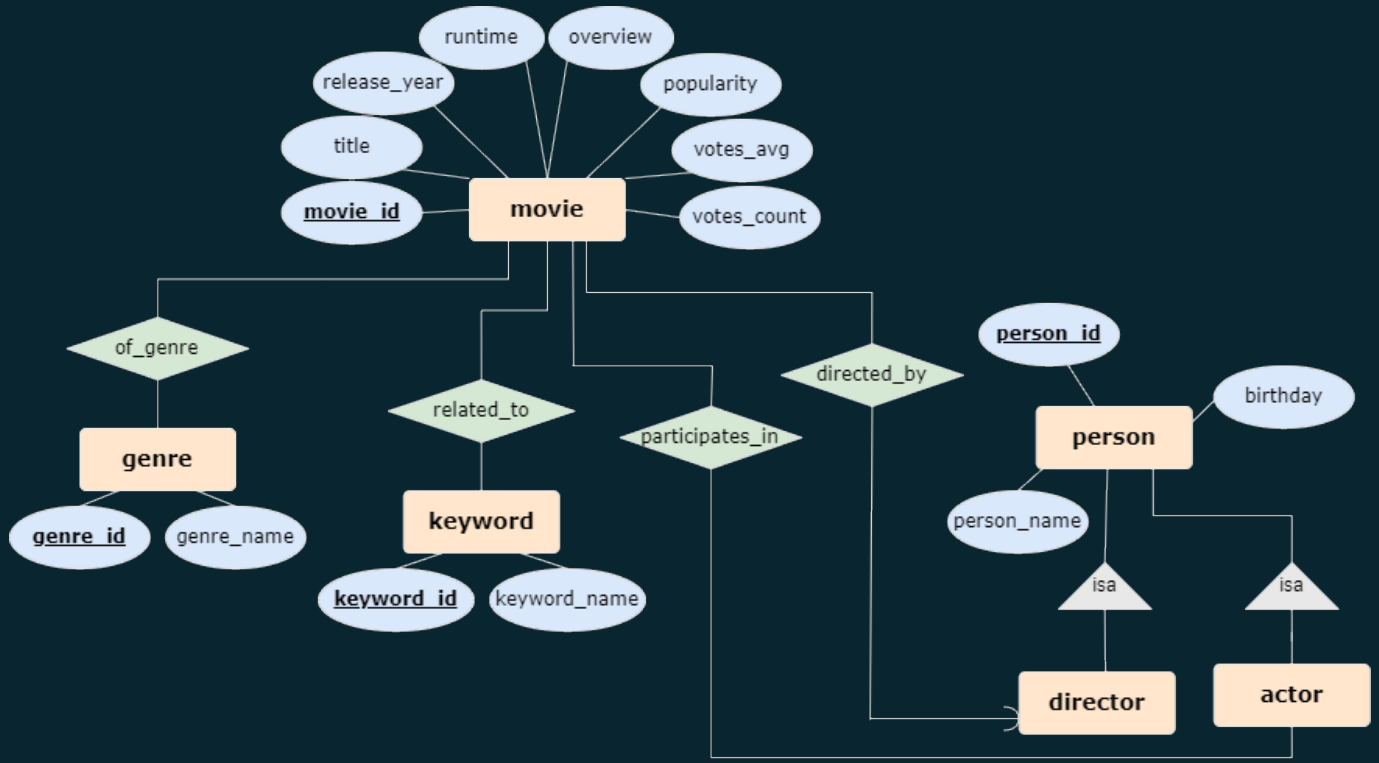
SYSTEM DOCS

LOLMOVIES.COM

The Best Comedy movies.

DBMS Final Assignment | Michal Berkheim, Michal Lahav, Reshit Carmel | 2025

ER Diagram


runtime
overview
release_year
popularity
title
votes_avg
movie_id
movie
votes_count
of_genre
related_to
participates_in
directed_by
person_id
birthday
genre
person
genre_id
genre_name
keyword
person_name
keyword_id
keyword_name
isa
isa
director
actor

Comedy Movies Schema: Tables

The schema is designed to support the storage of movie-related data, including relationships between movies, genres, keywords, and people (actors and directors).

- Movie(Movie_id, Title, Director_id, Release_year, Runtime, Overview, Popularity, Vote_average, Vote_count)
- Person(Person_id, Person_name, Birthday)
- Actor(Actor_id)
- Director(Director_id)
- Movie-person(Movie_id, Person_id)
- Genre(Genre_id, Genre_name)
- Movie-genre(Movie_id, Genre_id)
- Keywords(Keyword_id, Keyword_name)
- Movie-keyword(Movie_id, Keyword_id)

Comedy Movies Schema: Foreign Keys

The schema includes foreign keys to maintain relationships between tables.

- Movie(Director_id) → Director(Director_id)
- Actor(Actor_id) → Person(Person_id)
- Director(Director_id) → Person(Person_id)
- Movie-person(Movie_id) → Movie(Movie_id)
- Movie-person(Person_id) → Person(Person_id)
- Movie-genre(Movie_id) → Movie(Movie_id)
- Movie-genre(Genre_id) → Genre(Genre_id)
- Movie-keyword(Movie_id) → Movie(Movie_id)
- Movie-keyword(Keyword_id) → Keywords(Keyword_id

Comedy Movies Schema: Description

The schema includes a movies table, which stores the main characteristics of each movie along with a column for the director ID. Additionally, it features a genres table, a keywords table, and a table for relevant people characteristics.

Furthermore, the schema contains:

- An actors table with actor IDs
- A directors table with director IDs
- Link tables connecting movies to people, movies to genres, and movies to keywords.

Database Design: Explanation

The database is designed to aggregate key features of movies that may be of interest to users, while supporting efficient search and retrieval. The design is based on the following principles:

- Information is divided into separate tables for genres, keywords, and people.
- These tables support many-to-many relationships, allowing each value to be shared by multiple movies, and each movie to contain multiple values.
- The Movie table serves as the central repository for movie-specific attributes that cannot have multiple values for the same movie (e.g., title, release year, runtime).
- Inheritance: Actors and directors are both types of people. To avoid duplication, the schema uses separate tables that reference the Person table, following the “is-a” relationship as we’ve learned in this course.
- One-to-Many Relationship: The relationship between a movie and its director is one-to-many, so the Director_id attribute is included as a column in the Movie table.

Database Optimizations

As we saw in class, indexes are critical for improving query efficiency, as they allow data to be retrieved more efficiently and quickly. Thus. We created five indexes to optimize query performance.

The majority of our indexes are applied to the Movie table, as it is the most heavily accessed and contains the bulk of the information in the database.

By optimizing this table, we ensure that the most common queries run efficiently, providing a better user experience.

- Fulltext index on title and overview that supports complex textual search;
- Index on release_year that supports filtering movies by year;
- Index on popularity that supports filtering movies by popularity;
- Index on vote_average that supports filtering movies by vote average;
- Index on keyword_name – the unusual index that is defined on the keyord table. Supports inserting unique keywords.

Query #1: Full-Text Search in Movie Titles & Overviews

- Full-text query
- Allows users to search for movies based on words in the title or overview.
- Outputs: Movie title, genres, release year, overview, and rating.
- Uses MATCH(title, overview) AGAINST(%s) for full-text search.
- Uses GROUP_CONCAT to concatenate all genres into one column.
- Uses JOIN, GROUP BY and ORDER BY.
- The full-text index we created on (title,overview) supports efficient execution of this query.

Query #2: Full-Text Search by Keywords

- Full-text query
- Allow users to enter a keyword and find movies linked to that keyword.
- Outputs: Movie title, release year, director, overview, and rating.
- Uses MATCH(title, overview) AGAINST(%s) for full-text search.
- Uses JOIN and ORDER BY.
- The full-text index we created on keyword_name supports efficient execution of this query.

Query #3: "Director's Favorite Collaborators"

- Complex query
- Finds actors who appeared in the 10 most popular movies of a given director.
- Only includes actors who appeared in more than one of the director's top 10 movies.
- Outputs: Actor’s name, birthday, and the number of times they worked with the director.
- Uses a nested query, JOIN, GROUP BY, HAVING and ORDER BY.
- The index we created on release year and popularity supports efficient execution of this query.

Query #4: "Hall of Fame" for a Sub-Genre & Decade

- Complex query
- Finds actors who starred in the most movies of a specific sub-genre in a given decade.
- Outputs: Actor’s name, birthday, and number of movies.
- Uses JOIN and GROUP BY.
- The index we created on release year supports efficient execution of this query.

Query #5: “Hidden Gems”

- Complex query
- Finds unpopular by highly rated movies in a given year.
- Criteria: Rating > 7.0 & Popularity < average popularity.
- Outputs: Movie title, overview, rating (vote average) , and popularity.
- Uses a nested query, aggregation (AVG) and ORDER BY.
- The indexes we created on release year, popularity and vote average supports efficient execution of this query.

Extra Queries

We decided to add three more queries for the sake of completeness and app functionality.

- Query #6: "Most popular movies of a director“
  - Finds the 5 most popular movies for a given director.
  - Outputs: Movie title, overview, release year and popularity.
- Query #7: "Most popular movies of an actor“
  - Finds the 5 most popular movies for a given actor.
  - Outputs: Movie title, overview, release year and popularity.
- Query #8: "Most popular genres of an actor“
  - Shows which genres an actor has appeared in the most.
  - Outputs: Genre name and the number of movies.

Code Structure and API Usage

We used the TMDB API, which provides information on over a million movies.

- In the create_db_script.py file, we define the tables with the appropriate columns, as previously described, and create indexes.
- In the api_data_retrieve.py file, the API retrieves top-ranked comedy films that have received at least 500 votes (ensuring their ranking is reliable). Various functions populate the tables with the relevant movie data.
- The queries_db_script.py file contains functions that execute the queries we wrote, while the queries_execution.py file runs these functions and displays the results.