SQL : IMDB Movie Database

Your Mission

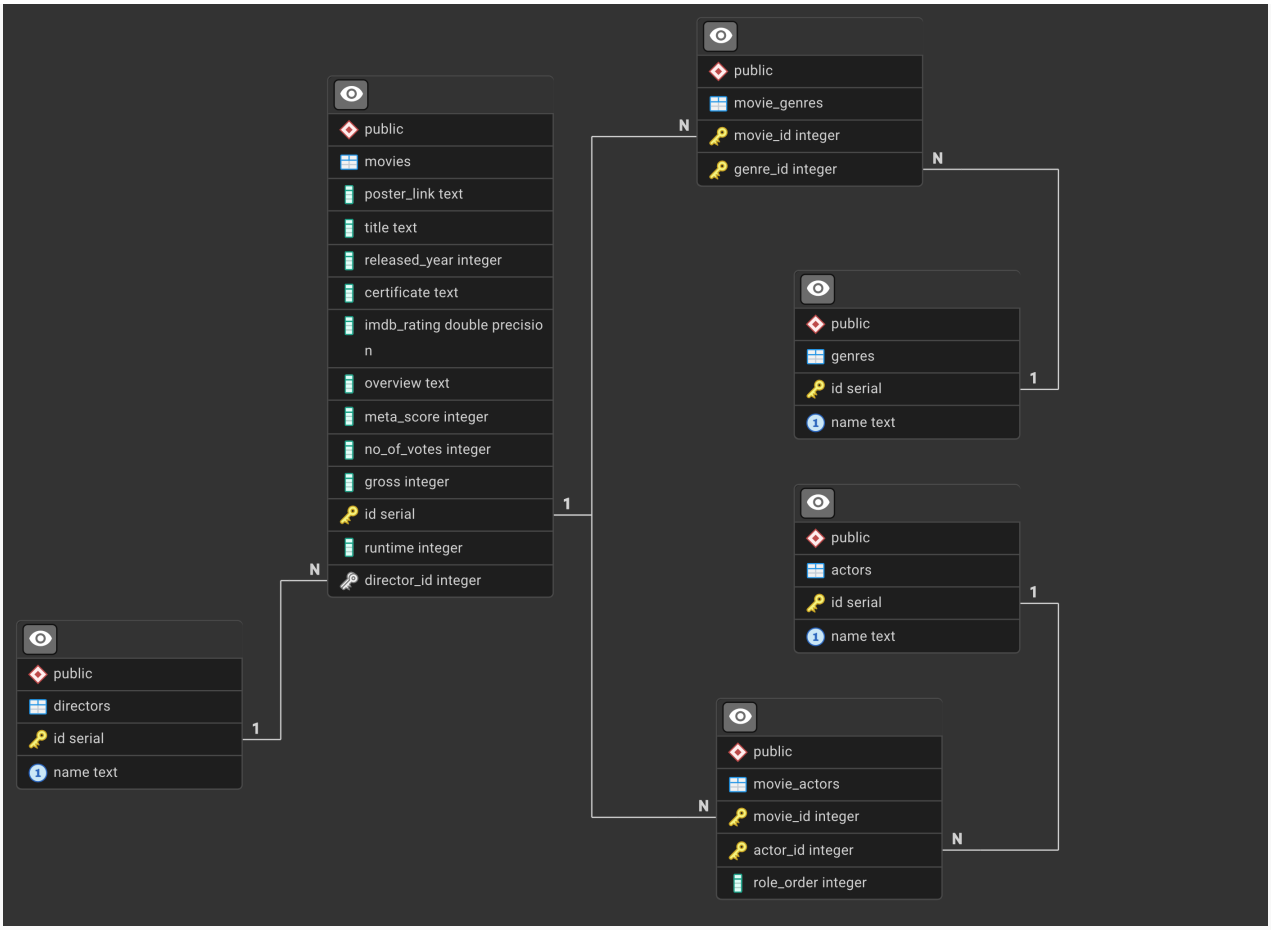
You've just been hired as a Junior Data Analyst at **CineStream**, a new streaming service that wants to make data-driven decisions about which movies to license for their platform.

Your manager has given you access to a database containing information about the top-rated movies from IMDB. She needs you to explore this database and answer several business questions to help the content acquisition team make informed decisions about: - Which genres are most popular among highly-rated films - Which directors and actors appear most frequently in successful movies - What patterns exist in movie ratings and viewer engagement

This is your first day working with SQL (Structured Query Language), the standard language for interacting with databases. Don't worry - we'll guide you through each step!

The IMDB Database Tables

The moviesdb database contains **6 tables** that work together to store movie information:



Main Tables (Storing Core Data)

1. `movies` - The heart of the database. (1000 movies)
    - Contains: Movie titles, release years, ratings, runtime, revenue (gross), and more
    - Each row = One movie
    - Key columns:
        - `id`: Unique identifier for each movie
        - `title`: Movie name
        - `released_year`: When it came out
        - `imdb_rating`: Score from 0-10
        - `no_of_votes`: How many people rated it
        - `runtime`: Length in minutes
        - `director_id`: Links to the directors table
2. `actors` - All actors in the database. (2709 actors)
    - Contains: Actor names
    - Each row = One actor
    - Key columns:
        - `id`: Unique identifier

        - `name`: Actor's full name
3. `directors` - All directors in the database. (548 directors)
    - Contains: Director names
    - Each row = One director
    - Key columns:
        - `id`: Unique identifier
        - `name`: Director's full name
4. `genres` - All possible movie genres. (21 genres)
    - Contains: Genre categories (Action, Drama, Comedy, etc.)
    - Each row = One genre
    - Key columns:
        - `id`: Unique identifier
        - `name`: Genre name

🔗 **Relationship Tables (Connecting Data)**

1. `movie_actors` - Links movies with their actors. (3996 relations)
    - A movie has multiple actors, and actors appear in multiple movies
    - Key columns:
        - `movie_id`: Which movie
        - `actor_id`: Which actor
        - `role_order`: Billing order (1 = lead role, 2 = supporting, etc.)
2. `movie_genres` - Links movies with their genres. (2541 relations)
    - A movie can have multiple genres (e.g., "Action" AND "Comedy"), a genre can be applied to multiple movies
    - Key columns:
        - `movie_id`: Which movie
        - `genre_id`: Which genre

Understanding Table Relationships

One-to-Many Relationship (1:N)

- **movies** ↔ **directors**: Each movie has ONE director, but a director can direct MANY movies
    - The `director_id` in the movies table points to an `id` in the directors table

Many-to-Many Relationships (N:N)

- **movies** ↔ **actors**: A movie has MANY actors, and actors appear in MANY movies

    - Connected through the `movie_actors` table
- **movies** ↔ **genres**: A movie can have MANY genres, and each genre applies to MANY movies
    - Connected through the `movie_genres` table

Load the database

Download the database file (< 1 Mb) from this link

Launch sqlite with the following command: `bash sqlite3 moviesdb.sqlite`

or using a GUI tool like DB Browser for SQLite

Exercises - deliverable

In the following exercises, you will be asked to write SQL queries to answer specific questions about the database. Write your queries in the sqlite session.

Once you are satisfied with the result and copy the SQL query to a file called `queries.sql`. Do not copy the result, just the sql statement.

Submit your `queries.sql` file at the end of the TP.

SQL Query Exercises

Section A: Your First Queries - Simple SELECT Statements

Let's start by exploring single tables. These queries will help you get comfortable with basic SQL syntax.

A.1 - See All Genres

Task: Display all available genres in the database.

Expected: A list of all genre names with their IDs

A.2 - Find Specific Movie Information

Task: Your manager wants to know the `title`, `release year`, and `IMDB rating` for all movies. Display only these three columns.

Hint: Instead of using *, list the specific column names separated by commas

---

A.3 - Browse Recent Movies

Task: Find all movies released after 2015. Show `title`, `release year`, and `IMDB rating`.

Hint: use the WHERE clause to filter your results

A.4 - Find Highly-Rated Movies

Task: The content team wants movies with an IMDB rating of 8.5 or higher. Show `title`, `imdb_rating`, and `no_of_votes`.

A.5 - Search for a Specific Director

Task: Check if "Christopher Nolan" is in our directors database.

Hint: Use `=` for exact matches with text (put text in single quotes)

Expected: A list of all genre names with their IDs

Section B: Sorting and Limiting Results

Now let's control how our results are displayed.

B.1 - Top Rated Movies

Task: Show the top 10 highest-rated movies. Display `title` and `imdb_rating`, sorted by rating (highest first).

Hint: DESC = descending (high to low), ASC = ascending (low to high)

Expected: A list of the top 10 highest-rated movies with their titles and ratings

---

B.2 - Most Popular Movies

Task: Find the 5 movies with the most votes. Show `title` and `no_of_votes`, ordered by votes.

B.3 - Longest Movies

Task: What are the 10 longest movies? Display `title` and `runtime` (in minutes), ordered by `runtime`.

B.4 - Recent Blockbusters

Task: Find movies from 2010 or later with a rating above 8.0. Show `title`, `released_year`, and `imdb_rating`. Sort by year (newest first).

Hint: Use AND to combine multiple conditions

B.5 - Finding Movies in a Rating Range

Task: Find all movies with ratings between 7.5 and 8.0 (inclusive). Display title and imdb_rating, sorted by rating.

Alternative: You can use `BETWEEN` instead of `>=` and `<=`

Section C: Using Simple Functions

Let's learn some basic SQL functions to analyze and clean our data.

C.1 - Counting Movies

Task: How many movies are in our database? return the total as `total_movies`

Hint: COUNT(*) counts all rows. AS gives a nickname to the result column.

C.2 - Counting High-Budget Films

Task: How many movies have box office earnings (gross) recorded?

Hint: COUNT(column_name) only counts non-NULL values

C.3 - Handling Missing Data

Task: Display movie titles and their meta*score, but show 'No Score' for movies without a meta*score.

Hint: COALESCE replaces NULL values with something else. check out the sqlite documentation for more info

---

C.4 - Cleaning Up Certificate Data

Task: Show movie titles and certificates. Replace NULL certificates with 'Not Rated'.

C.5 - Finding Movies with Missing Information

Task: Count how many movies are missing their gross (box office) information.

Hint: COUNT(*) minus COUNT(gross) gives us the count of NULL values

---

Section D: Your First Joins - Connecting Tables

Now for the exciting part! Let's connect tables to answer more complex questions.

D.1 - Movies with Director Names

Task: Show movie titles with their director names (not just director_id). Display the first 10 results.

Hint: JOIN connects two tables (movies and directors) using matching values (director_id = id)

D.2 - Finding a Director's Movies

Task: Find all movies directed by "Christopher Nolan". Show title and released_year.

D.3 - Movies and Their Genres

Task: Show movie titles with their genre names. Display the first 20 results.

Hint: You can chain multiple JOINs to connect several tables (movies, movie_genres, genres)

D.4 - Finding Action Movies

Task: Find all movies in the "Action" genre. Display `title` and `imdb_rating`, sorted by rating.

D.5 - Movies with Their Lead Actors

Task: Show movies with their lead actors (role_order = 1). Display `movie title` and `actor name`.

Section E: Combining Everything - Advanced Queries

Let's combine all the concepts you've learned!

E.1 - High-Rated Movies with Directors

Task: Find movies with rating >= 8.5 and show them with director names. Sort by rating.

E.2 - Recent Movies in Drama Genre

Task: Find Drama movies from 2010 onwards. Show `title`, `year`, and `rating`.

E.3 - Director's Best Movies

Task: Find Steven Spielberg's top 5 highest-rated movies. Display `title`, `rating`, and `year`.

E.4 - Movies with Complete Information

Task: Find movies that have both gross earnings AND meta_score recorded. Show `title` and both values, sorted by gross earnings.

Hint: IS NOT NULL checks for non-missing values

E.5 - Star-Studded Movies

Task: Find movies where "Tom Hanks" is the lead actor (role_order = 1). Show `movie title`, `release year`, and `rating`.

Section F: GROUP BY - Aggregating Data

Time to learn one of SQL's most powerful features: GROUP BY. This allows us to create summaries and calculate statistics for groups of data.

Understanding GROUP BY

What does GROUP BY do? GROUP BY groups rows that have the same values in specified columns and allows you to perform calculations on each group.

Think of it like this: - Without GROUP BY: "Count all movies in the database" - With GROUP BY: "Count movies FOR EACH director" or "Count movies FOR EACH genre"

The Rule: When using GROUP BY, you can only SELECT: 1. The columns you're grouping by 2. Aggregate functions (COUNT, AVG, MAX, MIN, SUM) on other columns

Example Structure: `sql SELECT director_id, COUNT(*) as movie_count FROM movies GROUP BY director_id;` This counts how many movies each director has made.

F.1 - Movies Per Director

Task: Count how many movies each director has directed. Show director_id and the count.

Hint: GROUP BY creates one row per unique director_id

F.2 - Movies Per Year

Task: How many movies were released each year? Show the year and count, sorted by year.

F.3 - Average Rating Per Director

Task: Calculate the average IMDB rating for each director. Show director name and average rating, sorted by average rating (highest first).

Hint: AVG() calculates the average of a numeric column

F.4 - Genre Popularity

Task: Count how many movies belong to each genre. Show genre name and count.

F.5 - Actor Appearances

Task: Count how many movies each actor appears in. Show actor name and count for actors with 5 or more movies.

Hint: HAVING filters groups (use it after GROUP BY, while WHERE filters rows before grouping)

F.6 - Directors' Best and Worst

Task: For directors with at least 3 movies, show their name, movie count, highest rating, and lowest rating.

Hint: You can use multiple aggregate functions in one query

F.7 - Box Office by Genre

Task: Calculate the total and average box office (gross) for each genre. Only include genres with at least 10 movies.

Hint: SUM() adds up all values in a group

🎉 Congratulations!

You've completed your first SQL worksheet! You've learned how to: - Write basic SELECT queries - Filter results with WHERE - Sort with ORDER BY and limit results - Use functions like COUNT and COALESCE - Join tables to answer complex questions - Use GROUP BY to aggregate and summarize data - Filter groups with HAVING

Final Challenge: Write a query to find which year had the highest average IMDB rating for movies (considering only years with at least 5 movies).

Quick Reference

- **SELECT**: Choose columns to display
- **FROM**: Specify the table
- **WHERE**: Filter rows (before grouping)
- **JOIN...ON**: Connect tables

- **GROUP BY**: Group rows for aggregation
- **HAVING**: Filter groups (after grouping)
- **ORDER BY**: Sort results
- **LIMIT**: Restrict number of rows

Aggregate Functions: - **COUNT()**: Count rows - **AVG()**: Calculate average - **SUM()**: Add up values - **MAX()**: Find maximum value - **MIN()**: Find minimum value - **COALESCE()**: Replace NULL values - **AND/OR**: Combine conditions - **IS NULL / IS NOT NULL**: Check for missing values