREATE TABLE
```

iii. Movie_Genre (movie_id, genre_id)

```
postgres=# CREATE TABLE movie_genre(
postgres(# movie_id int NOT NULL,
postgres(# genre_id int,
postgres(# PRIMARY KEY (movie_id, genre_id),
postgres(# FOREIGN KEY (movie_id) REFERENCES movies(movie_id),
postgres(# FOREIGN KEY (genre_id) REFERENCES genre(genre_id)
postgres(# );
CREATE TABLE
```

iv. Budget_revenue (movie_id, m_budget, m_revenue)

```
postgres=# CREATE TABLE budget_revenue(
postgres(# movie_id int NOT NULL,
postgres(# m_budget bigint,
postgres(# m_revenue bigint,
postgres(# FOREIGN KEY (movie_id) REFERENCES movies(movie_id)
postgres(# );
CREATE TABLE
```

v. Profit (movie_id, m_profit)

```
postgres=# CREATE TABLE profit(
postgres(# movie_id int NOT NULL,
postgres(# m_profit bigint,
postgres(# PRIMARY KEY (movie_id),
postgres(# FOREIGN KEY (movie_id) REFERENCES movies(movie_id)
postgres(# );
CREATE TABLE
```

vi. Gender(gender_id, gender)

```
postgres=# CREATE TABLE gender(
postgres(# gender_id int NOT NULL,
postgres(# gender int,
postgres(# PRIMARY KEY (gender_id)
postgres(# );
CREATE TABLE
```

vii. Actors (actor_id, actor_name, gender_id)

```
postgres=# CREATE TABLE actors(
postgres(# actor_id int NOT NULL,
postgres(# actor_name varchar(86),
postgres(# gender_id int,
postgres(# PRIMARY KEY (actor_id),
postgres(# FOREIGN KEY (gender_id) REFERENCES gender(gender_id)
postgres(# );
CREATE TABLE
```

viii. Movie_Actors (movie_id, actor_id)

```
postgres=# CREATE TABLE movie_actors(
postgres(# movie_id int NOT NULL,
postgres(# actor_id int,
postgres(# PRIMARY KEY (movie_id, actor_id),
postgres(# FOREIGN KEY (movie_id) REFERENCES movies(movie_id),
postgres(# FOREIGN KEY (actor_id) REFERENCES actors(actor_id)
postgres(# );
CREATE TABLE
```

b. Populating tables with Data:

For inserting data into each table COPY command is used and data is imported from .csv file.
COPY <Table_Name> FROM <File_Location_Path> DELIMITER ',' CSV HEADER;

i. Movies (movie_id, title, release_date, year, run_time, votes)

```
postgres=# COPY movies FROM 'D:\DBMS\project\data\csv files\movies.csv' DELIMITER ',' CSV HEADER;
COPY 1010
postgres=# select * from movies;
 movie_id |                          title                          | release_date | year | run_time | votes
----------+---------------------------------------------------------+--------------+------+----------+-------
    19995 | Avatar                                                  | 2009-12-10   | 2009 |      162 |   7.2
      285 | Pirates of the Caribbean At World s End                 | 2007-05-19   | 2007 |      169 |   6.9
   206647 | Spectre                                                 | 2015-10-26   | 2015 |      148 |   6.3
    49026 | The Dark Knight Rises                                   | 2012-07-16   | 2012 |      165 |   7.6
    49529 | John Carter                                             | 2012-03-07   | 2012 |      132 |   6.1
```

ii. Genre (genre_id, genre_name)

```
postgres=# COPY genre FROM 'D:\DBMS\project\data\csv files\genre.csv' DELIMITER ',' CSV HEADER;
COPY 18
postgres=# select * from genre;
 genre_id |   genre_name
----------+-----------------
       12 | Adventure
       14 | Fantasy
       16 | Animation
       18 | Drama
       27 | Horror
```

iii. Movie_Genre (movie_id, genre_id)

```
postgres=# COPY movie_genre FROM 'D:\DBMS\project\data\csv files\movie_genre.csv' DELIMITER ',' CSV HEADER;
COPY 1936
postgres=# select * from movie_genre;
 movie_id | genre_id
----------+----------
    19995 |       28
      285 |       12
   206647 |       28
    49026 |       28
    49529 |       28
```

iv. Budget_revenue (movie_id, m_budget, m_revenue)

```
postgres=# COPY budget_revenue FROM 'D:\DBMS\project\data\csv files\budget_revenue.csv' DELIMITER ',' CSV HEADER;
COPY 1010
postgres=# select * from budget_revenue;
 movie_id | m_budget  | m_revenue
----------+-----------+------------
    19995 | 237000000 | 2787965087
      285 | 300000000 |  961000000
   206647 | 245000000 |  880674609
    49026 | 250000000 | 1084939099
    49529 | 260000000 |  284139100
```

v. Profit (movie_id, m_profit)

unlike other tables where data is imported from .csv file, for this we imported data from another table(budget_revenue table).

Here profit.m_profit = budget_revenue.m_revenue - budget_revenue.m_budget

```
postgres=#  INSERT INTO profit(movie_id, m_profit)
postgres-# SELECT movie_id, m_revenue-m_budget from budget_revenue;
INSERT 0 1010
postgres=# select * from profit;
 movie_id |  m_profit
----------+------------
    19995 | 2550965087
      285 |  661000000
   206647 |  635674609
    49026 |  834939099
    49529 |   24139100
```

vi. Gender(gender_id, gender)

For this table data is entered manually using insert command.

Insert into table gender(gender_id, values)
Values (0, 'Not Specified'),
(1, 'Female'),
(2, 'Male');

```
postgres=# select * from gender;
 gender_id |    gender
-----------+---------------
         0 | Not Specified
         1 | Female
         2 | Male
(3 rows)
```

vii. Actors (actor_id, actor_name, gender_id)

```
postgres=# COPY actors FROM 'D:\DBMS\project\data\csv files\actors.csv' DELIMITER ',' CSV HEADER;
COPY 783
postgres=# select * from actors;
 actor_id |         actor_name          | gender_id
----------+-----------------------------+-----------
    65731 | Sam Worthington             |         2
       85 | Johnny Depp                 |         2
     8784 | Daniel Craig                |         2
     3894 | Christian Bale              |         2
    60900 | Taylor Kitsch               |         2
     2219 | Tobey Maguire               |         2
```

viii. Movie_Actors (movie_id, actor_id)

```
postgres=# COPY movie_actors FROM 'D:\DBMS\project\data\csv files\movie_actors.csv' DELIMITER ',' CSV HEADER;
COPY 2020
postgres=# select * from movie_actors;
 movie_id | actor_id
----------+----------
    19995 |    65731
      285 |       85
   206647 |     8784
    49026 |     3894
    49529 |    60900
```

20 Questions translation into SQL Queries:

In this step, the questions presented in part 1 are translated into SQL Queries

1. **Which movies were release on same date as the movie "Harry Potter and the Goblet of Fire"**

   select m1.movie_id, m1.title, m1.release_date, m1.year from movies m1, movies m2
   where m2.title = 'Harry Potter and the Goblet of Fire' and
   m1.release_date = m2.release_date;

```
postgres=# select m1.movie_id, m1.title, m1.release_date, m1.year from movies m1, movies m2
postgres-# where m2.title = 'Harry Potter and the Goblet of Fire' and
postgres-# m1.release_date = m2.release_date;
 movie_id |               title                | release_date | year
----------+------------------------------------+--------------+------
      674 | Harry Potter and the Goblet of Fire | 2005-11-05   | 2005
(1 row)
```

   - Seems like it is the only movie released on that day.

2. **which movies have low budget but high revenue?**

   select m.movie_id, m.title, br.m_budget, br.m_revenue from budget_revenue br, movies m
   where m.movie_id = br.movie_id and
   br.m_budget < m_revenue;

```
postgres=# select m.movie_id, m.title,br.m_budget, br.m_revenue from budget_revenue br, movies m
postgres-# where m.movie_id = br.movie_id and
postgres-# br.m_budget < m_revenue;
 movie_id |                          title                          | m_budget  | m_revenue
----------+---------------------------------------------------------+-----------+------------
    19995 | Avatar                                                  | 237000000 | 2787965087
      285 | Pirates of the Caribbean At World s End                 | 300000000 |  961000000
   206647 | Spectre                                                 | 245000000 |  880674609
    49026 | The Dark Knight Rises                                   | 250000000 | 1084939099
    49529 | John Carter                                             | 260000000 |  284139100
```

3. **List the name and id of all movies that has a budget over 150000000, with female actors**

```
select m.movie_id, m.title, a.actor_name, br.m_budget from movies m, actors a, budget_revenue br,
gender g, movie_actors ma
where m.movie_id = br.movie_id and
br.m_budget > 150000000 and
ma.movie_id = br.movie_id and
ma.actor_id = a.actor_id and
a.gender_id = g.gender_id and

g. gender = 'Female';
```

```
postgres=# select m.movie_id, m.title, a.actor_name, br.m_budget from movies m, actors a, budget_revenue br,
gender g, movie_actors ma
postgres-# where m.movie_id = br.movie_id and
postgres-# br.m_budget > 150000000 and
postgres-# ma.movie_id = br.movie_id and
postgres-# ma.actor_id = a.actor_id and
postgres-# a.gender_id = g.gender_id and
postgres-# g. gender = 'Female';
 movie_id |                          title                          |      actor_name      | m_budget
----------+---------------------------------------------------------+----------------------+-----------
     2268 | The Golden Compass                                      | Dakota Blue Richards | 180000000
      254 | King Kong                                               | Naomi Watts          | 207000000
      597 | Titanic                                                 | Kate Winslet         | 200000000
    12155 | Alice in Wonderland                                     | Mia Wasikowska       | 200000000
    62177 | Brave                                                   | Kelly Macdonald      | 185000000
```

4. **Which actor has appeared in most movies and what is that count?**

```
select a.actor_id, a.actor_name, count(ma.actor_id) as max_movies_acted from movie_actors ma,
actors a
where a.actor_id = ma.actor_id
group by (a.actor_id, a.actor_name)
having count(ma.actor_id) IN (select max(ma_count) from
                                (select count(actor_id) as ma_count from movie_actors
                                 group by (actor_id)) as max_ma_count);
```

```
postgres=# select a.actor_id, a.actor_name, count(ma.actor_id) as max_movies_acted from movie_actors ma, actors a
postgres-# where a.actor_id = ma.actor_id
postgres-# group by (a.actor_id, a.actor_name)
postgres-# having count(ma.actor_id) IN (select max(ma_count) from
postgres(#                               (select count(actor_id) as ma_count from movie_actors
postgres(#                                group by (actor_id)) as max_ma_count);
 actor_id |  actor_name  | max_movies_acted
----------+--------------+------------------
     1892 | Matt Damon   |               21
       62 | Bruce Willis |               21
(2 rows)
```

5. What is the highest budgeted movie per year?

```
select m.movie_id, m.title, m.year, br.m_budget as max_budget from movies m, budget_revenue br,
(select m.year as movie_year, max(br.m_budget) as max_budget from budget_revenue br, movies m
where m.movie_id = br.movie_id
group by(m.year)) as max_budget_year

where m.movie_id = br.movie_id and
m.year = max_budget_year.movie_year and
br.m_budget = max_budget_year.max_budget

order by m.year DESC;
```

```
postgres=# select m.movie_id, m.title, m.year, br.m_budget as max_budget from movies m, budget_revenue br,
postgres-# (select m.year as movie_year, max(br.m_budget) as max_budget from budget_revenue br, movies m
postgres(# where m.movie_id = br.movie_id
postgres(# group by(m.year)) as max_budget_year
postgres-#
postgres-# where m.movie_id = br.movie_id and
postgres-# m.year = max_budget_year.movie_year and
postgres-# br.m_budget = max_budget_year.max_budget
postgres-#
postgres-# order by m.year DESC;
 movie_id |                  title                  | year | max_budget
----------+-----------------------------------------+------+------------
   271110 | Captain America Civil War               | 2016 |  250000000
   209112 | Batman v Superman Dawn of Justice       | 2016 |  250000000
    99861 | Avengers Age of Ultron                  | 2015 |  280000000
   122917 | The Hobbit The Battle of the Five Armies | 2014 |  250000000
   127585 | X Men Days of Future Past               | 2014 |  250000000
    57201 | The Lone Ranger                         | 2013 |  255000000
    49529 | John Carter                             | 2012 |  260000000
     1865 | Pirates of the Caribbean On Stranger Tides | 2011 |  380000000
```

6. list the name of the movies and its genre which has the highest votes

```
select m.title, g.genre_name, m.votes from movies m, genre g, movie_genre mg
where m.movie_id = mg.movie_id and
mg.genre_id = g.genre_id and
m.votes = (select max(m.votes) from movies m);
```

```
postgres=# select m.title, g.genre_name, m.votes from movies m, genre g, movie_genre mg
postgres-# where m.movie_id = mg.movie_id and
postgres-# mg.genre_id = g.genre_id and
postgres-# m.votes = (select max(m.votes) from movies m);
   title    | genre_name | votes
------------+------------+-------
 Fight Club | Drama      |   8.3
(1 row)
```

7. **find the actor who has the most movies under the genre "Action"**

select a.actor_name, count(ma.actor_id) as max_action_movies from genre g, movie_genre mg, movie_actors ma, actors a

where ma.movie_id = mg.movie_id and
mg.genre_id = g.genre_id and
a.actor_id = ma.actor_id and
g.genre_name = 'Action'

GROUP BY (a.actor_name)
HAVING count(ma.actor_id) = (

select max(action_count)from
(select count(ma.actor_id) as action_count from genre g, movie_genre mg, movie_actors ma, actors a
where ma.movie_id = mg.movie_id and
mg.genre_id = g.genre_id and
a.actor_id = ma.actor_id and
g.genre_name = 'Action'
GROUP BY (a.actor_name)) as max_action_movies

);

```
postgres=# select a.actor_name, count(ma.actor_id) as max_action_movies from genre g, movie_genre mg, movie_actors ma, ac
tors a
postgres-#
postgres-# where ma.movie_id = mg.movie_id and
postgres-# mg.genre_id = g.genre_id and
postgres-# a.actor_id = ma.actor_id and
postgres-# g.genre_name = 'Action'
postgres-#
postgres-# GROUP BY (a.actor_name)
postgres-# HAVING count(ma.actor_id) = (
postgres(#
postgres(# select max(action_count)from
postgres(# (select count(ma.actor_id) as action_count from genre g, movie_genre mg, movie_actors ma, actors a
postgres(# where ma.movie_id = mg.movie_id and
postgres(# mg.genre_id = g.genre_id and
postgres(# a.actor_id = ma.actor_id and
postgres(# g.genre_name = 'Action'
postgres(# GROUP BY (a.actor_name)) as max_action_movies
postgres(#
postgres(# );
  actor_name  | max_action_movies
--------------+-------------------
 Bruce Willis |                13
(1 row)
```

8. find the Names of the male and female actors who has a total revenue over 1000000000

```
select DISTINCT a.actor_name, g.gender from budget_revenue br, movie_actors ma, gender g, actors a
where br.movie_id = ma.movie_id and
ma.actor_id = a.actor_id and
a.gender_id = g.gender_id and
br.m_revenue > 1000000000
ORDER BY (a.actor_name);
```

```
postgres=# select DISTINCT a.actor_name, g.gender from budget_revenue br, movie_actors ma, gender g, actors a
postgres-# where br.movie_id = ma.movie_id and
postgres-# ma.actor_id = a.actor_id and
postgres-# a.gender_id = g.gender_id and
postgres-# br.m_revenue > 1000000000
postgres-# ORDER BY (a.actor_name);
     actor_name      | gender
---------------------+--------
 Bryce Dallas Howard | Female
 Chris Evans         | Male
 Chris Hemsworth     | Male
 Chris Pratt         | Male
 Christian Bale      | Male
```

9. how many Number of movies were acted by each actor.

```
select a.actor_id, a.actor_name, count(ma.movie_id) as number_of_movies_acted
from actors a, movie_actors ma
where a.actor_id = ma.actor_id
GROUP BY (a.actor_id, a.actor_name)
ORDER BY (a.actor_id, a.actor_name);
```

```
postgres=# select a.actor_id, a.actor_name, count(ma.movie_id) as number_of_movies_acted
postgres-# from actors a, movie_actors ma
postgres-# where a.actor_id = ma.actor_id
postgres-# GROUP BY (a.actor_id, a.actor_name)
postgres-# ORDER BY (a.actor_id, a.actor_name);
 actor_id |         actor_name          | number_of_movies_acted
----------+-----------------------------+------------------------
        2 | Tommy Lee Jones             |                      1
        3 | Harrison Ford               |                     12
       13 | Albert Brooks               |                      1
       14 | Ellen DeGeneres             |                      1
       20 | Elizabeth Perkins           |                      1
       31 | Tom Hanks                   |                     17
```

10.find the names of the actor who had no releases in the year 2008

```
select a.actor_name from actors a
where a.actor_name NOT IN (
select a.actor_name from actors a, movie_actors ma, movies m1, movies m2
where a.actor_id = ma.actor_id and
ma.movie_id = m1.movie_id and
m1.year = 2008 and
m1.year = m2.year
)
ORDER BY (a.actor_name)
```

```
postgres=# select a.actor_name from actors a
postgres-# where a.actor_name NOT IN (
postgres(#
postgres(# select a.actor_name from actors a, movie_actors ma, movies m1, movies m2
postgres(# where a.actor_id = ma.actor_id and
postgres(# ma.movie_id = m1.movie_id and
postgres(# m1.year = 2008 and
postgres(# m1.year = m2.year
postgres(#
postgres(# )
postgres-#
postgres-# ORDER BY (a.actor_name);
        actor_name
---------------------------
 Aaran Thomas
 Aaron Eckhart
 Aaron Kwok
 Aaron Paul
 Abbie Cornish
 Adam Beach
 Adrien Brody
```

11. find the movies with votes > 8 and budget less than 105000000 and had revenue greater than 105000000

```
select m.title from movies m, budget_revenue br
where m.votes > 8 and
br.m_budget < 105000000 and
br.m_revenue > 105000000
```

```
postgres=# select m.title from movies m, budget_revenue br
postgres-# where m.votes > 8 and
postgres-# br.m_budget < 105000000 and
postgres-# br.m_revenue > 105000000;
                    title
-----------------------------------------------
 The Dark Knight
 Interstellar
 Inception
 The Lord of the Rings The Return of the King
 Fight Club
```

12. What is the budget spent on each genre?

```
select g.genre_name, SUM(br.m_budget) as budget_on_genre
from genre g, movie_genre mg, budget_revenue br
where br.movie_id = mg.movie_id and
mg.genre_id = g.genre_id
GROUP BY (g.genre_name)
ORDER BY (g.genre_name);
```

```
postgres=# select g.genre_name, SUM(br.m_budget) as budget_on_genre
postgres-# from genre g, movie_genre mg, budget_revenue br
postgres-# where br.movie_id = mg.movie_id and
postgres-# mg.genre_id = g.genre_id
postgres-# GROUP BY (g.genre_name)
postgres-# ORDER BY (g.genre_name);
   genre_name    | budget_on_genre
-----------------+-----------------
 Action          |     41170179574
 Adventure       |     35228240400
 Animation       |     11274300067
 Comedy          |     16457836226
 Crime           |      5313500000
```

13. find the total revenue made by "James Bond" each year. (the database does not have data related to “James Bond” (among the rows we populated), so changing the actor name**)**

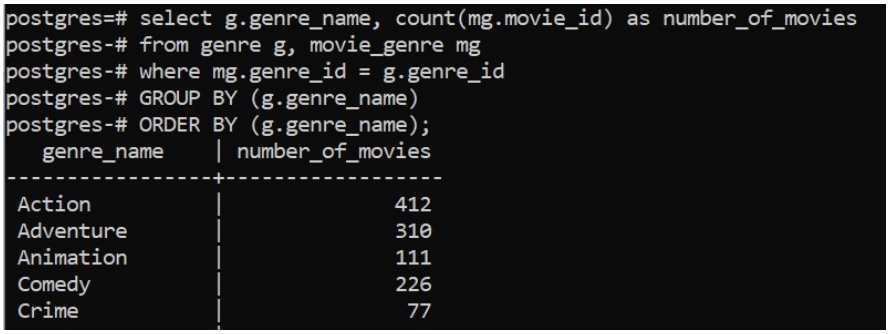
13. find the total revenue made by "James McAvoy" each year.

```
select m.year, SUM(br.m_revenue) as James_McAvoy_revenue
from movies m, budget_revenue br, movie_actors ma, actors a
where br.movie_id = m.movie_id and
ma.movie_id = br.movie_id and
a.actor_id = ma.actor_id and
a.actor_name = 'James McAvoy'
GROUP BY (m.year)
ORDER BY (m.year);
```

```
postgres=# select m.year, SUM(br.m_revenue) as James_McAvoy_revenue
postgres-# from movies m, budget_revenue br, movie_actors ma, actors a
postgres-# where br.movie_id = m.movie_id and
postgres-# ma.movie_id = br.movie_id and
postgres-# a.actor_id = ma.actor_id and
postgres-# a.actor_name = 'James McAvoy'
postgres-# GROUP BY (m.year)
postgres-# ORDER BY (m.year);
 year | james_mcavoy_revenue
------+----------------------
 2008 |            258270008
 2011 |            534592881
 2014 |            747862775
 2016 |            543934787
(4 rows)
```

14. What are the number of movies made per each genre?

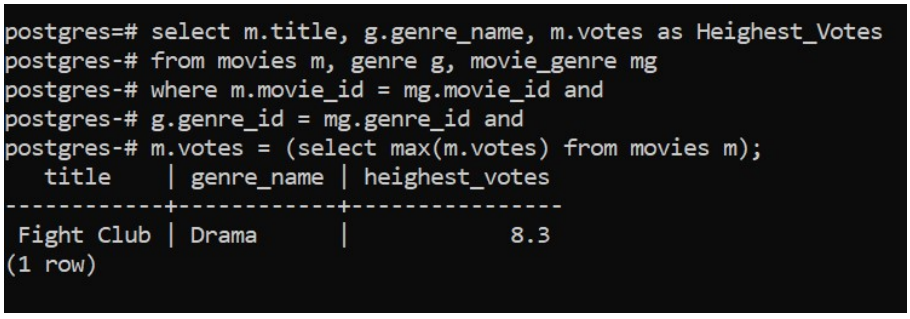
```
select g.genre_name, count(mg.movie_id) as number_of_movies
from genre g, movie_genre mg
where mg.genre_id = g.genre_id
GROUP BY (g.genre_name)
ORDER BY (g.genre_name);
```



```
postgres=# select g.genre_name, count(mg.movie_id) as number_of_movies
postgres-# from genre g, movie_genre mg
postgres-# where mg.genre_id = g.genre_id
postgres-# GROUP BY (g.genre_name)
postgres-# ORDER BY (g.genre_name);
   genre_name    | number_of_movies
-----------------+-----------------
 Action          |              412
 Adventure       |              310
 Animation       |              111
 Comedy          |              226
 Crime           |               77
```

15. find which genre of movie is most popular(highest votes)

```
select m.title, g.genre_name, m.votes as Heighest_Votes
from movies m, genre g, movie_genre mg
where m.movie_id = mg.movie_id and
g.genre_id = mg.genre_id and
m.votes = (select max(m.votes) from movies m);
```



```
postgres=# select m.title, g.genre_name, m.votes as Heighest_Votes
postgres-# from movies m, genre g, movie_genre mg
postgres-# where m.movie_id = mg.movie_id and
postgres-# g.genre_id = mg.genre_id and
postgres-# m.votes = (select max(m.votes) from movies m);
   title    | genre_name | heighest_votes
------------+------------+----------------
 Fight Club | Drama      |            8.3
(1 row)
```

16. Which year saw the most releases?

```
select m.year, count(m.movie_id) from movies m
GROUP BY (m.year)
HAVING count(m.movie_id) = (select max(m_count) from
                            (select m.year, count(m.movie_id) as m_count
                            from movies m
                            GROUP BY (m.year)) as m_count_value);
```

```
postgres=# select m.year, count(m.movie_id) as max_movies_released from movies m
postgres-# GROUP BY (m.year)
postgres-# HAVING count(m.movie_id) = (select max(m_count) from
postgres(#                            (select m.year, count(m.movie_id) as m_count
postgres(#                            from movies m
postgres(#                            GROUP BY (m.year)) as m_count_value);
 year | max_movies_released
------+--------------------
 2009 |                  55
(1 row)
```

17. How many Number of movies were released in each year?

```
select m.year, count(m.movie_id) from movies m
GROUP BY (m.year)
ORDER BY (m.year) DESC;
```

```
postgres=# select m.year, count(m.movie_id) from movies m
postgres-# GROUP BY (m.year)
postgres-# ORDER BY (m.year) DESC;
 year | count
------+-------
 2016 |    31
 2015 |    50
 2014 |    52
 2013 |    52
 2012 |    49
 2011 |    54
 2010 |    53
```

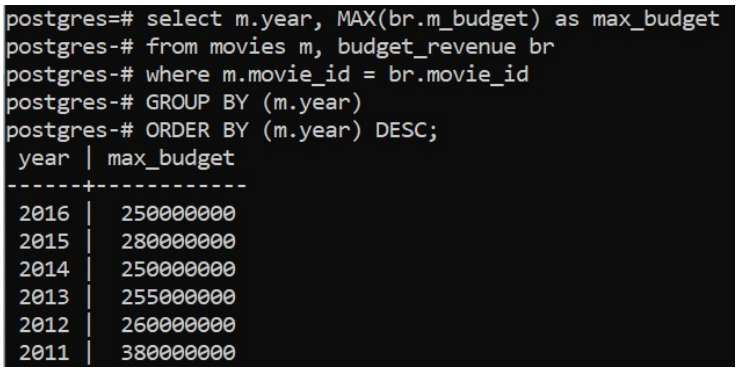
18. find the genre's released in the year 2009

```
select DISTINCT g.genre_name as genres_released_in_2009
from movies m, genre g, movie_genre mg
where m.year = 2009 and
m.movie_id = mg.movie_id and
mg.genre_id = g.genre_id;
```

```
postgres=# select DISTINCT g.genre_name as genres_released_in_2009
postgres-# from movies m, genre g, movie_genre mg
postgres-# where m.year = 2009 and
postgres-# m.movie_id = mg.movie_id and
postgres-# mg.genre_id = g.genre_id ;
 genres_released_in_2009
-------------------------
 Music
 Fantasy
 Drama
 Thriller
 Documentary
 Romance
```

19. list the highest budgeted movie in each year

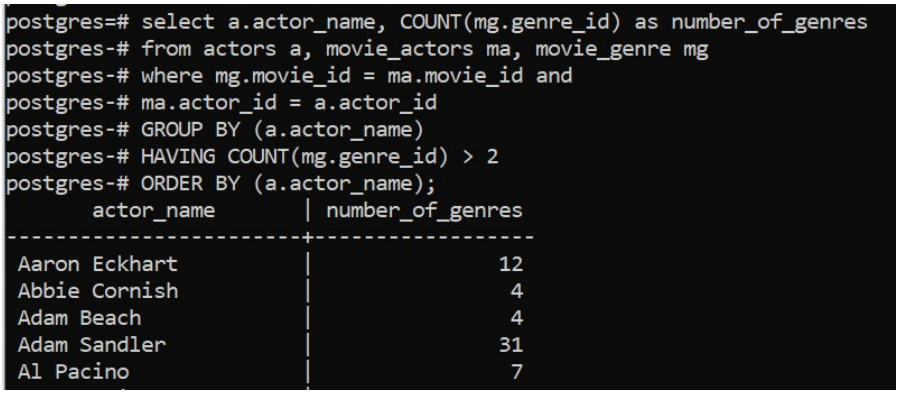
```
select m.year, MAX(br.m_budget) as max_budget
from movies m, budget_revenue br
where m.movie_id = br.movie_id
GROUP BY (m.year)
ORDER BY (m.year) DESC;
```



```
postgres=# select m.year, MAX(br.m_budget) as max_budget
postgres-# from movies m, budget_revenue br
postgres-# where m.movie_id = br.movie_id
postgres-# GROUP BY (m.year)
postgres-# ORDER BY (m.year) DESC;
 year | max_budget
------+------------
 2016 |  250000000
 2015 |  280000000
 2014 |  250000000
 2013 |  255000000
 2012 |  260000000
 2011 |  380000000
```

20. find the name of the actors who has worked in more than 2 genres

```
select a.actor_name, COUNT(mg.genre_id) as number_of_genres
from actors a, movie_actors ma, movie_genre mg
where mg.movie_id = ma.movie_id and
ma.actor_id = a.actor_id
GROUP BY (a.actor_name)
HAVING COUNT(mg.genre_id) > 2
ORDER BY (a.actor_name);
```



```
postgres=# select a.actor_name, COUNT(mg.genre_id) as number_of_genres
postgres-# from actors a, movie_actors ma, movie_genre mg
postgres-# where mg.movie_id = ma.movie_id and
postgres-# ma.actor_id = a.actor_id
postgres-# GROUP BY (a.actor_name)
postgres-# HAVING COUNT(mg.genre_id) > 2
postgres-# ORDER BY (a.actor_name);
      actor_name       | number_of_genres
-----------------------+-----------------
 Aaron Eckhart         |               12
 Abbie Cornish         |                4
 Adam Beach            |                4
 Adam Sandler          |               31
 Al Pacino             |                7
```