

The database schema consists of five main tables: **movies**, **genres**, **persons**, **movie_genres**, and **movie_cast**. Below is a detailed description of each table:

Tables and Columns:

1. **movies:**

- Columns:
 - **movie_id** (INT, PRIMARY KEY): Unique identifier for each movie.
 - **title** (VARCHAR(255), NOT NULL): Title of the movie.
 - **release_date** (DATE, NOT NULL): Release date of the movie.
 - **vote_average** (FLOAT): Average rating of the movie.
 - **overview** (TEXT): Brief description of the movie.
 - **popularity** (FLOAT): Popularity score of the movie.
- Indexes:
 - **idx_overview** (FULLTEXT): Full-text index on the overview column for efficient text searches.
 - **idx_popularity** (INDEX): Index on the popularity column for efficient sorting and filtering.

2. **genres:**

- Columns:
 - **genre_id** (INT, PRIMARY KEY): Unique identifier for each genre.
 - **name** (VARCHAR(255), NOT NULL): Name of the genre.
- Indexes:
 - **idx_genre_id** (INDEX): Index on the genre_id column for efficient joins.

3. **persons:**

- Columns:
 - **person_id** (INT, PRIMARY KEY): Unique identifier for each person (actor or crew member).
 - **name** (VARCHAR(255), NOT NULL): Name of the person.
- Indexes:
 - **idx_person_id** (INDEX): Index on the person_id column for efficient joins.

4. **movie_genres:**

- Columns:
 - movie_id (INT): Foreign key referencing movies(movie_id).
 - genre_id (INT): Foreign key referencing genres(genre_id).
- Indexes:
 - idx_movie_genre (INDEX): Composite index on movie_id and genre_id for efficient joins.

5. **movie_cast:**

- Columns:
 - movie_id (INT): Foreign key referencing movies(movie_id).
 - person_id (INT): Foreign key referencing persons(person_id).
 - role (VARCHAR(255)): Role of the person in the movie (e.g., Actor, Director).
 - character_name (VARCHAR(255)): Name of the character played by the actor.
- Indexes:
 - idx_cast_movie (INDEX): Composite index on movie_id and person_id for efficient joins.
 - idx_cast_role (INDEX): Composite index on person_id and role for efficient filtering by role.
 - idx_role (INDEX): Index on the role column for efficient filtering.

Efficiency Considerations:

- **Normalization:** The database schema is normalized to reduce redundancy and ensure data integrity. Each table represents a distinct entity (movies, genres, persons) and their relationships (movie_genres, movie_cast).
- **Indexes:** Indexes are strategically placed to optimize query performance. Full-text indexes on the overview column enable efficient text searches, while composite indexes on join columns (movie_id, genre_id, person_id) ensure fast joins and aggregations.
- **Foreign Keys:** Foreign key constraints enforce referential integrity, ensuring that relationships between tables are maintained correctly.

Drawbacks of Alternative Designs:

- **Denormalization:** Denormalizing the schema (e.g., combining movies, genres, and cast into a single table) could lead to data redundancy and increased storage requirements. It would also complicate updates and increase the risk of data inconsistencies.
- **Lack of Indexes:** Omitting indexes would result in slower query performance, especially for complex queries involving joins, aggregations, and text searches.

Database Optimizations:

- **Index Usage:**
 - **Full-Text Index:** idx_overview on the overview column of the movies table enables efficient full-text searches using the MATCH ... AGAINST syntax.
 - **Composite Indexes:** idx_movie_genre on the movie_genres table (movie_id, genre_id) optimizes joins between movies and genres. idx_cast_movie on the movie_cast table (movie_id, person_id) optimizes joins between movies and cast members. idx_cast_role on the movie_cast table (person_id, role) optimizes filtering by role.
 - **Single-Column Indexes:** idx_popularity on the popularity column of the movies table optimizes sorting and filtering by popularity. idx_genre_id on the genre_id column of the genres table and idx_person_id on the person_id column of the persons table optimize joins.

Main Queries:

- 1. Find the Top 5 Movies Mentioning 'Leonardo DiCaprio' in Their Overview with the Highest Average Rating, Including Genres:**
 - Purpose: Identify highly-rated movies that mention 'Leonardo DiCaprio' in their overview.
 - Database Design Support: Uses the idx_overview full-text index for efficient text search. Joins with movie_genres and genres tables to include genre information.
- 2. Find the Top 5 Most Popular Movies Along with Their Genres and the Number of Actors in Each Movie:**
 - Purpose: Identify the most popular movies, their genres, and the number of actors involved.
 - Database Design Support: Uses the idx_popularity index for efficient sorting by popularity. Joins with movie_genres, genres, and movie_cast tables to gather genre and actor information.
- 3. Find Movies with the Most Diverse Cast (Actors from Different Genres):**
 - Purpose: Identify movies with a diverse cast, measured by the number of different genres represented by the actors.
 - Database Design Support: Joins with movie_cast, movie_genres, and genres tables to count distinct genres.
- 4. Find the Highest-Rated Movie in Each Genre, Along with Its Rating, Genre Name, and Popularity:**
 - Purpose: Identify the top-rated movie in each genre.
 - Database Design Support: Uses subqueries to find the highest-rated movie per genre. Joins with movie_genres and genres tables to include genre information.
- 5. Find Top 5 Directors by Average Vote_Average of the Movies They Directed:**
 - Purpose: Identify the top 5 directors based on the average rating of their movies.
 - Database Design Support: Joins with movie_cast and persons tables to gather director information. Uses aggregation to calculate the average rating.

Code Structure and API Usage:

- **Code Structure:**

- `src/create_db_script.py`: Creates the database schema and indexes. Ensures tables and indexes are created if they do not already exist.
- `src/api_data_retrieve.py`: Fetches data from the TMDB API and populates the database. Includes functions to fetch movies, genres, and credits, and insert them into the database.
- `src/queries_db_script.py`: Contains the SQL query functions (`query_1`, `query_2`, `query_3`, `query_4`, `query_5`). Each function connects to the database, executes the query, and prints the results.
- `src/queries_execution.py`: Demonstrates the execution of the queries. Calls each query function and prints the results.

- **API Usage:**

- **TMDB API**: The application uses the TMDB API to fetch movie data, genres, and credits.
- **API Endpoints**:
 - `/movie/popular`: Fetches popular movies.
 - `/genre/movie/list`: Fetches movie genres.
 - `/movie/{movie_id}/credits`: Fetches credits for a specific movie.
- **API Key and Base URL**: Stored in environment variables (`TMDB_API_KEY`, `TMDB_BASE_URL`).