System Documentation

February 2, 2025

Ibrahim Abomokh
ID: 315270678

Mohamed Jabali
ID: 212788293

Database Schema Description

The database schema includes the following tables:

- **Movie**: Contains columns such as `movie_id` (primary key), `title`, `average_rating`, `release_date`, `budget`, `revenue`, `runtime`, and `meta_score`. A `FULLTEXT` index on `title` (`idx_movie_title`) supports efficient keyword searches.
- **Person**: Contains the column `name` (primary key) to represent individuals involved in movies.
- **Staff_Movie**: Represents the relationship between staff members and movies, with columns `person_name`, `movie_id`, and `role` (an `ENUM` for roles like `actor`, `writer`, and `director`). Foreign keys reference `Person.name` and `Movie.movie_id`. A `FULLTEXT` index on `person_name` (`idx_person_name`) facilitates searches.
- **Country**: Contains the column `country_name` (primary key) to represent countries associated with movies.
- **Movie_Country**: Links movies to countries via `movie_id` and `country_name`. Foreign keys reference `Movie.movie_id` and `Country.country_name`.
- **Language**: Contains the column `language_name` (primary key) to represent languages in which movies are available.
- **Movie_Language**: Links movies to languages via `movie_id` and `language_name`. Foreign keys reference `Movie.movie_id` and `Language.language_name`.

1 Reasoning for the Database Design

Relational Database Justification

A relational database was chosen due to its ability to:

- Maintain strong consistency and support complex queries using structured data.
- Efficiently handle relationships between movies, staff, and languages via foreign keys and joins.

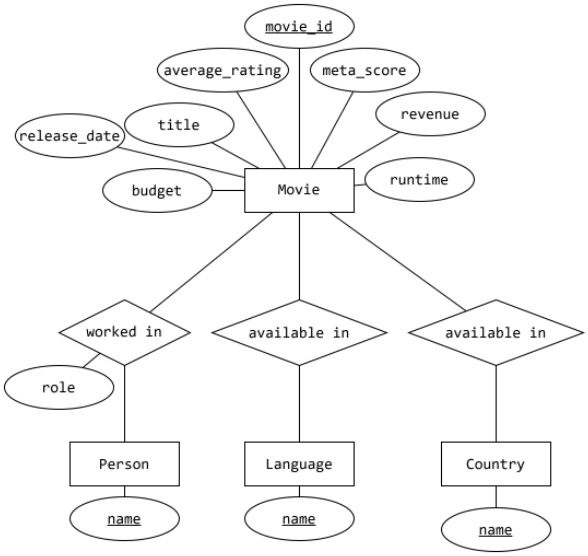

Figure 1: ER Diagram for the database

Design Alternatives

- **IDs**: we decided to add ID to the movies, we could instead use the title as the primary key for the movie. However, this approach could result in data redundancy and inefficiency. For instance, if the movie name needs to be updated, every occurrence of the name across multiple tables would require modification. In contrast, our design ensures consistency and efficiency by using a unique movie-id as a reference, allowing updates to be made in a single location—the Movie table.

2 Database Optimizations

- **Indexes**:
  - Use `FULLTEXT` indexes on `Movie.title` and `Staff_Movie.person_name` for fast keyword searches.
  - Add indexes to foreign key columns (e.g., `Staff_Movie.movie_id`, `Movie_Lang.movie_id`) to improve join performance.
  - Index columns used for sorting, such as `release_date` and `budget`.
- **Normalization**: The schema minimizes redundancy and maintains data integrity.

3 Main Queries

Query 1: Search Movies by Title

Purpose: Finds movies containing a specific keyword in their title.
Support: `FULLTEXT` search on `Movie.title` ensures fast keyword matching.

Query 2: Filter Movies by Staff Member

Purpose: Retrieves movies filtered by staff names and roles.
Support: `FULLTEXT` search on `Staff_Movie.person_name` combined with a join to the `Movie` table.

Query 3: Revenue Grouped by Release Year

Purpose: Calculates yearly total revenue grouped by release year.
Support: Indexes on `release_date` and `revenue` enhance query performance.

Query 4: Actors with Higher Ratings

Purpose: Lists actors whose average movie rating exceeds the global average.
Support: Aggregates ratings from the `Movie` table and groups by `Staff_Movie.person_name`.

Query 5: High-Budget Multilingual Movies

Purpose: Identifies movies with budgets over $100M, available in multiple languages, and includes a boolean for Arabic availability.
Support: Joins between `Movie` and `Movie_Language` support multilingual analysis, and a `CASE` statement handles the boolean calculation.

4 Code Structure and API Usage

The `/queries_execution.py` file is responsible for executing and demonstrating the functionality of the queries defined in the `queries_db_script.py` module. It includes the following key functions:

- `exe_query_1(keyword)`: Executes `query_1` with the provided keyword. It:
  - Prints a separator and the query description (from the query's docstring).
  - Passes the `keyword` parameter to `query_1`.
  - Executes the query using `execute_query`, printing results in a tabular format.
- `exe_query_2(keyword)`: Executes `query_2` with the provided keyword. It performs the same steps as `exe_query_1`, using `query_2` to filter by staff member names.
- `exe_query_3()`: Executes `query_3` without any parameters. This function calculates the total revenue grouped by release year and displays the results in ascending order of years.
- `exe_query_4()`: Executes `query_4`, which identifies actors whose average movie ratings exceed the global average. Results are sorted by actor average rating in descending order, and only the top 20 actors are shown.
- `exe_query_5()`: Executes `query_5`, listing movies with a budget over $100M, produced in multiple languages, and indicating whether they are available in Arabic. Results are sorted by budget in descending order.

- `main()`: Serves as the entry point of the script. It:
  - Calls each execution function sequentially: `exe_query_1`, `exe_query_2`, `exe_query_3`, `exe_query_4`, and `exe_query_5`.
  - Uses predefined constants `KEYWORD_1` (`"adventures"`) and `KEYWORD_2` (`"Hiddleston"`) for queries requiring keywords.
  - Handles errors related to database operations using a `try-except` block, ensuring graceful error reporting.

This modular structure allows for the straightforward extension and maintenance of query executions. Each execution function is tightly coupled with a corresponding query, enabling clear separation of concerns and enhanced readability.

Output Formatting

The `tabulate` library is used for clean and structured query result presentation.

5 Web Application

Overview

This Flask-based API interacts with a MySQL database to retrieve movie-related data. It supports searching, filtering, and executing predefined SQL queries. In addition to the frontend design provided in the submission files, we have provided a front end implementation of our web application.

Dependencies

Install dependencies with:

```
pip install flask flask-cors mysql-connector-python
```

API Endpoints

/search

GET /search

- Searches for movies by title.
- **Query Parameters:** query (optional, string)

/movies

GET /movies

- Retrieves movies with optional filters.
- **Query Parameters:** release year, rating min, language, country

/custom-query

GET /custom-query

- Executes the predefined queries, those are the queries explained above.
- **Query Parameters:** query type (1-5), keyword (optional)

Running the Application

To run the Application you can use live server extension in VScode.
Start the server with:

```
python3 server.py
```