# □ Movie Review & Recommendation Engine using SQL

## ✅Project Overview

This project focuses on building a SQL-based Movie Recommendation and Review system using PostgreSQL. It simulates a mini version of platforms like IMDb or Netflix by storing and analyzing user ratings, reviews, watch behavior, and movie metadata. This project highlights key AI concepts like sentiment analysis (via sentiment scores) and recommendation logic using SQL.

---

## □ Objectives

- Design a normalized database schema for a movie platform.
- Collect and insert sample data for users, movies, genres, reviews, and ratings.
- Use SQL queries and views to derive user preferences and recommendations.
- Use SQL Window Functions and Aggregations to identify top movies.
- Enable exportable reports from views and queries.

---

## □ Schema Design

### □ Tables:

- **Users** – Stores user profiles
  (user_id, name, age, gender, location)
- **Movies** – Stores movie information
  (movie_id, title, release_year, duration, language)
- **Genres** – List of genres
  (genre_id, genre_name)
- **Movie_Genres** – Many-to-many link between movies and genres
  (movie_id, genre_id)
- **Ratings** – Stores ratings by users
  (rating_id, user_id, movie_id, rating)
- **Reviews** – Text-based reviews and sentiment score
  (review_id, user_id, movie_id, review_text, sentiment_score)
- **Watch_History** – Tracks viewing activity
  (history_id, user_id, movie_id, watch_date, watch_duration, completed)

---

## □ Sample Data

- **Movies** include "Inception", "3 Idiots", "Interstellar", "Dangal", etc.
- **Users** from different cities and age groups
- **Genres**: Action, Comedy, Drama, Sci-Fi, Romance, Thriller, etc.
- **Ratings** range from 1.0 to 5.
- **Reviews** include short comments with a **sentiment_score** (0.0 to 1.0)
- **Watch History** logs partial or complete watch sessions

---

## □ Key SQL Features Used

### 1. Aggregations

To calculate average ratings:

SELECT movie_id, AVG(rating) AS avg_ratingFROM RatingsGROUP BY movie_id;

### 2. Window Functions

To rank top movies globally:sql

CopyEdit
SELECT movie_id, AVG(rating) AS avg_rating,
RANK() OVER (ORDER BY AVG(rating) DESC) AS rankFROM RatingsGROUP BY movie_id;

### 3. Views

View for highly rated movies:

CREATE VIEW Top_Rated_Movies ASSELECT m.title, AVG(r.rating) AS avg_ratingFROM Movies mJOIN Ratings r ON m.movie_id = r.movie_idGROUP BY m.titleHAVING AVG(r.rating) > 4.0;

### 4. Recommendation View (User-Based)

Recommend movies a user hasn't watched, but similar users liked:

CREATE VIEW Recommended_For_User1 ASSELECT m.titleFROM Movies mWHERE movie_id NOT IN (
SELECT movie_id FROM Watch_History WHERE user_id = 1
)AND movie_id IN (
SELECT movie_id FROM Ratings
WHERE rating >= 4.0
GROUP BY movie_id
HAVING COUNT(user_id) > 1
);

---

## □ AI Element: Sentiment Analysis

- Each review is assigned a **sentiment_score** (0.0 to 1.0)
- You can analyze emotional impact using:

SELECT movie_id, AVG(sentiment_score) AS avg_sentimentFROM ReviewsGROUP BY movie_id;

---

## □ Final Outputs / Deliverables

- SQL Script with DDL + DML (table creation + data)
- Views for top-rated, recently watched, and recommended movies

- Ranking reports using window functions
- Exportable CSV results (via SQL client)
- Optional: Dashboard integration using tools like **Metabase** or **Power BI** (if needed)

---

## □ Conclusion

This SQL-based Movie Recommendation Engine showcases the power of relational databases in simulating AI-driven applications. With structured queries, user behavior tracking, and sentiment insights, this project can be a foundational step toward building full-fledged AI-based recommendation systems using SQL and Python in combination.