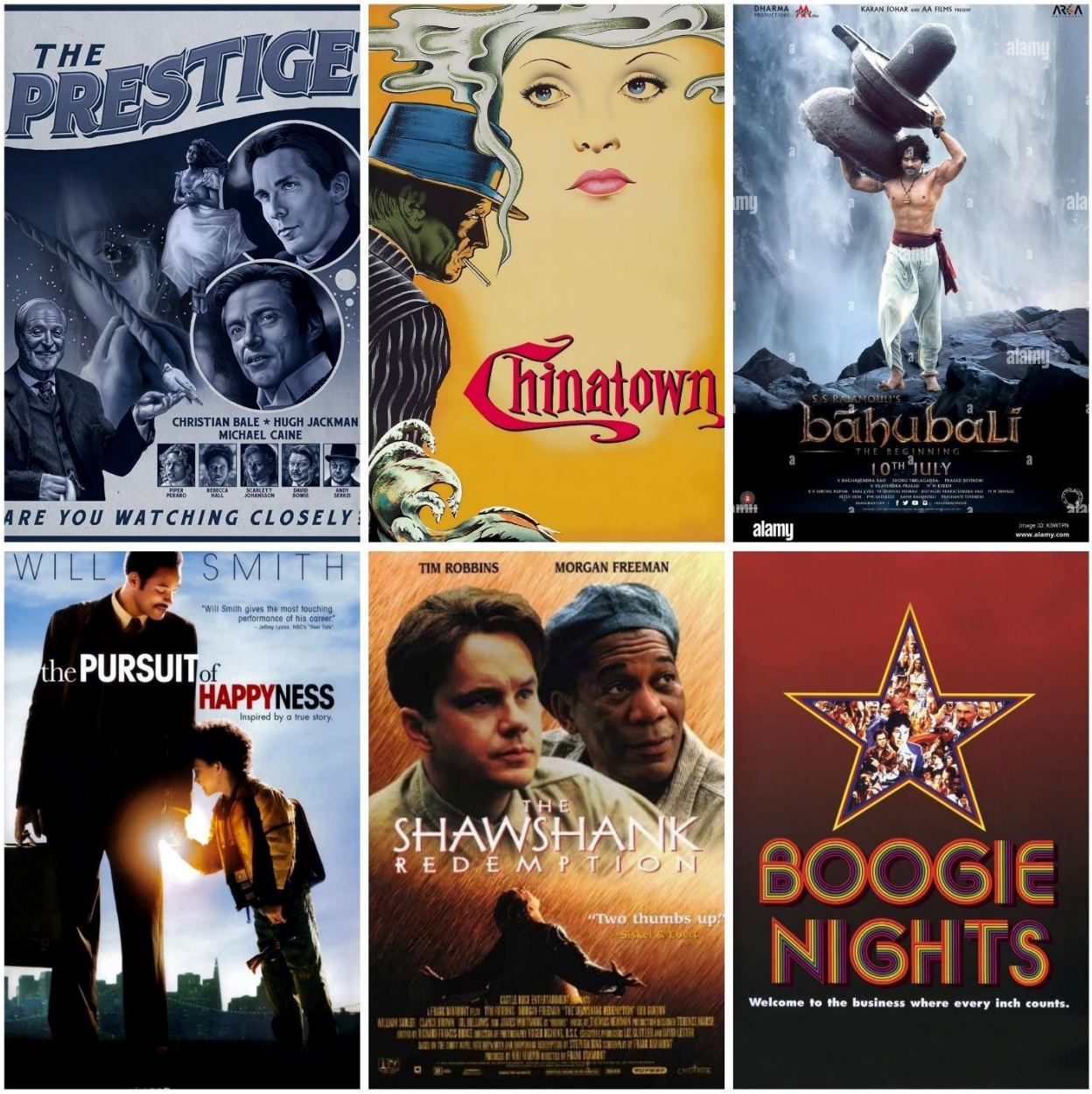
SQL GROUP PROJECT ON MOVIES DATABASE



INTRODUCTION

Most of us watch movies in our free time and like to talk about it with our friends. We often talk about actors, directors and the budget spent on a particular movie. Based on the ratings we recommend others to watch the same movie. Also, during this pandemic time, we all have been binge-watching because we couldn't go out. Therefore, we have chosen this ***Movies*** database to analyze 50 movies along with various other factors such as genre, budget, ratings etc.

INTRODUCTION TO DATA SET

This database will have 6 tables in total. Below is the snapshot of movies table followed by brief description of all the tables.

- **This dataset had few blank values for budget and earnings columns, so we inserted approximate values after checking data on internet sources.**

	A	B	C	D	E	F	G	H	I	J	K	L
1	movie_id	movie_title	movie_time	movie_lang	release_date	budget	genre_id	actor_id	director_i	movie_co	total_earnings	
2	901	Vertigo	128	English	8/24/1958	78000000	1001	101	201	1957	150000000	
3	902	The Innocents	100	English	2/19/1962	50000000	1001	103	203	1961	80000000	
4	903	Lawrence of Arabia	216	English	12/11/1962	85000000	1001	103	203	1962	120000000	
5	904	The Deer Hunter	183	English	3/8/1979	80000000	1013	104	204	1978	150000000	
6	905	Amadeus	160	English	1/7/1985	46000000	1002	105	205	1983	85000000	
7	906	Blade Runner	117	English	9/9/1982	69000000	1012	106	206	1982	130000000	
8	907	Eyes Wide Shut	159	English	9/9/1999	82000000	1010	107	207	1997	85000000	
9	908	The Usual Suspects	106	English	8/25/1995	63000000	1006	108	208	1994	59000000	
10	909	Chinatown	130	English	8/9/1974	59000000	1013	109	209	1974	98000000	
11	910	Boogie Nights	155	English	2/16/1998	66000000	1009	110	210	1997	99000000	
12	911	Annie Hall	93	English	4/20/1977	56000000	1005	111	211	1976	78000000	
13	912	Princess Mononoke	134	Japanese	10/19/2001	88000000	1003	112	212	2000	120000000	
14	913	The Shawshank Redemption	142	English	2/17/1995	29000000	1006	113	213	1995	68000000	
15	914	American Beauty	122	English	2/10/1999	78000000	1011	114	214	1998	140000000	

1) Actor:

a. actor_id – this is a unique ID for each actor and will be the primary key for this table
b. actor_first_name – this is the first name of each actor
c. actor_last_name – this is the last name of each actor
d. gender – this is the gender of each actor
e. salary-this is the salary of each actor

2) ***Genre***:

a. genre_id – this is a unique ID for each genre and will be the primary key for this table
b. genre_title – this is the description of the genre

3) ***Director***:

a. director_id-this is a unique ID for each director and will be the primary key for this table
b. director_first_name- this is the first name of the director
c. director_last_name- this is the last name of the director

4) Movie:

a. movie_id – this is the unique ID for each movie and will be the primary key for this table
b. movie_title – this column represents the name of the movie
c. movie_time– this is the year of making the movie
d. movie_lang– duration of the movie i.e., how long it was running
e. release_date– the language in which movie was casted
f. budget– this is the release date of the movie
g. genre_id-this is the ID of the genre, which is referencing the genre_id column of the table Genre and will be the foreign key in this table
h. actor_id- this is the ID of the actor, which is referencing the actor_id column of the table Actor and will be the foreign key in this table

i. director_id- this is the ID of the director, which is referencing the director_id column of the table Director and will be the foreign key in this table
j. movie_completion_year – year in which the movie was completed.
k. Total_earnings- total earnings made by movie.

5) Reviewer:

a. reviewer_id – this is the unique ID for each reviewer and will be the primary key for this table
b. reviewer_name – this is the name of the reviewer

6) Rating:

a. rating_id – this is the unique ID for each rating and will be the primary key for this table
b. movie_id –this is the ID of the movie, which is referencing the movie_id column of the table Movie and will be the foreign key in this table
c. reviewer_id – this is the ID of the reviewer, which is referencing the reviewer_id column of the table Reviewer and will be the foreign key in this table
d. rev_stars – this indicates how many stars a reviewer rated for a review of a movie
e. no_of_ratings – this indicates how many ratings a movie achieved till date

WORKSPACE

We have analyzed this database using ***PostgreSQL.*** Firstly, we have imported the dataset with below command:

```
COPY movie (movie_id,movie_title,movie_time,movie_lang,release_date,budget,genre_id,actor_id,director_id,
            movie_completion_year,total_earnings) FROM 'C:\Users\hp\Documents\Sql\movie_data_group_project.csv'
            WITH DELIMITER ',' CSV HEADER;
```

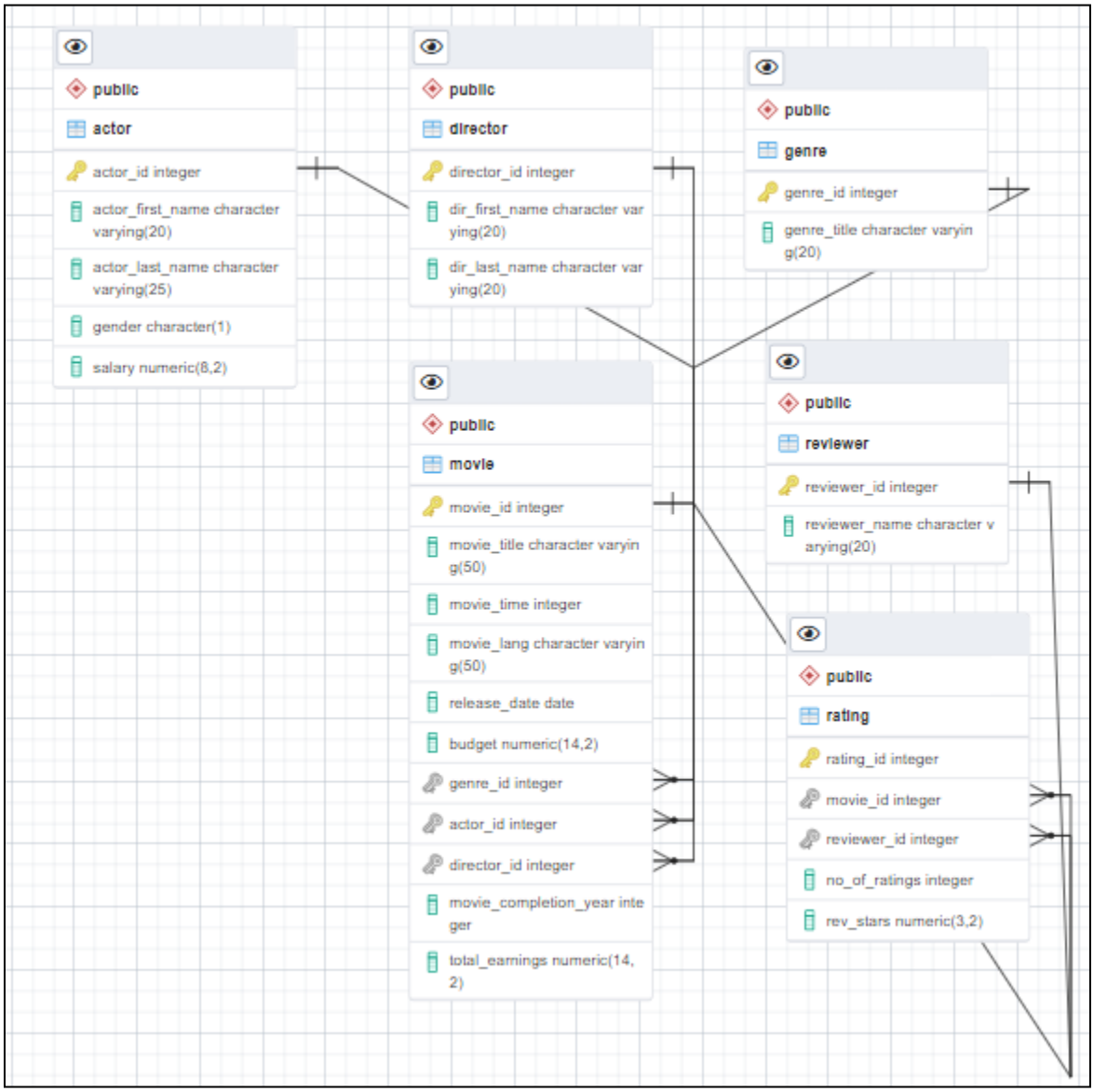
All other tables were created in the Postgre itself:

```
--2 table-Actor
CREATE TABLE Actor(
    actor_id SERIAL PRIMARY KEY,
    actor_first_name CHARACTER VARYING (20) NOT NULL,
    actor_last_name CHARACTER VARYING (25),
    gender CHARACTER (1),
    salary NUMERIC (8, 2) NOT NULL
);
--3 table-Genre
CREATE TABLE Genre(
    genre_id SERIAL PRIMARY KEY,
    genre_title CHARACTER VARYING (20)
);
```

```
--4 Dimension table-Director
CREATE TABLE Director(
    director_id SERIAL PRIMARY KEY,
    dir_first_name CHARACTER VARYING (20) NOT NULL,
    dir_last_name CHARACTER VARYING (20)
);
--5 Dimension table-Reviewer
CREATE TABLE Reviewer(
    reviewer_id SERIAL PRIMARY KEY,
    reviewer_name CHARACTER VARYING (20)
);
```

```
--6 Fact table-Rating
CREATE TABLE RATING(
    rating_id SERIAL PRIMARY KEY,
    movie_id INTEGER NOT NULL,
    reviewer_id INTEGER NOT NULL,
    no_of_ratings INTEGER,
    rev_stars NUMERIC(3,2),
    FOREIGN KEY (movie_id) REFERENCES Movie (movie_id) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (reviewer_id) REFERENCES Reviewer (reviewer_id) ON UPDATE CASCADE ON DELETE CASCADE
);
```

RELATIONAL SCHEMA



NOW LETS START ANALYZING DATA

1) Show the count of movies completed year-wise: We have used **GROUP-BY** to group movies as per same completed year and to fetch movie title along with **COUNT** we have used **temporary table**. Max count was 3 for movies released under same year.

```
--QUERY 1
--Show the count of movies completed year-wise.
WITH movie_count_per_year AS (SELECT movie_completion_year, COUNT(*) AS movie_count
FROM movie
GROUP BY movie_completion_year
)
SELECT movie_title, moy.*
FROM movie mov
INNER JOIN movie_count_per_year moy
ON mov.movie_completion_year=moy.movie_completion_year
ORDER BY moy.movie_completion_year;
```

Query History | Data Output | Messages | Notifications

	movie_title character varying (50)	movie_completion_year integer	movie_count bigint
1	Seven Samurai	1954	1
2	Vertigo	1957	1
3	The Innocents	1961	1
4	Lawrence of Arabia	1962	1
5	Chinatown	1974	1
6	Annie Hall	1976	1

2) Display the movie which has made highest profit(total_earnings-budget) among all: We have used **SUBQUERY**, **MAX** and **ABS** functions and formula **total_earnings-budget**

```
--QUERY 2
--Display the movie which has made highest profit(total_earnings-budget) among all.
SELECT *
FROM movie
WHERE (total_earnings-budget)=(SELECT MAX(ABS(total_earnings-budget))
FROM movie)
```

Query History | Data Output | Messages | Notifications

	movie_id [PK] integer	movie_title character varying (50)	movie_time integer	movie_lang character varying (50)	release_date date	budget numeric (14,2)	genre_id integer	actor_id integer	director_id integer
1	937	RRR	180	Hindi	2022-03-25	5000000000.00	1013	118	218

3) Display movies which were made with high budget but failed at box office: In this query, we have used **JOIN, SUBQUERY, ORDER BY and LIMIT.**

```
100 --QUERY 3
101 --Display movies which were made with high budget but failed at box office
102
103 SELECT *
104 FROM rating rat
105 INNER JOIN movie mov
106 ON mov.movie_id=rat.movie_id
107 WHERE budget IN (SELECT budget FROM movie order by budget desc LIMIT 5)
108 AND rev_stars IN (SELECT rev_stars FROM rating order by rev_stars  LIMIT 5)
109
110
```

Query History | Data Output | Messages | Notifications

	rating_id integer	movie_id integer	reviewer_id integer	no_of_ratings integer	rev_stars numeric (3,2)	movie_id integer	movie_title character varying (50)	movie_time integer	movie_lang character varying (50)
1	331	931	9010	34000	5.10	931	Dilwale	158	Hindi

4) Display top 5 genre-id under which most of the movies are released.

```
110 --Query 4
111 --Display top 5 genre-id under which most of the movies are released.
112
113 SELECT M.genre_id,genre_title,COUNT(*) movie_count
114 FROM movie M
115 INNER JOIN genre G
116 on M.genre_id=G.genre_id
117 GROUP BY M.genre_id,G.genre_title
118 ORDER BY movie_count desc
119 LIMIT 5
120
```

Query History | Data Output | Messages | Notifications

	genre_id integer	genre_title character varying (20)	movie_count bigint
1	1013	War	11
2	1011	Romance	8
3	1001	Action	7
4	1007	Drama	6
5	1004	Biography	4

✔ Successfully run.

5) Display movie titles starting with 'B' released under different genre along with director details: In this query, we have used **CONCAT**, **JOIN** and **LIKE** functions.

```
--QUERY 5
---Display movie titles starting with 'B' released under different genre along with director details.
SELECT movie_ID,movie_title,gen.genre_id,genre_title,dir.director_id,
CONCAT_WS (' ',dir_first_name,dir_last_name) AS director_name
FROM movie mov
INNER JOIN genre gen
ON mov.genre_id=gen.genre_id
INNER JOIN director dir
ON mov.director_id=dir.director_id
WHERE movie_title LIKE 'B%'
```

Query History | Data Output | Messages | Notifications

	movie_id integer	movie_title character varying (50)	genre_id integer	genre_title character varying (20)	director_id integer	director_name text
1	927	Back to the Future	1010	Mystery	201	Alfred Hitchcock
2	906	Blade Runner	1012	Thriller	206	Ridley Scott
3	928	Braveheart	1007	Drama	208	Bryan Singer
4	923	Beyond the Sea	1009	Music	209	Roman Polanski
5	910	Boogie Nights	1009	Music	210	Woody Allen
6	948	Baahubali	1004	Biography	217	Todd Phillips

6) Using a view find the most active reviewer and display number of movies they rated: In this query we have used **VIEW**, **AGGREGATE** functions **GROUP BY** and **HAVING** clause.

```
--QUERY 6
--Using a view find the most active reviewer and display number of movies they rated.
CREATE VIEW  count_rev AS
(SELECT MAX(x.counter) AS rev_count
FROM ( SELECT reviewer_id,COUNT(*) AS counter
       FROM rating
       GROUP BY reviewer_id)x )

SELECT rat.reviewer_id,reviewer_name,COUNT(*) AS counter
FROM rating rat
INNER JOIN reviewer rev
ON rat.reviewer_id=rev.reviewer_id
GROUP BY rat.reviewer_id,reviewer_name
HAVING COUNT(*)= (SELECT rev_count FROM count_rev )
```

Query History | Data Output | Messages | Notifications

	reviewer_id integer	reviewer_name character varying (20)	counter bigint
1	9011	Vincent Cadena	6

7) Display movies which has been released later an year after its completion: **EXTRACT** function is used in this query.

```
148  --QUERY 7
149  ---Display movies which has been released later an year after its completion
150
151  SELECT movie_id,movie_title,movie_completion_year,
152  EXTRACT (year FROM release_date) AS release_year
153  FROM movie
154  WHERE EXTRACT (year FROM release_date)-movie_completion_year>1
155
156
```

Query History | Data Output | Messages | Notifications

	movie_id [PK] integer	movie_title character varying (50)	movie_completion_year integer	release_year numeric
1	905	Amadeus	1983	1985
2	907	Eyes Wide Shut	1997	1999
3	921	Slumdog Millionaire	2007	2009
4	924	Avatar	2007	2009

8) SORT MOVIES AS **Blockbuster**, **Good**, **Average** and **Below Average** according to their rating and name the new column as **'box_office_performance'**: We have used **CASE** statement in this query.

```
156  --QUERY 8
157  --SORT MOVIES AS Blockbuster,Good, Average and Below Average according to their rating and name
158  --the new column as 'box_office_performance'.
159
160  SELECT movie_title,rev_stars,
161  CASE
162  WHEN rev_stars >8 THEN 'Blockbuster'
163  WHEN rev_stars BETWEEN 6 AND 8 THEN 'Good'
164  WHEN rev_stars BETWEEN 5 AND 6 THEN 'Average'
165  ELSE 'Below Average'
166  END AS box_office_performance
167  FROM movie mov
168  INNER JOIN rating rat
169  ON mov.movie_id=rat.movie_id
170  ORDER BY rev_stars DESC;
171
```

Query History | Data Output | Messages | Notifications

	movie_title character varying (50)	rev_stars numeric (3,2)	box_office_performance text
1	KGF 2	9.60	Blockbuster
2	Trainspotting	9.40	Blockbuster
3	The Shawshank Redemption	9.40	Blockbuster
4	Spirited Away	9.20	Blockbuster
5	The Pursuit of Happyness	9.10	Blockbuster

✔ Successfully run. Total query runtime: 152 msec.

9) Display actor-director combination that repeated more than once: **PARTITION BY** and **DISTINCT** keywords are used in this query to get the desired output.

```
--QUERY 9
--Display actor-director combination that repeated more than once.

WITH act_dir_mov_count AS (SELECT actor_id,director_id,
                           COUNT(*) OVER (PARTITION BY actor_id,director_id )
AS movie_count
FROM movie )
SELECT DISTINCT CONCAT_WS (' ',actor_first_name,actor_last_name) AS actor_name,
CONCAT_WS (' ',dir_first_name,dir_last_name) AS director_name,
movie_count
FROM act_dir_mov_count mov
INNER JOIN actor act
ON mov.actor_id=act.actor_id
INNER JOIN director dir
ON mov.director_id=dir.director_id
WHERE movie_count>1
ORDER BY movie_count DESC;
```

Query History | Data Output | Messages | Notifications

	actor_name text	director_name text	movie_count bigint
1	Joaquin Phoenix	Todd Phillips	3
2	Dipika Padukon	Sanjay Bhansali	2
3	Peter OToole	David Lean	2
4	Will Smith	Gabriele Muccino	2

10) Divide movies according to their budget in 5 different buckets.

```
--Query 10
--Divide movies according to their budget in 5 different buckets.

SELECT WIDTH_BUCKET(BUDGET,
                    (SELECT MIN(budget) FROM movie), (SELECT MAX(budget) FROM movie), 5) AS BUCKET,
                    COUNT (*) AS MOVIE_COUNT
FROM movie
GROUP BY
WIDTH_BUCKET(BUDGET, (SELECT MIN(budget) FROM movie), (SELECT MAX(budget) FROM movie), 5)
ORDER BY MOVIE_COUNT DESC;
```

Query History | Data Output | Messages | Notifications

	bucket integer	movie_count bigint
1	1	48
2	6	1
3	2	1

11) Find all reviewers who have rated 8 or more stars to movies.

```
201  --QUERY 11
202  --Find all reviewers who have rated 8 or more stars to movies.
203
204  SELECT *
205  FROM reviewer rev
206  INNER JOIN rating rat
207  ON rev.reviewer_id=rat.reviewer_id
208  WHERE rat.rev_stars>=8
209
210
```

Query History | Data Output | Messages | Notifications

	reviewer_id integer	reviewer_name character varying (20)	rating_id integer	movie_id integer	reviewer_id integer	no_of_ratings integer	rev_stars numeric (3,2)
1	9001	Righty Sock	301	901	9001	263575	8.40
2	9003	Righty Sock	303	903	9003	202778	8.30
3	9004	Alec Shaw	304	904	9004	484746	8.20
4	9006	Sasha Goldshtein	306	905	9006	779489	8.60

12) Write a SQL query to compute the number of movies for each genre, **Average** movie time and round it to 2 decimal places.

```
201  --QUERY 12
202  --Write a SQL query to compute the average time and count number of movies for each genre.
203
204  SELECT genre_title, ROUND(AVG(movie_time),2) AS Average_time,COUNT(genre_title)
205  FROM movie mov
206  INNER JOIN genre gen ON
207  mov.genre_id=gen.genre_id
208  GROUP BY genre_title;
209
```

Query History | Data Output | Messages | Notifications

	genre_title character varying (20)	average_time numeric	count bigint
1	Music	136.50	2
2	Drama	130.33	6
3	Biography	140.75	4
4	Action	153.00	7
5	Thriller	117.00	1
6	Comedy	147.25	4

We have tried to cover almost all the concepts covered in the classes and after the analyses, it can be concluded that most of the movies for which we have done the analysis, are more than average rated movies and all-time favorites and they can be recommended to anyone to watch at least once.