IMDB DATASET -

ADVANCED SQL

PROJECT

* INTRODUCTION:

The IMDB (Internet Movie Database) dataset is a comprehensive collection of data related to movies, actors, directors, and various movie attributes. This dataset is structured into six distinct tables, each containing specific pieces of information crucial for analyzing movies and their performance. The tables included in this dataset are:

* + - Director Mapping
    - Role Mapping
    - Actor name
    - Ratings
    - Genre
    - Movie

The goal of this dataset is to provide an opportunity for analyzing the relationships between these different pieces of data. By leveraging SQL (Structured Query Language), we can reinforce key concepts such as joins, aggregation, filtering, and grouping. These SQL operations allow us to efficiently extract, analyze, and manipulate the data to gain meaningful insights.

By analyzing this real-world movie dataset, we can uncover various insights, such as the impact of directors on a movie’s success, the most popular genres, or trends in audience ratings over time. These insights can also have implications for movie industry professionals, like filmmakers and marketers, helping them understand audience preferences and improve future productions.

* QUERY EXPLANATION :

1.Count the total number of records in each table of the database.

The **count(\*)** function returns the total number of rows in the specified table. In this case, it will count how many records are listed in the **movie** table. The result is aliased as movie\_records to clearly indicate that this is the count of movie records. Similarly done for

2. Identify which columns in the movie table contain null values.

* This query counts the number of NULL values in specific columns of the **each** table by comparing the total number of rows in the each table (COUNT(\*)) with the number of non-NULL entries in each column (COUNT(column\_name)). The difference between these two counts gives the number of NULL values in each column.

3. Determine the total number of movies released each year, and analyze how the trend changes month-wise.

The goal of this query is to determine the total number of movies released each year and how the release trend changes on a month-by-month basis. By aggregating the data at both the **year** and **month** level, we can identify patterns in movie release dates and observe any seasonal trends or shifts in the industry over time.

* YEAR(date\_published) and MONTH(date\_published) are used to break down the release date into year and month.
* COUNT(\*) counts the total number of movies for each year-month combination.
* WHERE date\_published IS NOT NULL condition filters out movies without a published date.
* GROUP BY movie\_year, release\_month groups the data by the extracted year and month.
* ORDER BY movie\_year, release\_month ensures the results are sorted in chronological order.

4. How many movies were produced in either the USA or India in the year 2019?

* COUNT(\*) AS movie\_produced is used to count the number of movies produced in each country,
* WHERE year = 2019 filters the movies to include only those released in 2019.
* (country = 'USA' OR country = 'India') ensures that only movies produced in either the USA or India are selected.
* The GROUP BY country groups the data by country, allowing the query to count movies separately for the USA and India.

5. List the unique genres in the dataset, and count how many movies belong exclusively to one genre.

* Selecting the movie\_id from the genre table and grouping the results by movie\_id.
* Having count(genre) = 1 condition ensures that only movies that are linked to exactly one genre are considered. The subquery then returns a list of these movie IDs.
* The outer query then counts the number of movies in this list, effectively counting how many movies are associated with just one genre. The final result is displayed as unique\_genre, representing the number of movies with exactly one genre assigned to them.

6. Which genre has the highest total number of movies produced?

* **S**elects the **genre** column from the **genre** table.
* **COUNT(genre) AS highest\_total:** This counts how many movies belong to each genre. The result is aliased as highest\_total, representing the total number of movies produced in each genre from genre table.
* Groups the rows by the **genre** column, meaning the query will aggregate the movie count for each distinct genre.
* Orders the results by the **highest\_total** in descending order (DESC), meaning the genre with the most movies will appear first.
* Limits the output to just the first row, which corresponds to the genre with the highest total number of movies.

7. Calculate the average movie duration for each genre.

* Selects the genre column from the genre table as g, which contains the genre of each movie.
* The AVG() function calculates the average movie duration for each genre, using the duration column from the movie table. The result is aliased as average\_movie\_duration for clarity.
* The movie table aliased as m, the source of the movie data, including the duration of each movie.
* INNER JOIN between the movie table and the genre table. By joining on m.id = g.movie\_id ensures that the genres are matched to their respective movies based on the movie's id and the movie\_id in the genre table.
* Groups the data by the genre column, meaning the AVG(m.duration) is calculated separately for each genre.

8. Identify actors or actresses who have appeared in more than three movies with an average rating below 5.

* Selects the name of the actor or actress from the names table aliased as n and category represents an actor or actress from the role\_mapping table aliased as rm.
* Performs an INNER JOIN between the names table and the role\_mapping table using the condition n.id = rm.name\_id, meaning that the query links actors/actresses.
* Again performs INNER JOIN between the role\_mapping table and the ratings table aliased as r. The join condition rm.movie\_id = r.movie\_id links the movie IDs to get the ratings for each movie.
* Filters the results to include only movies that have an average rating below 5.
* Groups the results by actor/actress name and category so that we can aggregate data based on both the name and actor/actress.
* Having clause filters the grouped data to include only actors/actresses who have appeared in more than three distinct movies with an average rating below 5.

9. Find the minimum and maximum values for each column in the ratings table, excluding the movie\_id column.

This part of the query calculates the minimum value and maximum value for the avg\_rating, total\_votes, median\_rating column, representing the lowest and highest values from the ratings table as the source of the data.

10. Which are the top 10 movies based on their average rating?

* Selects the title column from the movie table, which gives the name of the movie. The title is then renamed as movie\_name.
* Selects the avg\_rating column from the ratings table, which contains the average rating for each movie.
* Inner join between the movie table and the ratings table. The ratings table is aliased as r, and the join condition is m.id = r.movie\_id. This ensures that the correct movie title is paired with the corresponding rating information.
* The join allows access to the avg\_rating from the ratings table, matching it to the appropriate movie.
* Sorts the results in descending order based on the avg\_rating column. By ordering the ratings in this way, the movies with the highest average ratings appear first in the results.
* Limits the results to only the top 10 rows. Therefore, the query will return the 10 movies with the highest average ratings, without including all the movies in the database.

11. Summarize the ratings table by grouping movies based on their median ratings.

* Selects the median\_rating column from the ratings table, which represents the median rating for each movie.
* COUNT() function to count the number of movies that share the same median\_rating. The result is aliased as movie\_count to represent how many movies have the same median rating.
* Groups the rows by the median\_rating column, which means the query will aggregate data and count the number of movies that have each unique median rating.
* Orders the results by the median\_rating column in ascending order. This ensures that the output lists the median ratings from lowest to highest.

12. How many movies, released in March 2017 in the USA within a specific genre, had more than 1,000 votes?

* Selects the genre column from the genre table aliased as g. It groups the results by genre, showing how many movies in each genre fit the criteria.
* Counts the number of movies using the id column from the movie table that match the specified conditions and groups them by genre. The result is aliased as movie\_count to indicate the total number of movies in each genre.
* INNER JOIN between the movie table and the genre table aliased as g. The join condition m.id = g.movie\_id ensures that each movie's genre is linked with the correct movie.
* The movie's year should be 2017 and month should be March and released in the country of USA.
* The subquery retrieves all movie\_id values where total\_votes > 1000, and the main query checks if the movie's id is in that list.
* Groups the results by genre, so that the query counts the number of movies in each genre that meet the specified criteria.
* Orders the results by movie\_count in descending order, meaning the genres with the fewest movies will appear first.

13. Find movies from each genre that begin with the word “The” and have an average rating greater than 8.

* Selects the genre column from the genre table (aliased as g) and the title column from the movie table (aliased as m). The movie title is renamed as movie\_name for clarity.
* Inner join between the genre table and the movie table (aliased as m) based on the movie**\_**id column.This join allows us to access the title of the movie from the movie table.
* Inner join between the movie table and the ratings table (aliased as r). The join is based on the id column from the movie table and the movie\_id column from the ratings table.This join ensures that we can access the **avg\_rating** from the **ratings** table for each movie.
* Only movies with an average rating greater than 8 are included in the results and whose titles start with the word "The". The % symbol is a wildcard that matches any characters that follow "The".
* Orders the results by genre, ensuring that movies are grouped and listed by their genre in alphabetical order.

14. Of the movies released between April 1, 2018, and April 1, 2019, how many received a median rating of 8?

* Count the number of rows (movies) that meet the specified conditions.
* This performs an INNER JOIN between the movie table and the ratings table (aliased as **r**). The join condition is m.id = r.movie\_id, which ensures that the query links each movie to its corresponding rating data.
* Filtering condition specifies that the date\_published (release date) of the movie must be between April 1, 2018, and April 1, 2019.The only movies released within this date range are considered in the query.
* Filters the movies further to only include those that have a median\_rating of 8 in the ratings table.

15. Do German movies receive more votes on average than Italian movies?

* Counting how many movies meet the condition of having a median\_rating of 8. The result is aliased as movie\_count, representing the total number of movies that fit the criteria.
* INNER JOIN between the movie table and the ratings table (aliased as r). The join condition is m.id = r.movie\_id, which ensures that the query links each movie to its corresponding rating data.
* Filtering condition specifies that the date\_published (release date) of the movie must be between April 1, 2018, and April 1, 2019.This ensures that only movies released within this date range are considered in the query.
* Filters the movies further to only include those that have a median\_rating of 8 in the ratings table.

16. Identify the columns in the names table that contain null values.

* This query counts the number of NULL values in specific columns of the **each** table by comparing the total number of rows in the each table (COUNT(\*)) with the number of non-NULL entries in each column (COUNT(column\_name)). The difference between these two counts gives the number of NULL values in each column from the names table.

17. Who are the top two actors whose movies have a median rating of 8 or higher?

* The name of the actor, from the names table (aliased as n). This column is aliased as Top\_actors.
* The median rating of the movies, from the ratings table (aliased as r).
* This specifies the role\_mapping table (aliased as rm) as the source. It contains the mapping between actors (via name\_id) and movies (via movie\_id).
* INNER JOIN links the role\_mapping table with the names table (aliased as n) using the name\_id. This allows us to retrieve the name of the actor from the names table.
* INNER JOIN links the role\_mapping table with the ratings table (aliased as r) based on the movie\_id, allowing us to access the median\_rating of each movie.
* The WHERE clause filters the results to include only those movies with a median\_rating of 8 or higher.
* This orders the results by median\_rating in descending order (DESC), so that actors with the highest ratings are listed first.
* The LIMIT clause restricts the results to the top 2 actors based on the highest median ratings.

18. Which are the top three production companies based on the total number of votes their movies received?

* This column comes from the movie table (aliased as m) and contains the name of the production company that produced the movie.
* This column comes from the ratings table (aliased as r) and represents the total number of votes that the movie received.
* This specifies the movie table (aliased as m) as the source of the data. The movie table contains information about the movies, including the production company.
* INNER JOIN between the movie table and the ratings table (aliased as **r**). The join condition m.id = r.movie\_id ensures that each movie is linked to its corresponding rating data.
* Orders the results by the total\_votes column in descending order. The DESC keyword ensures that movies with the highest number of votes appear first.
* Sorted so that the production companies with the highest total votes will come at the top.
* Limits the results to the top three rows. It means that only the top three production companies with the highest total votes will be returned.

19. How many directors have worked on more than three movies?

* The query joins the director\_mapping table (aliased as d) with the names table (aliased as n) on the condition that n.id matches d.name\_id. This join links each movie in the director\_mapping table to the director's name in the names table.
* The count(d.movie\_id) counts the number of movies associated with each director. d.movie\_id represents the movies directed by each director.
* The results are grouped by n.name, meaning that the query will count the movies for each director individually.  
  The having count(d.movie\_id) > 3 condition filters the results to only include directors who have directed more than 3 movies.
* The order by tot\_movie desc sorts the directors by the total number of movies they have directed, in descending order, so that directors with the most movies are listed first.

20. Calculate the average height of actors and actresses separately.

* The query calculates the average height for each category using the AVG() function on the height column from the names table (aliased as n). The result is aliased as height\_average..
* INNER JOIN between the names table and the role\_mapping table (aliased as r). The join condition n.id = r.name\_id ensures that each actor/actress is linked to their role information..
* Groups the results by the category (actor or actress). It ensures the average height is calculated separately for actors and actresses.

21. List the 10 oldest movies in the dataset along with their title, country, and director.

* The title of the movie from the movie table (aliased as m).The country where the movie was produced.The director's name, which is retrieved from the names table (aliased as n).The date the movie was published, also from the movie table (aliased as m) .
* INNER JOIN links the movie table with the director\_mapping table (aliased as d) using the movie's id.
* INNER JOIN connects the director\_mapping table with the names table (aliased as n) using the name\_id. This allows us to retrieve the director’s name from the names table.
* Order the results by the date\_published column in ascending order (ASC), ensuring that the oldest movies are listed first.
* Limit the results to the top 10 oldest movies based on their date\_published.

22. List the top 5 movies with the highest total votes, along with their genres.

* The title of the movie from the movie table (aliased as m). The total number of votes for the movie from the ratings table (aliased as r).The genre of the movie from the genre table (aliased as g).
* INNER JOIN between the movie table and the ratings table (aliased as r) based on the movie\_id.
* INNER JOIN between the genre table (aliased as g) and the movie table. The join is done based on the movie\_id, so it links the movie to its genre.
* Orders the results by the total\_votes column in descending order (DESC). This ensures that movies with the highest total votes come first.
* Limits the output to the top 5 rows, i.e., the 5 movies with the highest total votes.

23. Identify the movie with the longest duration, along with its genre and production company.

* The movie table (aliased as m) as the source of movie data (title, production company, duration).
* This INNER JOIN links the movie table with the genre table (aliased as g) based on the movie\_id. This join allows us to access the genre for each movie.
* This orders the results by m.duration in descending order (DESC), so the movie with the longest duration appears first.
* This limits the result to only the top movie (the one with the longest duration).

24. Determine the total number of votes for each movie released in 2018.

* The title of the movie from the movie table (aliased as m).
* The SUM() function is used to add up all the votes for each movie (from the ratings table aliased as r). This sum is aliased as total\_votes.
* INNER JOIN between the movie table and the ratings table (aliased as r) using the movie\_id. This join allows us to access the total\_votes for each movie.
* The WHERE clause filters the results to include only movies released in 2018. It ensures that the query only looks at movies from that specific year.
* The GROUP BY clause groups the results by m.title, meaning the total votes will be calculated for each individual movie.
* This order by clause orders the results by total\_votes in descending order (DESC), so that the movies with the highest number of votes appear first.

25. What is the most common language in which movies were produced?

* The title of the movie from the movie table (aliased as m).The total number of votes for each movie.
* The SUM() function is used to add up all the votes for each movie (from the ratings table aliased as r). This sum is aliased as total\_votes.
* This specifies the movie table (aliased as m) as the main source of movie data, including the movie title and the release year.
* INNER JOIN between the movie table and the ratings table (aliased as r) using the movie\_id. This join allows us to access the total\_votes for each movie.
* WHERE clause filters the results to include only movies released in 2018. It ensures that the query only looks at movies from that specific year.
* The GROUP BY clause groups the results by m.title, meaning the total votes will be calculated for each individual movie.
* This orders the results by total\_votes in descending order (DESC),so that the movies with the highest number of votes appear first.
* KEY INSIGHTS:
* Among the six different table from imdb database role contains large number of records(rows)
* Only two movies received the average rating out of 10 which are Kirket and Love in Kilnerry.
* The median rating about 7 has highest number of movies.
* Drama is the genre where more than 1000 number of people voted.
* The title of drama genre is mostly start with the word “The”.
* 391 movies are released between April 1,2018 and April 1, 2019.
* German movies receive more average votes than Italian movies.
* Marvel Studios is the top production company.
* Nine directors are worked more than three movies.
* Avengers:Infinity war movie receives the maximum votes from people.
* La flor is the drama genre movie which has the longest duration.
* English is the most common language where most of the movie produced.
* CONCLUSION:

This analysis of the IMDb dataset provides valuable insights into various aspects of the movie industry, highlighting key trends, actor and director performances, and production patterns. By examining movie data such as release trends, total votes, genres, and ratings. The movies with the highest total votes are likely those that have captured the largest audience attention, indicating their widespread appeal or significant marketing campaigns. The analysis of movie releases by year and month reveals patterns in the production cycle. In conclusion, this dataset provides a comprehensive view of the film industry, highlighting patterns in audience preferences, movie success factors, and the influence of key players like actors, directors, and production companies. These insights are essential for making informed decisions in movie production, distribution, and marketing strategies, as well as understanding broader trends in the global entertainment market.