System Documentation: DBMS Project - The Polyglot Cinema

Team Members: Edan Weinberg 209363381, Nadav Eyal 323870188

1. Database Schema Structure

The database is designed to store movie metadata across five distinct languages to support cross-language analysis. It consists of the following **entity tables**:

Languages

- `lang_code` (VARCHAR) - ISO 639-1 code identifying the language (e.g., 'en' for English, 'sv' for Swedish).
- `lang_name` (VARCHAR) - Full English name of the language (e.g., 'English', 'Swedish').

Movies

- `movie_id` (INT) - Unique identifier for the movie in the database.
- `title` (VARCHAR NOT NULL) - Movie title in the language of the dataset.
- `original_title` (VARCHAR) - Title in the movie's original language.
- `tagline` (VARCHAR) - Short marketing phrase or slogan for the movie.
- `overview` (TEXT) - Plot summary or description of the movie.
- `release_year` (INT) - Year the movie was released.
- `runtime` (INT) - Movie duration in minutes.
- `popularity` (FLOAT) - Popularity score from TMDB, higher means more popular.
- `vote_average` (FLOAT) - Average rating from user votes.
- `vote_count` (INT) - Number of user votes contributing to the average rating.
- `lang_code` (VARCHAR) - Language code of the movie (foreign key to `languages.lang_code`).
- `budget` (INT) - Production budget in USD.

Genres

- `genre_id` (INT) - Unique identifier for each genre.
- `name` (VARCHAR) - Name of the genre.

Movie_Genres

Purpose: Implements a many-to-many relationship between movies and genres.

- `movie_id` (INT) - foreign key to `movies(movie_id)`
- `genre_id` (INT) - foreign key to `genres(genre_id)`

People

- `person_id` (INT) - Unique identifier for a person (e.g., director, actor).
- `name` (VARCHAR) - Person's full name.
- `popularity` (FLOAT) - Popularity metric from TMDB.

Movie_Crew

Purpose: Links people to movies with specific roles, allowing many-to-many relationships while tracking jobs.

- `movie_id` (INT) - foreign key to `movies(movie_id)`
- `person_id` (INT) - foreign key to `people(person_id)`
- `job` (VARCHAR) - Role in the movie (e.g., 'Director', 'Producer').

Relations Between Tables

1. **Languages → Movies**
   - Type: One-to-Many
   - Description: One language can have many movies. Each movie has exactly one original language
   - Foreign Key: `movies.lang_code → languages.lang_code`
2. **Movies ↔ Genres**
   - Type: Many-to-Many
   - Description: A movie can belong to multiple genres. A genre can be associated with multiple movies. This relationship is implemented via the junction table `movie_genres`.
   - Foreign Keys: `movie_genres.movie_id → movies.movie_id`, `movie_genres.genre_id → genres.genre_id`
3. **People ↔ Movies**

- Type: Many-to-Many
- Description: A person (e.g., director, producer, writer) can work on multiple movies. A movie can have multiple crew members. This relationship is implemented via the junction table `movie_crew`.
- Foreign Keys: `movie_crew.person_id → people.person_id`, `movie_crew.movie_id → movies.movie_id`

2. Database Design Reasoning

Use of Junction Tables

The database contains several many-to-many relationships:

- A movie can belong to multiple genres.
- A genre can be associated with multiple movies.
- A movie can have multiple crew members.
- A person can work on multiple movies.

To handle these relationships efficiently, we use junction tables (`movie_genres` and `movie_crew`).

Why we designed it this way:

- `movie_genres` - links movies and genres to avoid duplicating movie data and allows fast genre-based queries.
  - *Alternative 1*: Add genre info to the `movies` table or add movie info to the `genres` table.
  - *Problem*: Both approaches duplicate data - either the movie data or the genre data - and make queries and updates more difficult.
  - *Alternative 2*: Store all genres in a single text field in `movies`.
    Problem: This makes it inconvenient to parse and query movies by genre efficiently.
- `movie_crew` - links people to movies and tracks their role.
  - *Alternative:* Add crew data in `movies` or a list of movies in `people`.
  - *Problem:* Both options duplicate data and complicate updates.

Primary and Foreign Keys

- **Primary Keys:** `movie_id, genre_id, person_id` for main tables. Composite keys (`movie_id + genre_id`, `movie_id + person_id + job`) for junction tables.
- **Foreign Keys:** Link junction tables to main tables to ensure data integrity and prevent orphan records.

Choice of Languages and Dataset Size

- Five languages: English, French, Spanish, Swedish, German.
- 1,000 movies per language for balanced, comparable cross-language analysis.
- Dataset is manageable but large enough for meaningful queries. Additional languages can be added easily.

3. Indexing Strategy

Movies Table:

- `idx_movies_lang_budget (lang_code, budget)` - speeds up query 1 (finding blockbusters per language) by allowing the database to quickly filter movies by language and budget without scanning the entire table.
- `idx_movies_lang_vote (lang_code, vote_average)` - helps query 3 (finding the top-rated movie for a genre in a given language) by keeping vote_average sorted for fast retrieval of the maximum rating.
- `ft_idx_movie_content (title, overview, tagline)` - allows queries 4 & 5 (searching for specific topics like war, crime, dark mysteries) to use efficient full-text search rather than scanning text row by row..

Movie_Crew Table:

- `idx_movie_crew_job (job)` - optimizes query 2 (finding top directors per language) by quickly filtering for 'Director' rows without scanning the entire crew table.

Genres Table:

- `idx_genres_name (name)` - helps queries 3 & 4 by allowing fast filtering of movies by genre.

4. Main Queries Description

Query 1 - Blockbuster Movies per Language

- *Purpose:* This query retrieves blockbuster movies in a selected non-English language. A blockbuster is defined as a movie with a budget higher than the average budget of English language movies.
- *Usefulness:* The query helps users who want to learn a new language through cinema discover high-quality, popular movies in that language. By focusing on well-funded productions, users are more likely to encounter influential and culturally significant films.
- *Database Support:* The index `idx_movies_lang_budget` improves performance by enabling efficient filtering based on language and budget without scanning the entire movies table.

Query 2 - Top Directors per Language

- *Purpose:* This query identifies the director or directors who directed the highest number of movies in a selected language.
- *Usefulness:* This query helps users discover leading directors in a chosen language. Watching multiple films by the same director allows learners to better understand the language, storytelling style, and cultural patterns.
- *Database Support:* The index `idx_movie_crew_job` speeds up execution by optimizing filtering on crew members with the role of “Director”.

Query 3 - Top-Rated Movie per Genre for a Language

- *Purpose:* Find the top-rated movie for a specific genre and language.
- *Usefulness:* Enables language learners to choose well-reviewed movies within genres they enjoy, increasing motivation while studying the target language.
- *Database Support:* `idx_movies_lang_vote` and `idx_genres_name` improve filtering by language, rating, and genre.

Query 4 - (FULLTEXT) Historical War Movies Excluding Comedy

- *Purpose:* Return movies with the keywords “war,” “history,” and exclude movies whose genres include comedy, science fiction, or fantasy, in order to focus on serious films.

- *Usefulness:* Allows learners to focus on serious, content-rich films, helping them acquire topic-specific vocabulary and gain cultural and historical insights relevant to the target language. It can also be beneficial to teachers for instance. They can also use these results to integrate films into educational lessons.
- *Database Support:* `ft_idx_movie_content` allows fast keyword search.

Query 5 - (FULLTEXT) Dark Crime Scandinavian Movies

- *Purpose:* Return Scandinavian crime movies.
- *Usefulness:* Helps learners focus on genre-specific vocabulary.
- *Database Support:* `ft_idx_movie_content` enables fast and effective thematic searches.

5. Code Structure Overview

1. **create_db_script.py** -

- *Purpose:*
    - Responsible for creating the database schema, including tables and indices.
- *Contents*:
    - Functions for creating each table (languages, movies, genres, movie_genres, people, movie_crew).
    - Functions to create indices to optimize queries
    - A get_connection() function to connect to the MySQL server.
    - The main() function checks the connection, creates tables, adds indices, and commits the schema to the database.

2. **api_data_retrieve.py**

- *Purpose:*
    - Handles data retrieval from TMDB API (The Movie Database API) and inserts it into the database.
- *Contents*:
    - Functions to fetch movie data (fetch_discover, fetch_full_movie) using the TMDB API.
    - Functions to insert data into tables (insert_language, insert_movie, insert_genres, insert_crew).
    - The main() function orchestrates fetching movies for multiple languages (English, French, Spanish, Swedish, German), processes the results, and populates the database.

3. **queries_db_script.py**

- *Purpose:*
    - Contains all the database query functions.
- *Contents*:
    - Each query is implemented as a separate Python function (query_1 to query_5).
    - Queries include full-text searches (for keyword-based movie search) and complex queries (using joins, nested queries, aggregation, and GROUP BY).
    - Functions accept input parameters (e.g., lang_code, genre_name) and return results as lists of tuples, where each tuple represents a database row.

4. **queries_execution.py**

- *Purpose:*
    - Demonstrates the execution of queries from queries_db_script.py.
- *Contents***:**
    - Calls each query function (with parameters when needed).
    - Prints query results in a clear and organized format.
    - Does not fetch new data, but operates on the database populated by api_data_retrieve.py.
    - Shows how the database and queries can be used by an application or Python script to retrieve meaningful results.

API Usage

- **TMDB API (The Movie Database)**:
    - Used to fetch movie metadata, languages, genres, and crew information.
    - Functions handle pagination, language filtering, and details enrichment.
    - All API results are processed in Python and inserted into the database.
- Purpose in Project: Provides a dynamic, real-world dataset for populating the database with at least 5,000 records across multiple tables and languages.