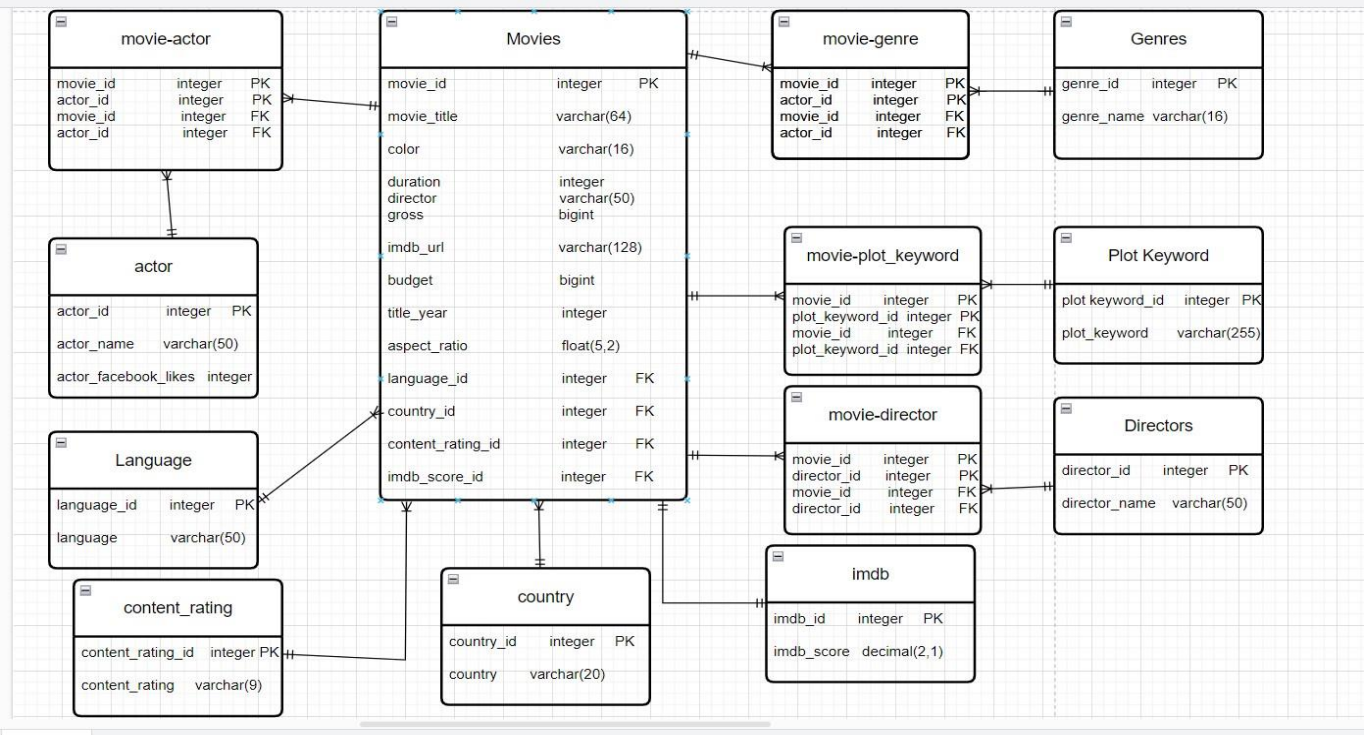
Data Source: https://www.kaggle.com/code/saurav9786/imdb-score-prediction-for-movies.

Entity Relation Diagram:



1. **Table:** movie

   Description: Stores information about movies.

```
CREATE TABLE movie(
movie_id INT PRIMARY KEY,
movie_title VARCHAR(64),
color VARCHAR(16), duration
INT, gross BIGINT, imdb_url
VARCHAR(128),
    budget BIGINT,
    title_year INT, aspect_ratio
VARCHAR(3), director_id INT
FOREIGN KEY, imdb_id INT
FOREIGN KEY, country_id INT
FOREIGN KEY, language_id
INT FOREIGN KEY,
    content_rating_id INT FOREIGN KEY
)ENGINE=INNODB;
```

2. Table: genre

```
DROP TABLE IF EXISTS genre;
CREATE TABLE genre(
genre_id INT PRIMARY KEY AUTO_INCREMENT, genre
VARCHAR(16)
)ENGINE=INNODB;
```

3. Table: movie_genre

```
DROP TABLE IF EXISTS movie_genre;
CREATE TABLE movie_genre(
movie_id int,
genre_id int,
PRIMARY KEY(movie_id, genre_id),
FOREIGN KEY(movie_id) REFERENCES movie(movie_id),
FOREIGN KEY(genre_id) REFERENCES genre(genre_id)
)ENGINE=INNODB;
```

4. Table: plot_keyword

```
DROP TABLE IF EXISTS plot_keyword;
CREATE TABLE plot_keyword(
plot_keyword_id INT PRIMARY KEY AUTO_INCREMENT,
plot_keyword VARCHAR(149)
)ENGINE=INNODB;
```

5. Table: movie_plot_keyword

```
DROP TABLE IF EXISTS movie_plot_keyword;
CREATE TABLE movie_plot_keyword(
movie_id int, plot_keyword_id int,
PRIMARY KEY(movie_id, plot_keyword_id),
FOREIGN KEY(movie_id) REFERENCES movie(movie_id),
FOREIGN KEY(plot_keyword_id) REFERENCES plot_keyword(plot_keyword_id)
)ENGINE=INNODB;
```

6. Table: actor

```
DROP TABLE IF EXISTS actor;
CREATE TABLE actor(
actor_id INT PRIMARY KEY AUTO_INCREMENT,
actor_name VARCHAR(125),
actor_facebooklikes INT(11)
)ENGINE=INNODB;
```

7. Table: movie_actor

```
DROP TABLE IF EXISTS movie_actor;
CREATE TABLE movie_actor( movie_id
int, actor_id int,
PRIMARY KEY(movie_id, actor_id),
FOREIGN KEY(movie_id) REFERENCES movie(movie_id),
FOREIGN KEY(actor_id) REFERENCES actor(actor_id)
)ENGINE=INNODB;
```

8. Table: director

```
DROP TABLE IF EXISTS director;
CREATE TABLE director(
director_id INT PRIMARY KEY AUTO_INCREMENT,
director VARCHAR(32), director_facebooklikes
INT(11)
)ENGINE=INNODB;
```

9. Table: language

```
DROP TABLE IF EXISTS language;
CREATE TABLE language(
language_id INT PRIMARY KEY AUTO_INCREMENT,
language VARCHAR(10)
)ENGINE=INNODB;
```

10. Table: country

```
DROP TABLE IF EXISTS country;
CREATE TABLE country(
country_id INT PRIMARY KEY AUTO_INCREMENT,
country VARCHAR(20)
)ENGINE=INNODB
```

11. Table: imdb

```
DROP TABLE IF EXISTS imdb;
CREATE TABLE imdb(
imdb_id INT PRIMARY KEY AUTO_INCREMENT,
imdb_score DECIMAL(2,1)
)ENGINE=INNODB;
```

12. Table: content_rating

```
DROP TABLE IF EXISTS content_rating; CREATE
TABLE content_rating(
content_rating_id INT PRIMARY KEY AUTO_INCREMENT,
content_rating VARCHAR(9)
)ENGINE=INNODB;
```

Preprocessing Steps:

1. Data Deduplication:

- Removed duplicate data from the movie spreadsheet, ensuring each movie record is unique.

2. Add Unique Identifier:

- Added a "movie_id" column to the movie spreadsheet to uniquely identify each record.
- Saved the spreadsheet with the "CSV UTF-8 (comma delimited) (.csv)" extension.

3. Import Movies Data:

- Imported the unnormalized dataset "movies-unf" using phpMyAdmin.

4. Genre Data Transformation:

- Copied the "movie_id" and "genre" columns from "movies-unf" to a separate tab.
- Utilized the "Text to Columns" feature, specifying delimiter settings (e.g., pipe), to create unique columns for each genre.
- Saved this transformed data as a CSV file with "CSV UTF-8 (comma delimited) (.csv)" extension.

5. Import Genre Data:

- Imported the unnormalized dataset "movie_genre-unf" using phpMyAdmin.

6. Repeat for Actors and Plot Keywords:

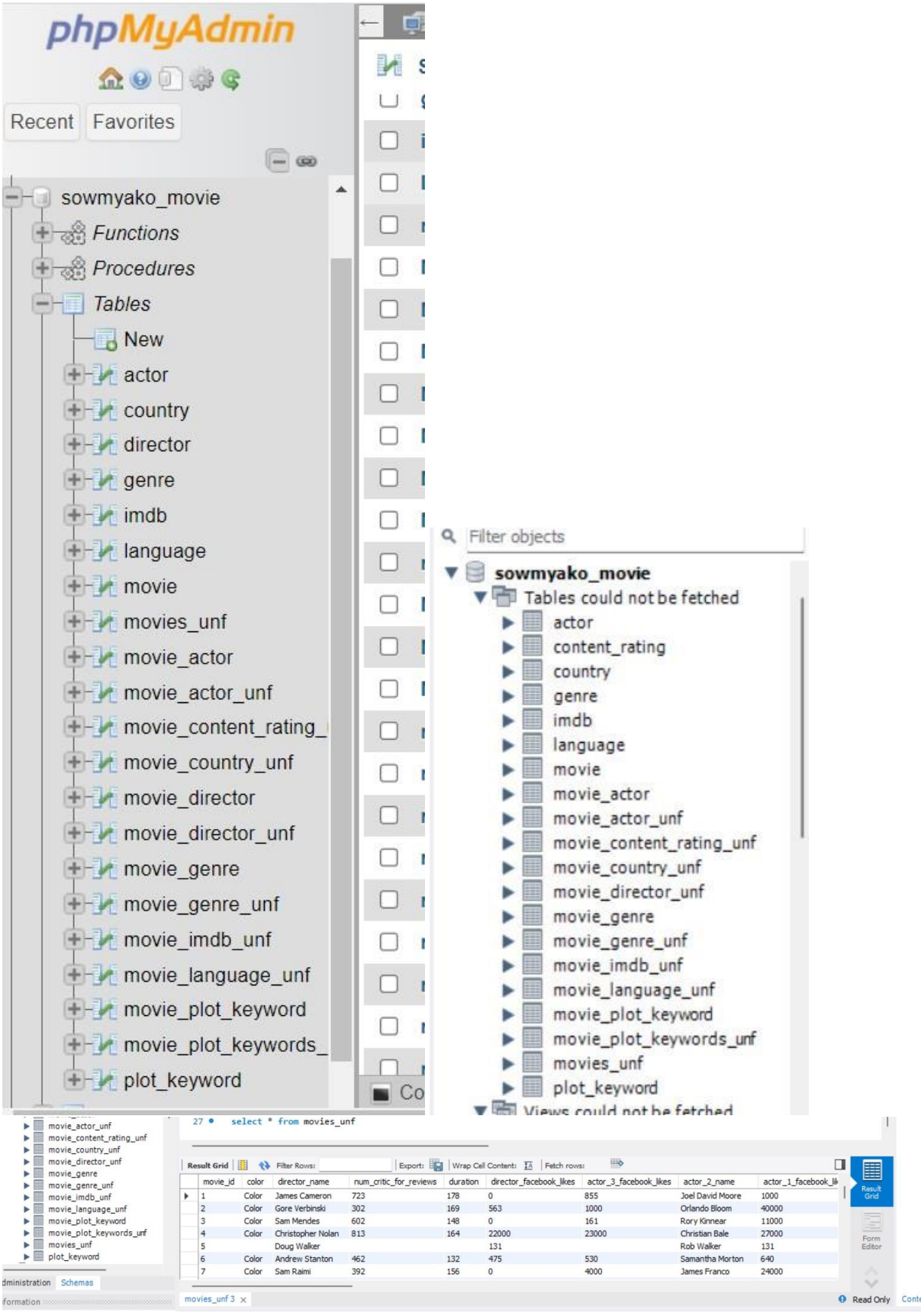
- Followed the same steps (steps 4 to 5) for actors and plot keywords, creating separate tabs and applying text-to-columns for data transformation.

7. Separate Columns:

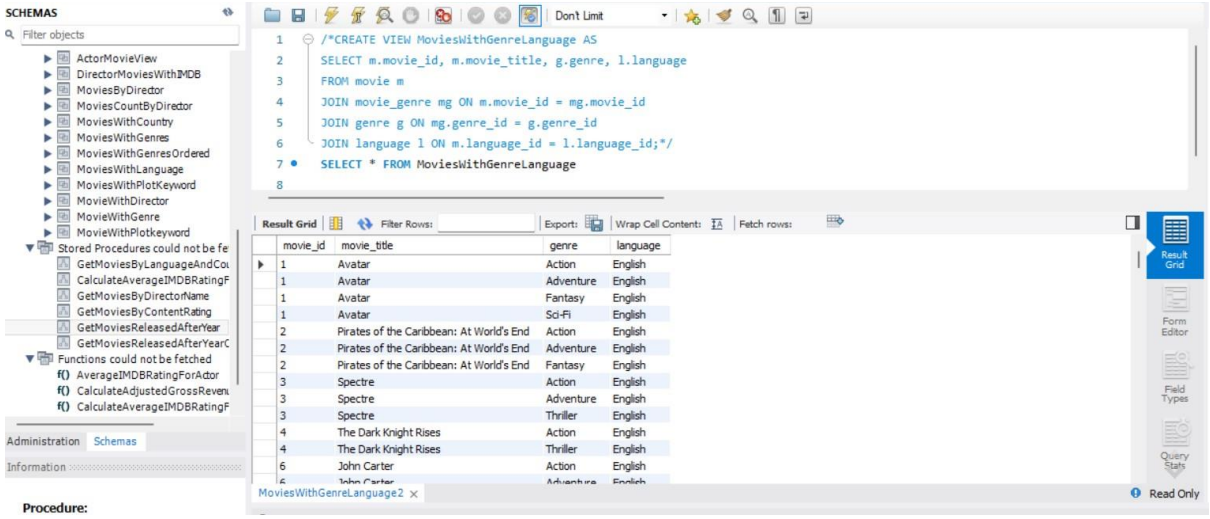
- Separated the columns for "country," "imdb_score," "language," and "content_rating" into individual tabs.
- In each tab, combined "movie_id" with the respective data (e.g., "movie_id" and "country" in one tab, "movie_id" and "language" in another tab).
- Saved all tabs with "CSV UTF-8 (comma delimited) (.csv)" extension.

8. Import Additional Data:

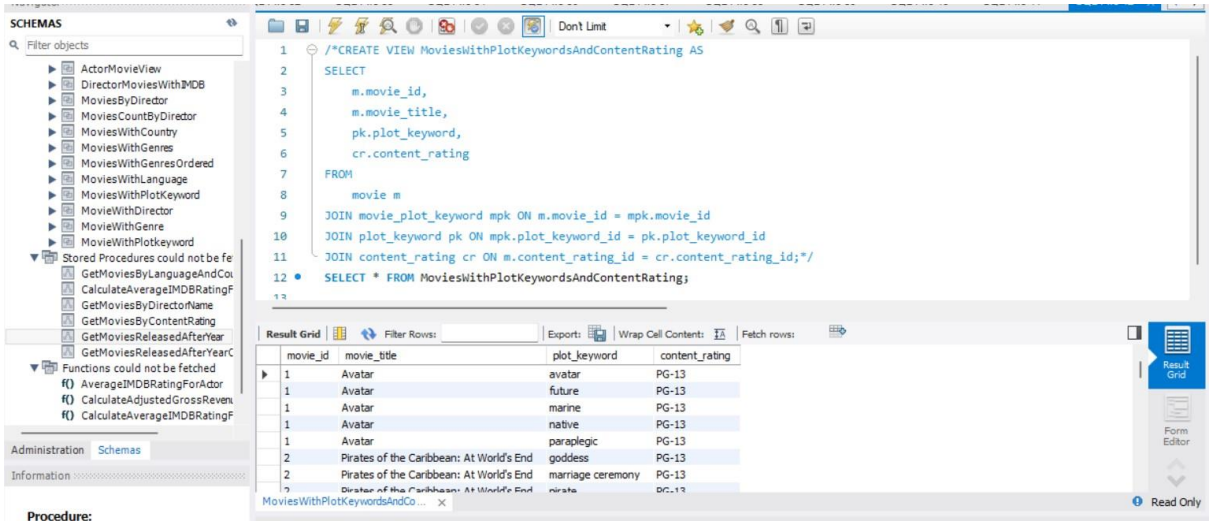
- Imported the unnormalized datasets "movie_language-unf," "movie_country-unf," "movie_content_rating-unf," and "movie_imdb_score-unf" into the MySQL database using phpMyAdmin.


phpMyAdmin
Recent
Favorites
sowmyako_movie
Functions
Procedures
Tables
New
actor
country
director
genre
imdb
language
movie
movies_unf
movie_actor
movie_actor_unf
movie_content_rating_
movie_country_unf
movie_director
movie_director_unf
movie_genre
movie_genre_unf
movie_imdb_unf
movie_language_unf
movie_plot_keyword
movie_plot_keywords_
plot_keyword
Filter objects
sowmyako_movie
Tables could not be fetched
actor
content_rating
country
genre
imdb
language
movie
movie_actor
movie_actor_unf
movie_content_rating_unf
movie_country_unf
movie_director_unf
movie_genre
movie_genre_unf
movie_imdb_unf
movie_language_unf
movie_plot_keyword
movie_plot_keywords_unf
movies_unf
plot_keyword
Views could not be fetched
27 • select * from movies_unf
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
Fetch rows:
movie_id color director_name num_critic_for_reviews duration director_facebook_likes actor_3_facebook_likes actor_2_name actor_1_facebook_lik
1 Color James Cameron 723 178 0 855 Joel David Moore 1000
2 Color Gore Verbinski 302 169 563 1000 Orlando Bloom 40000
3 Color Sam Mendes 602 148 0 161 Rory Kinnear 11000
4 Color Christopher Nolan 813 164 22000 23000 Christian Bale 27000
5 Doug Walker 131 Rob Walker 131
6 Color Andrew Stanton 462 132 475 530 Samantha Morton 640
7 Color Sam Raimi 392 156 0 4000 James Franco 24000
Result Grid
Form Editor
Read Only
movies_unf 3

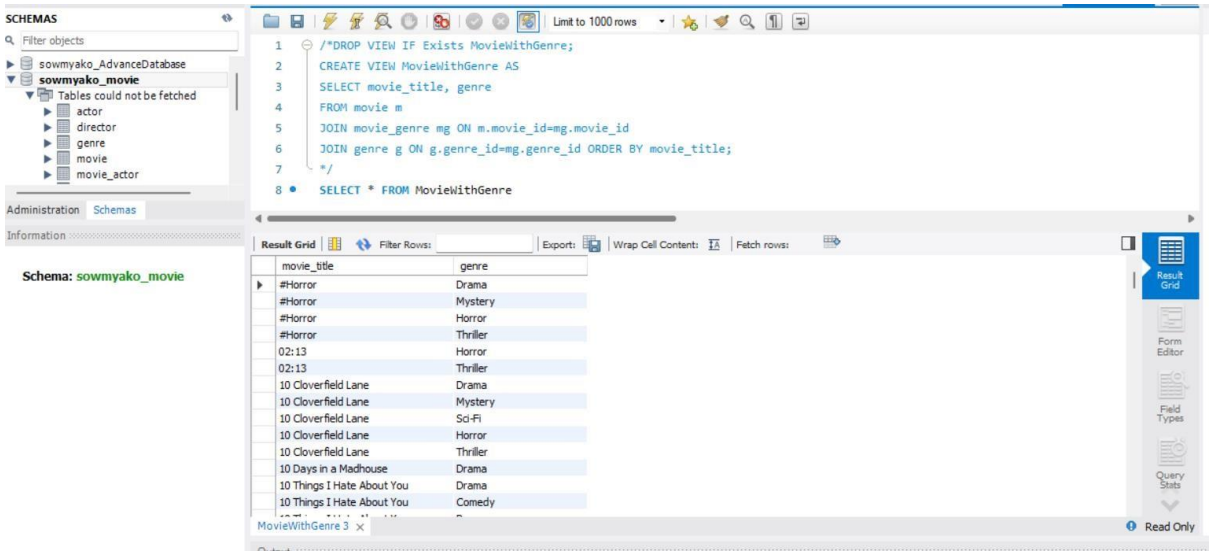
1. **View:** MoviesWithGenreLangauage



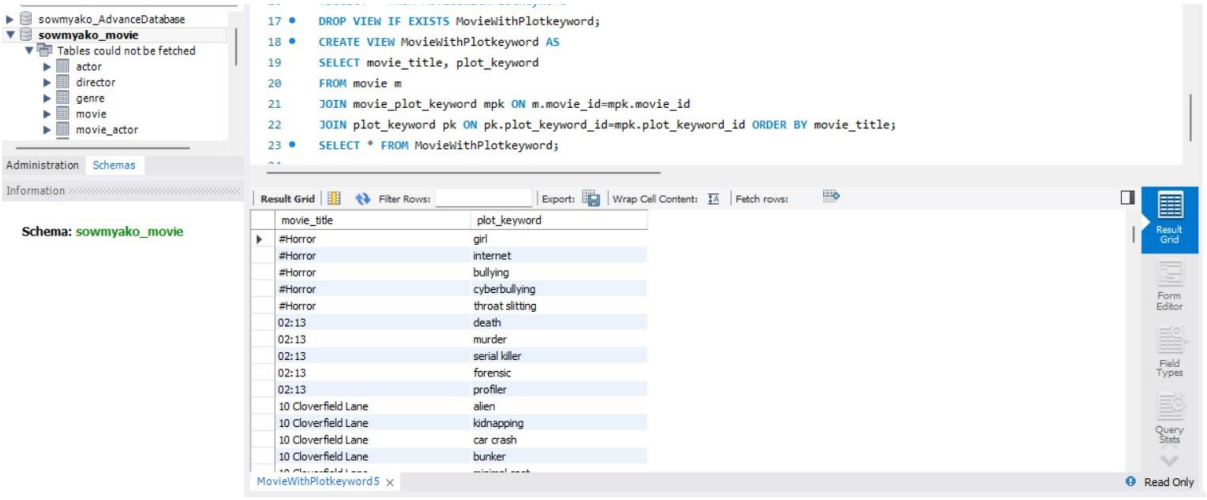
2. **View:** MovieWithPlotKeywordAndContentrating



3. **View:** MovieWithGenre



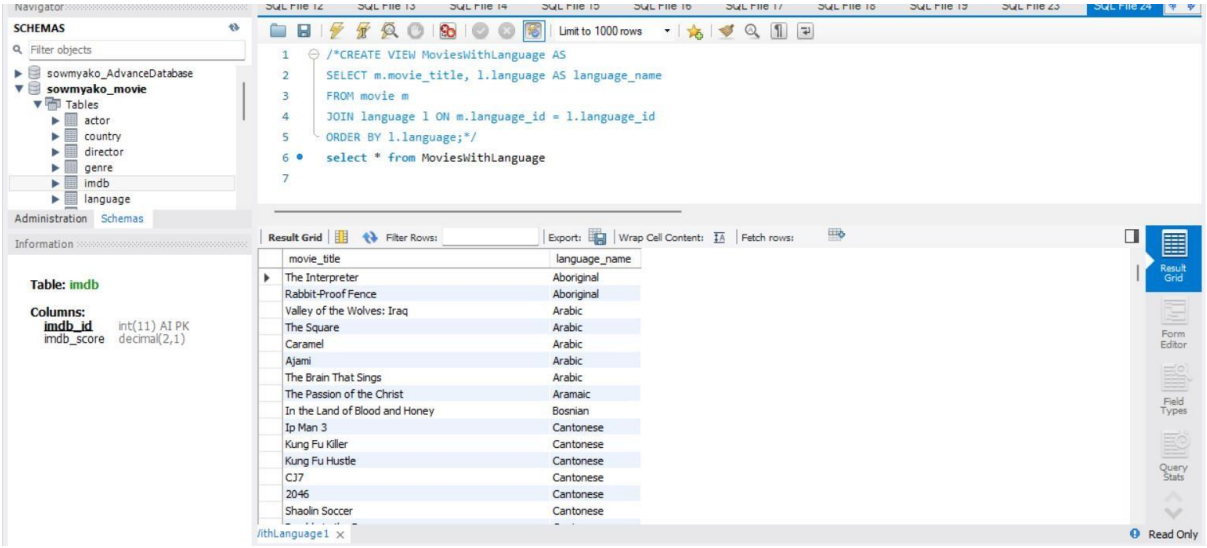
4. **View:** MovieWithPlotkeyword



```
DROP VIEW IF EXISTS MovieWithPlotkeyword;
CREATE VIEW MovieWithPlotkeyword AS
SELECT movie_title, plot_keyword
FROM movie m
JOIN movie_plot_keyword mpk ON m.movie_id=mpk.movie_id
JOIN plot_keyword pk ON pk.plot_keyword_id=mpk.plot_keyword_id ORDER BY movie_title;
SELECT * FROM MovieWithPlotkeyword;
```

movie_title	plot_keyword
#Horror	girl
#Horror	internet
#Horror	bullying
#Horror	cyberbullying
#Horror	throat slitting
02:13	death
02:13	murder
02:13	serial killer
02:13	forensic
02:13	profiler
10 Cloverfield Lane	alien
10 Cloverfield Lane	kidnapping
10 Cloverfield Lane	car crash
10 Cloverfield Lane	bunker

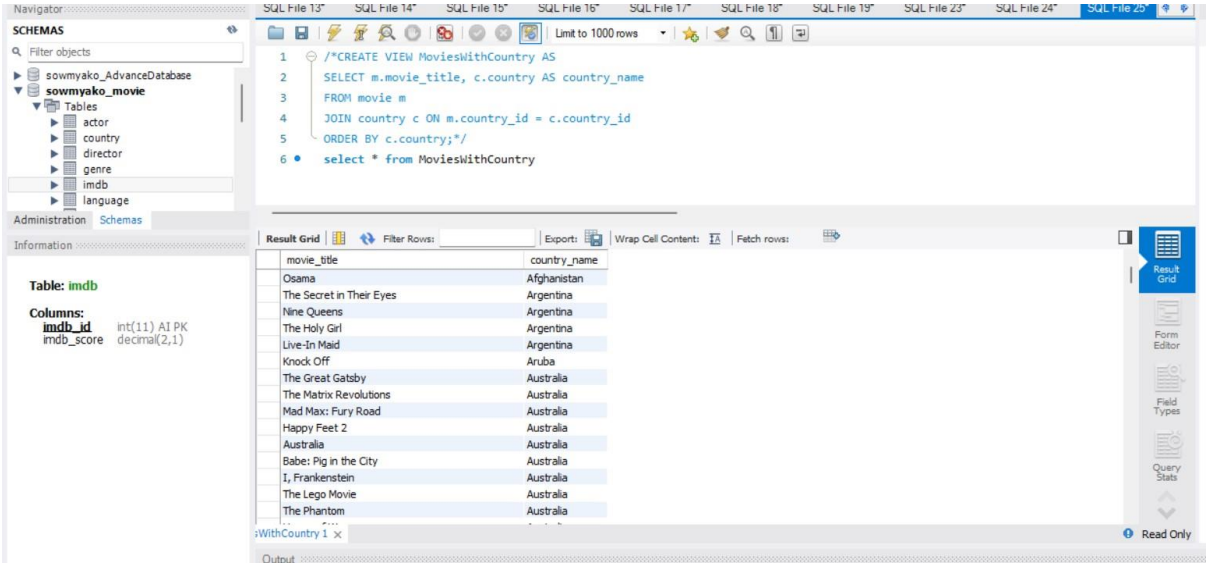
5. **View:** MoviesWithLanguage



```
/*CREATE VIEW MoviesWithLanguage AS
SELECT m.movie_title, l.language AS language_name
FROM movie m
JOIN language l ON m.language_id = l.language_id
ORDER BY l.language;*/
select * from MoviesWithLanguage
```

movie_title	language_name
The Interpreter	Aboriginal
Rabbit-Proof Fence	Aboriginal
Valley of the Wolves: Iraq	Arabic
The Square	Arabic
Caramel	Arabic
Ajami	Arabic
The Brain That Sings	Arabic
The Passion of the Christ	Aramaic
In the Land of Blood and Honey	Bosnian
Ip Man 3	Cantonese
Kung Fu Killer	Cantonese
Kung Fu Hustle	Cantonese
CJ7	Cantonese
2046	Cantonese
Shaolin Soccer	Cantonese

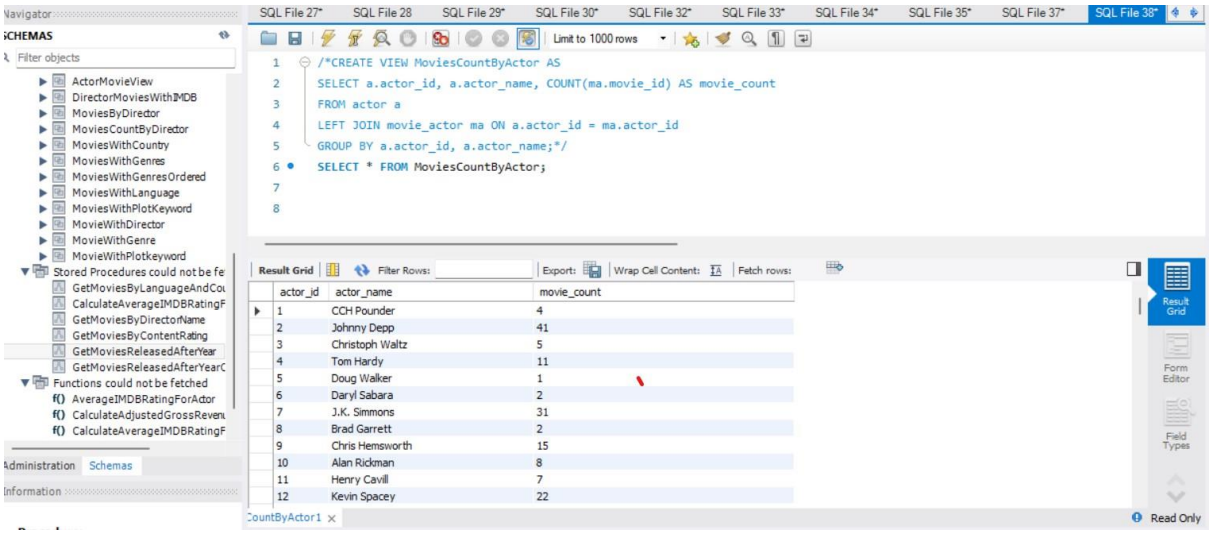
6. **View:** MoviesWithCountry



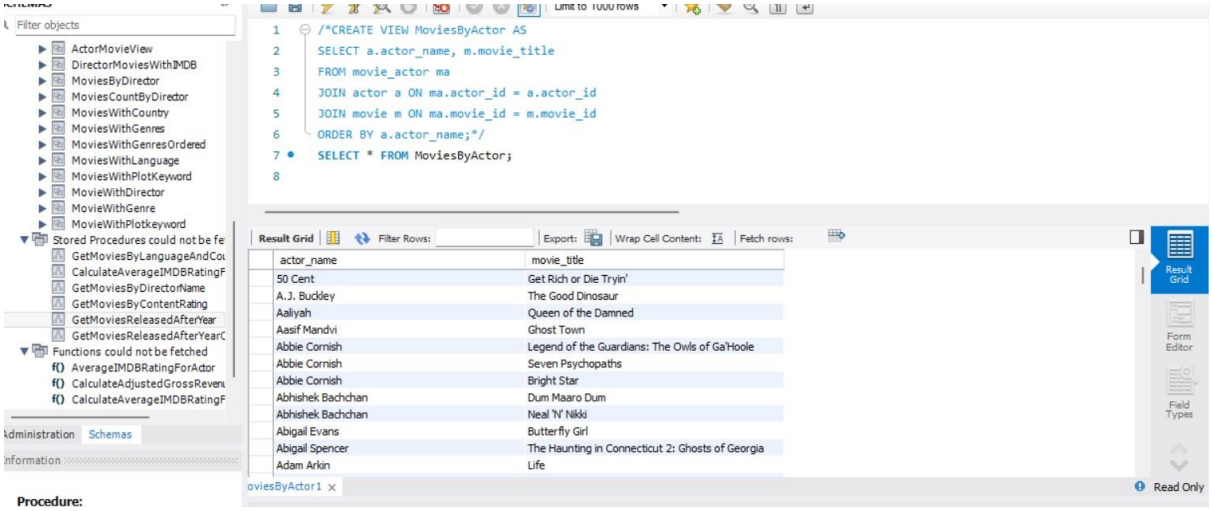
```
/*CREATE VIEW MoviesWithCountry AS
SELECT m.movie_title, c.country AS country_name
FROM movie m
JOIN country c ON m.country_id = c.country_id
ORDER BY c.country;*/
select * from MoviesWithCountry
```

movie_title	country_name
Osama	Afghanistan
The Secret in Their Eyes	Argentina
Nine Queens	Argentina
The Holy Girl	Argentina
Live-In Maid	Argentina
Knock Off	Aruba
The Great Gatsby	Australia
The Matrix Revolutions	Australia
Mad Max: Fury Road	Australia
Happy Feet 2	Australia
Australia	Australia
Babe: Pig in the City	Australia
I, Frankenstein	Australia
The Lego Movie	Australia
The Phantom	Australia

7. View: **MoviesCountByActor**

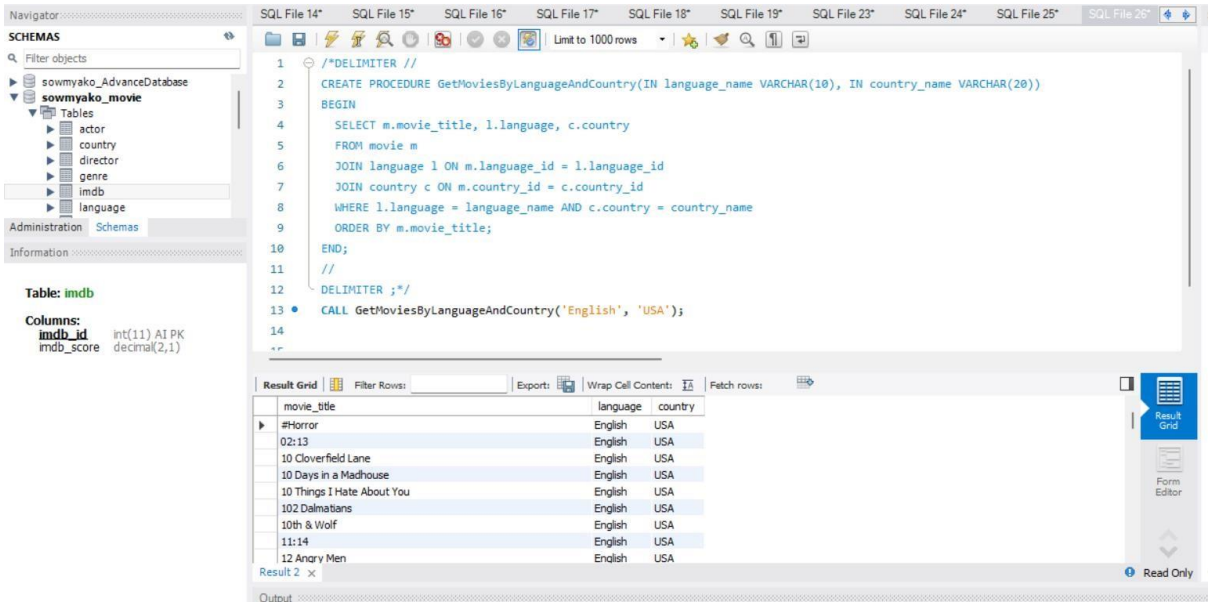


8. View: **MoviesByActor**

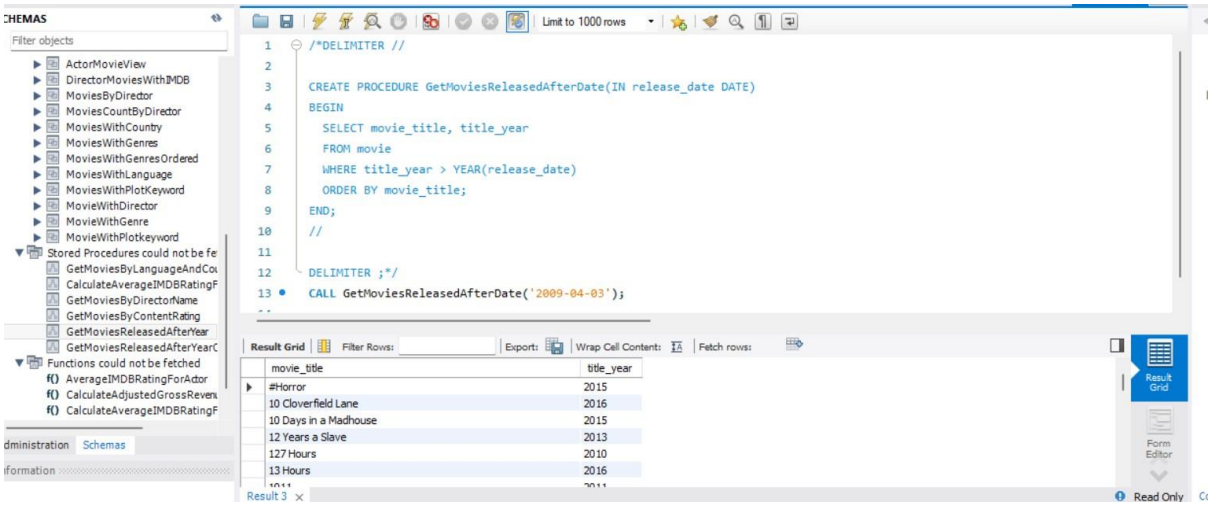


Procedure:

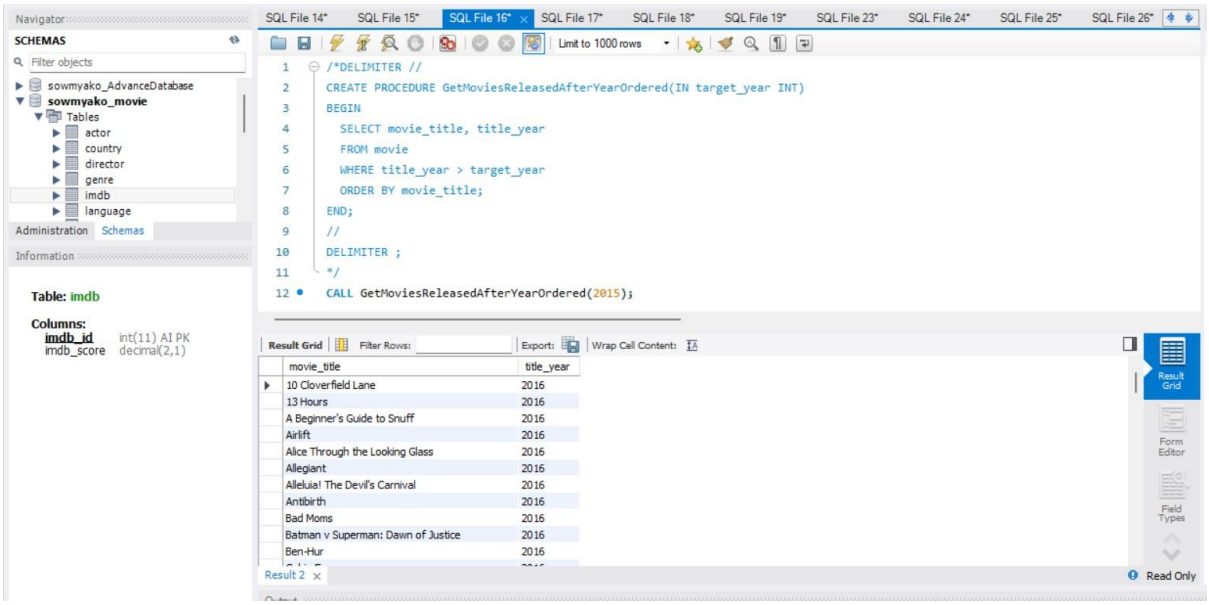
Stored Procedure 1: GetMoviesByLanguageAndCountry



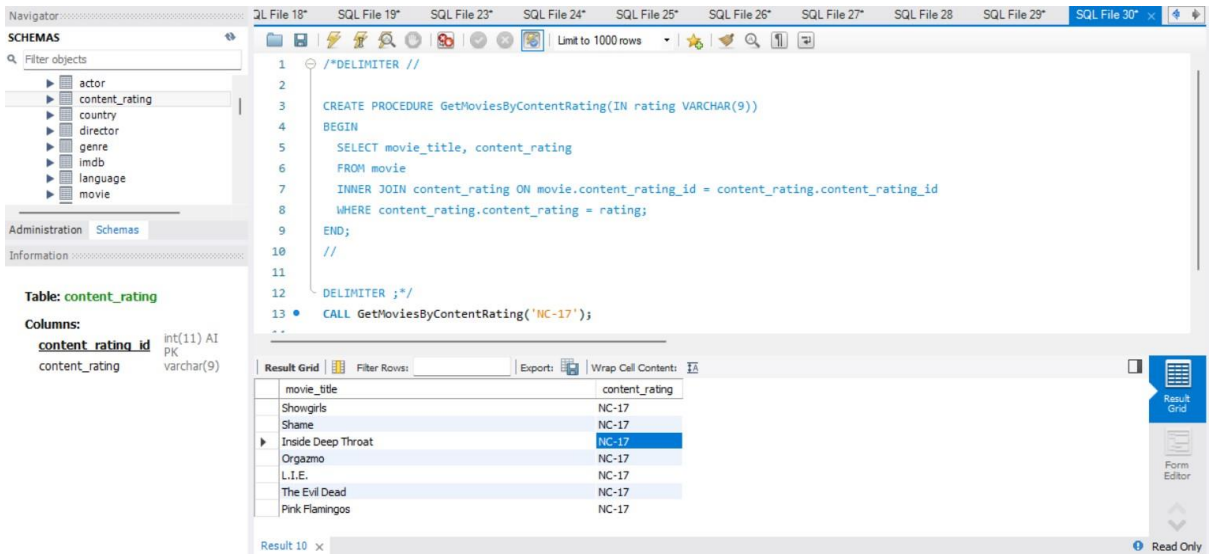
Stored Procedure 2: GetMoviesReleasedAfterDate



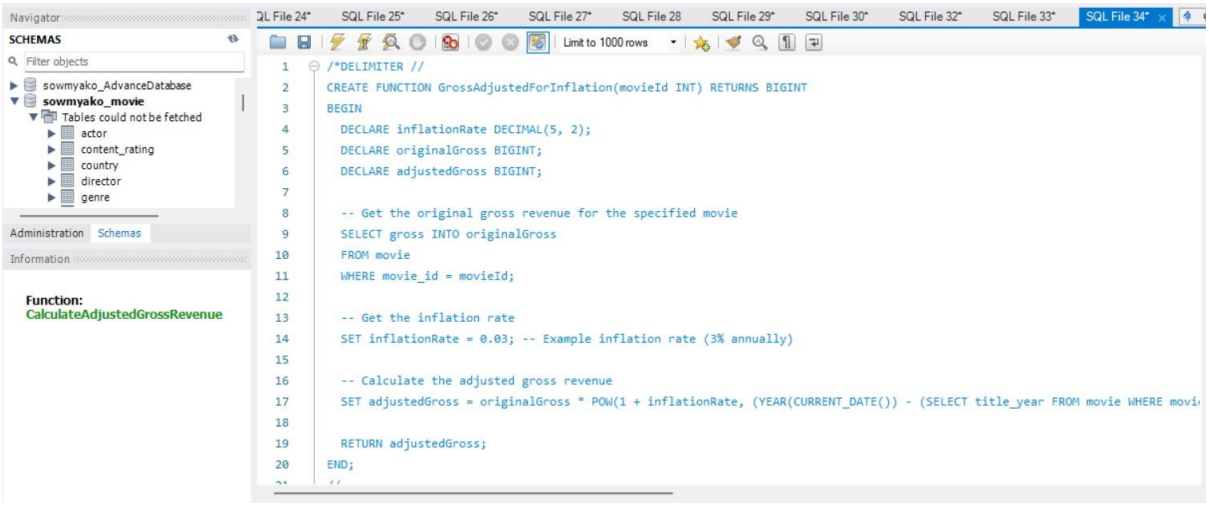
Stored Procedure 3: GetMoviesReleasedAfterYearOrdered

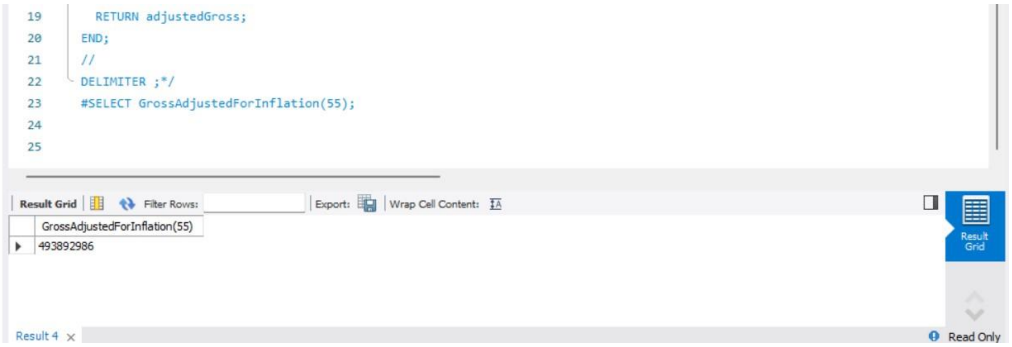


Stored Procedure 4: GetMoviesByContentRating

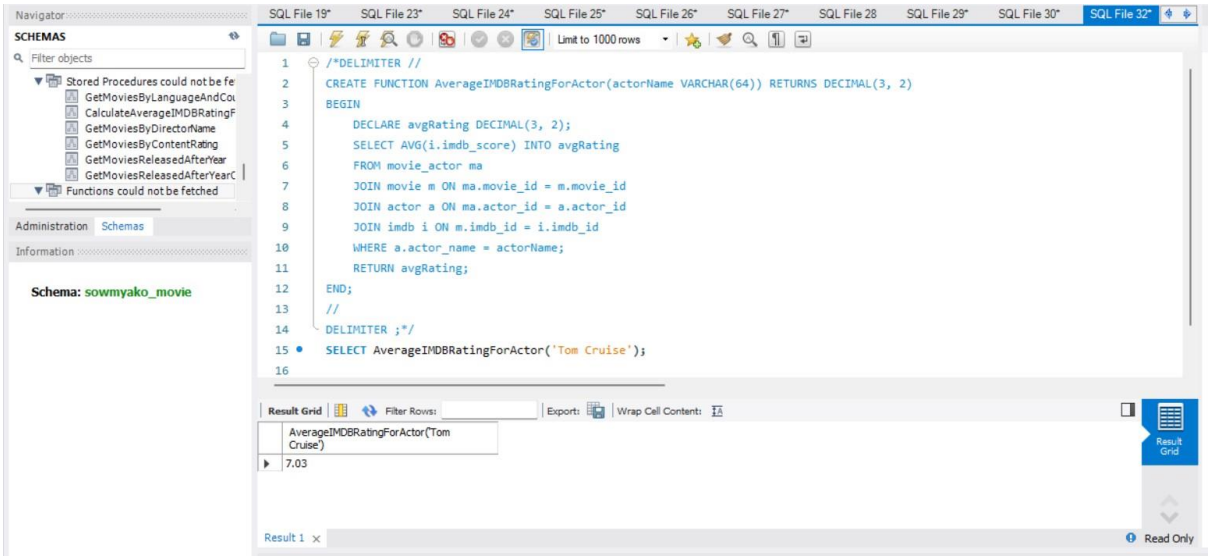


Function 1: GrossAdjustedForInflation

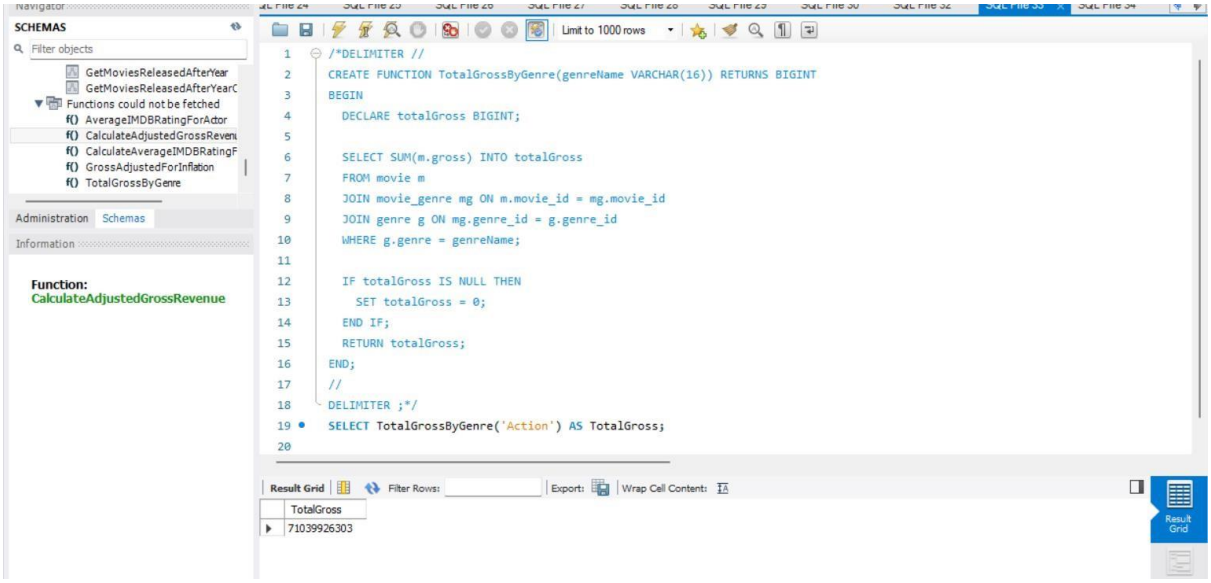




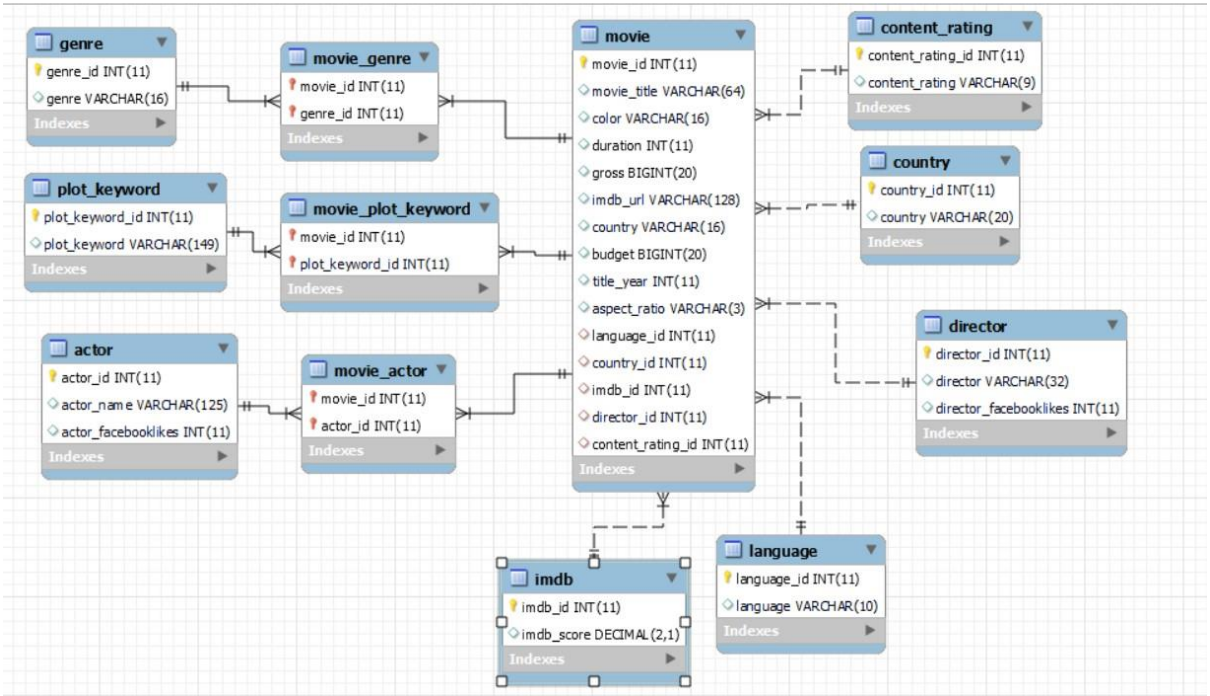
Function 2: AverageIMDBRatingForActor



Function 3: TotalGrossByGenre



Reverse Engineer



In my initial ERD diagram, I considered the possibility of an intersection table (movie_director) to handle the relationship between movies and directors. However, upon closer examination of the dataset, it became clear that each movie is directed by a single director, and one director can direct multiple movies. Therefore, I didn't create the intersection table (movie_director).

The relationship between the "movie" and "director" tables in the database is a Many-to-one (M:1) relationship (Movie to director). This means that each movie is associated with one director, and a single director can be linked to many movies.

In my initial Entity-Relationship Diagram (ERD), I initially depicted the relationship between movies and IMDB scores as a one-to-one (1:1) relationship. However, after a more thorough analysis of the dataset, I realized that each movie has a single IMDB score, but the same IMDB score can be associated with multiple movies. Therefore, I adjusted the relationship to a Many-to-one (M:1) relationship between movies and IMDB scores(Movie to IMDB) to accurately reflect the data structure. This change was necessary to maintain data integrity and accurately represent the dataset.

The other relationships in the database remained consistent with my initial ERD design, as they accurately represent the connections between tables based on the dataset's structure.