# Slide 1: Title Slide Content:

* Project Title: IMDB DATASET
* LOGESH.R
* Date Submitted: 14.10.2024

# Slide 2: Aim of the Project Content:

The aim of this project is to **reinforce and apply advanced SQL skills** by analyzing a real-world movie dataset (the IMDb dataset) through various SQL queries. Specifically, the project focuses on:

1. **Mastering Core SQL Concepts**: The project provides an opportunity to practice key SQL techniques, such as:
   * **Joins**: Combining data from multiple tables to extract meaningful insights.
   * **Aggregation**: Summarizing data through functions like COUNT(), AVG(), SUM(), etc.
   * **Filtering and Grouping**: Using WHERE, HAVING, and GROUP BY to narrow down and categorize data.
   * **Subqueries and Window Functions**: Applying advanced query techniques for more complex analyses.
2. **Data Analysis and Insights Extraction**: By performing queries on the IMDb dataset, the project encourages extracting valuable insights, such as trends in movie ratings, popular genres, actors' performance, box office performance, etc.

# Slide 3: Business Problem or Problem Statement Content:

The movie industry generates large volumes of data every year, with millions of movies being produced, rated, and reviewed globally. Movie studios, production companies, and streaming services need to make informed decisions based on various factors such as audience preferences, genre trends, director performance, and more.

# Slide 4: Project Description Content:

This project involves the analysis of a **simplified IMDb movie dataset** using **advanced SQL techniques**. The dataset provides comprehensive information about movies, genres, actors, directors, ratings, and other key attributes related to films. The primary goal of this project is to explore, analyze, and derive meaningful insights from the dataset to address real-world business problems in the entertainment industry, specifically in the movie production, distribution, and marketing sectors.

# Slide 5: Functionalities Content:

The functionalities of this project focus on providing a comprehensive analysis of the IMDb dataset using SQL queries to extract valuable insights. The analysis will be performed across various dimensions, such as movie ratings, genres, actors, directors, production companies, and regional preferences. Below are the key functionalities that the project will deliver:

**1. Data Exploration and Integrity Check**

**2. Trend Analysis of Movie Releases**

**3.Genre and Movie Popularity Analysis**

**Slide6:Description:**

In this project, we implement various modules using advanced SQL techniques to analyze and extract insights from the IMDb dataset. Each query is designed to handle specific data-related tasks such as counting records, filtering data, performing aggregations, and identifying trends within the movie industry. The primary goal is to reinforce key SQL concepts and apply them to real-world movie data, offering valuable insights into trends, ratings, production details, and actor involvement.

Let's consider the implementation of a Query management module for each query:

1. This query is used to count the total number of records in each table of the IMDb dataset. It selects the table name and the count of records for each table (Movie, Genre, Director Mapping, Role Mapping, Names, and Ratings). The UNION operator is used to combine the results of multiple SELECT statements into a single result set, where each row represents the table name and its respective record count. This query helps in quickly getting an overview of the data distribution across the different tables in the database.
2. This SQL query is designed to count the number of NULL values in each specified column of the "movie" table. It does this by performing separate SELECT statements for each column (e.g., id, title, year, etc.) where it checks for NULL values using the WHERE clause. Each SELECT statement returns the column name and the count of rows where that column has a NULL value. The UNION ALL operator is used to combine the results of all the SELECT statements into one result set, so that each row shows the column name and its corresponding NULL value count. This query helps to identify which columns have missing data in the table.
3. The three queries analyze movie release trends by year and month. The first query counts the total number of movies released each year, while the second counts movies released each month. The third query combines both year and month to show the number of movies released in each specific month of each year. All queries use GROUP BY to aggregate the data and ORDER BY to sort the results, providing insights into movie release patterns over time.
4. This query counts the number of movies produced in either the USA or India in the year 2019. It filters the data by specifying that the country should be either 'USA' or 'India' and the year should be 2019. The GROUP BY clause groups the results by country, and COUNT(\*) counts the total number of movies for each country. The result will show the number of movies produced in the USA and India in 2019.
5. This query lists the unique genres from the genre table and counts how many movies belong to each genre. It joins the genre table with the movie table based on the movie\_id field to ensure that each genre is associated with its respective movie. The GROUP BY clause groups the results by genre, and COUNT(m.id) counts the number of movies for each genre. The results are then ordered by movie\_count in descending order, showing the most popular genres first.
6. This query identifies the genre with the highest number of movies. It joins the genre table with the movie table using the movie\_id field. The GROUP BY clause groups the results by genre, and COUNT(m.id) counts the number of movies for each genre. The results are ordered by movie\_count in descending order, and the LIMIT 1 ensures that only the genre with the most movies is returned. The result will show the genre with the highest movie count.
7. This query calculates the average movie duration for each genre. It joins the genre table with the movie table using the movie\_id field, ensuring each genre is associated with its corresponding movies. The AVG(m.duration) function computes the average duration of movies for each genre. The GROUP BY clause groups the results by genre, and the results are ordered by average\_duration in descending order to show the genre with the longest average movie duration at the top.
8. This query identifies actors or actresses who have appeared in more than three movies with an average rating below 5. It joins three tables: role\_mapping (for movie-actor mappings), ratings (for the movie ratings), and names (for actor details). The query filters for movies with an avg\_rating below 5 using the WHERE clause. The GROUP BY clause groups the results by actor (using rm.name\_id and n.name). The HAVING clause ensures only those actors who have appeared in more than three such low-rated movies are included. The query also calculates the total number of movies (COUNT(rm.movie\_id)) and the average rating (AVG(rt.avg\_rating)) for each actor.
9. This query calculates the minimum and maximum values for the avg\_rating, total\_votes, and median\_rating columns in the ratings table. The MIN() and MAX() aggregate functions are used to find the lowest and highest values for each of these columns, respectively. The result will show the overall minimum and maximum ratings, vote counts, and median ratings across all movies in the dataset, giving an overview of the rating distribution and voting patterns.
10. This query retrieves the top 10 movies based on their average rating. It joins the movie table with the ratings table using the movie\_id field to associate each movie with its average rating. The ORDER BY rt.avg\_rating DESC sorts the movies in descending order of their average rating, ensuring that the highest-rated movies come first. The LIMIT 10 restricts the result to the top 10 movies with the highest ratings. The result will show the movie titles along with their average ratings.
11. This query groups movies by their median\_rating from the ratings table and counts how many movies have each median rating. It uses the COUNT(rt.movie\_id) function to count the number of movies for each median\_rating. The GROUP BY clause groups the results by the median\_rating, and the ORDER BY clause sorts the results in ascending order of the median\_rating. The result will show each unique median rating and the number of movies that have that rating.
12. This query retrieves the count of movies for each genre that meet specific criteria:

The movie was released in March 2017 (m.date\_published LIKE '2017-03-%').

The movie was produced in the USA (m.country = 'USA').

The movie received more than 1,000 votes (r.total\_votes > 1000).

It joins the genre, movie, and ratings tables to get the relevant data. The GROUP BY g.genre groups the results by genre, and COUNT(\*) counts the number of movies for each genre that meet these conditions. The result will show the genres and the corresponding number of movies released in March 2017 in the USA with over 1,000 votes.

1. This query retrieves a list of movies that:

Have a title starting with "The" (m.title LIKE 'The %').

Have an average rating greater than 8 (r.avg\_rating > 8).

It joins the genre, movie, and ratings tables to gather the genre, movie title, and average rating information. The results are ordered first by genre and then by avg\_rating in descending order, ensuring that within each genre, the highest-rated movies are listed at the top. The result will show the movie titles, their genres, and their average ratings, filtered by the specified conditions.

1. This query counts the number of movies released between April 1, 2018, and April 1, 2019, that have a median rating of 8. It joins the movie table with the ratings table based on the movie\_id to associate each movie with its median rating. The WHERE clause applies two filters: one for the date range (m.date\_published BETWEEN '2018-04-01' AND '2019-04-01') and another for movies that have a median rating of 8 (r.median\_rating = 8). The COUNT(\*) function counts the total number of such movies, and the result will show the number of movies that meet both conditions.
2. This query calculates the average number of votes for movies produced in Germany and Italy. It joins the movie table with the ratings table based on the movie\_id to get the voting data for each movie. The WHERE clause filters for movies produced in either Germany or Italy (m.country IN ('Germany', 'Italy')). The GROUP BY m.country groups the results by country, and the AVG(r.total\_votes) function calculates the average number of votes for each country's movies. The result will show the average number of votes for German and Italian movies, allowing you to compare them and determine if German movies receive more votes on average than Italian movies.
3. This query counts the number of NULL values for specific columns in the names table. It uses COUNT combined with CASE WHEN statements to conditionally count rows where each specified column (id, name, height, date\_of\_birth, known\_for\_movies) contains NULL values.

COUNT(CASE WHEN id IS NULL THEN 1 END) AS id\_nulls counts the number of NULL values in the id column.

Similarly, the other COUNT statements count NULL values in the respective columns: name, height, date\_of\_birth, and known\_for\_movies.

The result will show how many NULL values are present in each of these columns in the names table.

1. This query retrieves the names of actors (from the names table) who have worked on movies with a median rating of 8 or higher. It joins the director\_mapping table (which maps directors to movies) with the names table to get actor details and with the ratings table to access the movie's median rating. The WHERE clause filters for movies with a median\_rating of 8 or higher. The ORDER BY r.median\_rating DESC sorts the results in descending order based on the median rating, and LIMIT 2 restricts the output to the top 2 actors who have worked on the highest-rated movies. The result will show the names of the top two actors along with the corresponding median ratings.
2. This query retrieves the top 3 production companies based on the total number of votes their movies have received. It joins the movie table with the ratings table using the movie\_id field to associate each movie with its total votes. The SUM(r.total\_votes) function calculates the total number of votes for each production company. The GROUP BY m.production\_company groups the results by production company, and ORDER BY total\_votes DESC sorts the companies in descending order of total votes. The LIMIT 3 ensures that only the top 3 production companies with the highest total votes are returned.
3. This query identifies directors who have worked on more than three movies. It joins the director\_mapping table (which maps directors to movies) with the names table (which contains director information). The COUNT(d.movie\_id) counts the number of movies each director has worked on. The GROUP BY n.name groups the results by director name. The HAVING clause filters the results to only include directors who have worked on more than three movies. The result will show the names of directors along with the total number of movies they have directed, but only for those who have directed more than three movies.
4. This query calculates the average height of individuals in each role category (e.g., actor, director, producer, etc.). It joins the role\_mapping table (which links people to their roles in movies) with the names table (which contains information about those individuals, including their height). The AVG(n.height) function calculates the average height for each role category. The GROUP BY rm.category groups the results by role category, and the result will show the average height for each category, such as actors, directors, and producers.
5. This query retrieves information about the first 10 movies (ordered by their release date in ascending order) along with their directors. It joins three tables:

The movie table to get the movie details (title, country, date\_published).

The director\_mapping table to link movies to their respective directors.

The names table to get the director's name.

The ORDER BY m.date\_published ASC ensures that the results are sorted by the movie's release date, starting from the earliest. The LIMIT 10 restricts the output to the first 10 movies in the dataset based on the release date. The result will show the movie title, country, director name, and the movie's release date.

1. This query retrieves the top 5 movies based on the highest number of total votes, along with their genres and total votes. It joins three tables:

The movie table to get the movie details (title).

The ratings table to get the total votes for each movie.

The genre table to get the genre associated with each movie.

The query groups the results by m.id (movie ID) and r.total\_votes (total votes), ensuring that each movie is counted once per genre. The MIN(g.genre) is used to fetch the genre for each movie (assuming a movie may have multiple genres). The ORDER BY r.total\_votes DESC sorts the movies by the total votes in descending order, and LIMIT 5 ensures that only the top 5 movies with the most votes are returned. The result will show the movie title, total votes, and one of the movie's genres.

1. This query retrieves the movie with the longest duration from the movie and genre tables. It joins the two tables based on the movie\_id to associate each movie with its genre. The ORDER BY m.duration DESC sorts the movies by their duration in descending order, ensuring that the longest movie appears at the top. The LIMIT 1 ensures that only the top movie (the one with the longest duration) is returned. The result will display the movie's title, duration, genre, and production company.
2. This query calculates the total number of votes for each movie released in 2018. It joins the movie table with the ratings table using the movie\_id to associate each movie with its total votes. The WHERE m.year = 2018 filter restricts the results to movies released in 2018. The GROUP BY m.title groups the results by movie title, and SUM(r.total\_votes) calculates the total number of votes for each movie. The result will show the movie title, the sum of total votes, and the year (which will be 2018 for all rows).
3. This query identifies the most common language in which movies were produced, based on the count of movies for each language. It selects the languages column from the movie table and uses COUNT(\*) to calculate how many movies were produced in each language. The GROUP BY languages groups the results by language, and the ORDER BY language\_count DESC sorts the results in descending order of the movie count. The LIMIT 1 ensures that only the most common language is returned. The result will show the most frequently used language along with the number of movies produced in that language.

**7 . Results and Outcomes:**

In this project, a thorough analysis of the IMDb dataset was conducted using advanced SQL techniques. Key insights were gained, such as the total number of movies released each year, with trends examined by month. We identified the most common genres, directors, and actors based on various performance metrics like ratings and votes. Notably, German movies were found to receive more votes on average than Italian movies. We also determined the top-rated movies and the most prolific production companies based on total votes. The project further explored movie durations, actors' performance in low-rated films, and the most common movie language. Ultimately, the results highlighted trends in movie production, ratings, and audience engagement, enriching our understanding of the global film industry.

**8 . Conclusion:**

This project successfully applied advanced SQL techniques to extract meaningful insights from the IMDb dataset, shedding light on various aspects of the global film industry. By analyzing movie release trends, genres, ratings, and production companies, we were able to identify patterns in audience engagement and the performance of movies across different regions and time periods. Key findings included the dominance of certain genres, the correlation between director and movie ratings, and the impact of language and country on movie popularity. Additionally, the project highlighted the significant role of directors and actors in high-rated films, providing a deeper understanding of what drives movie success. The results contribute valuable insights for stakeholders in the film industry, offering data-driven recommendations for content production and audience targeting.