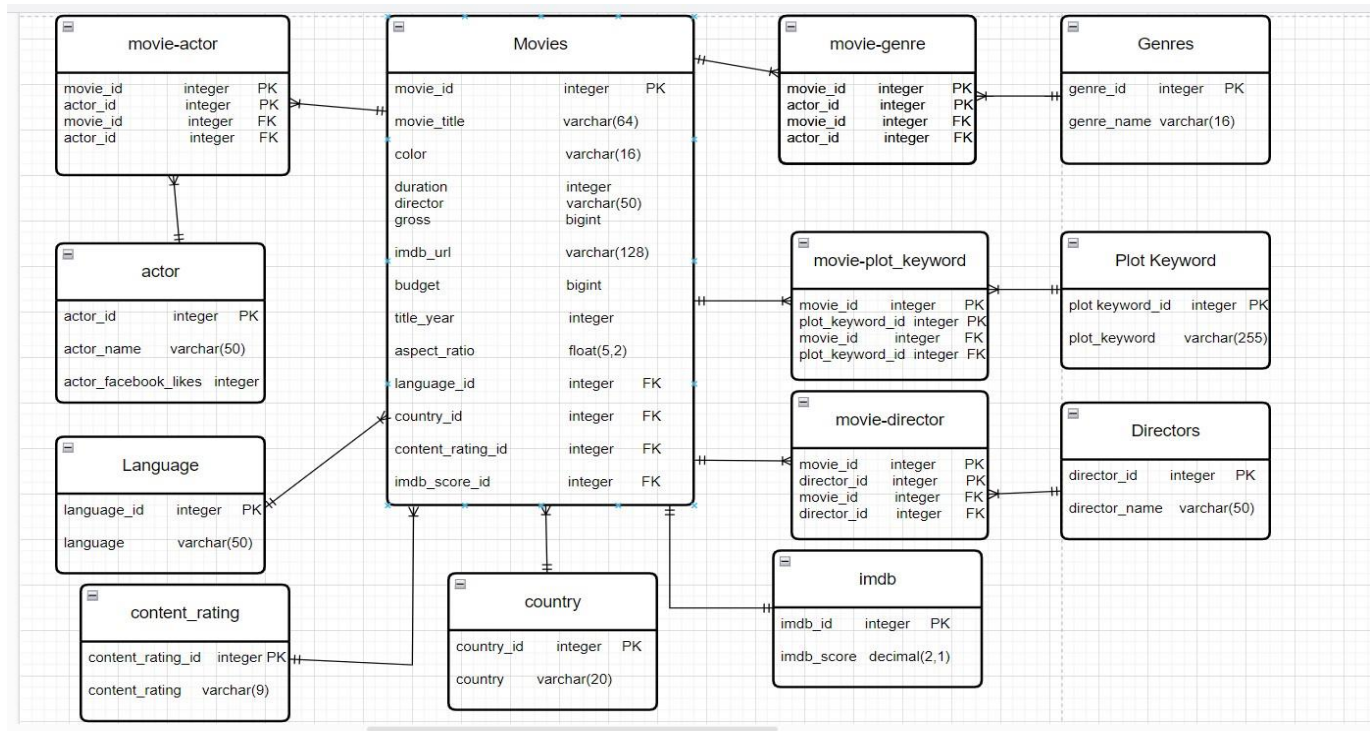


Data Source: <https://www.kaggle.com/code/saurav9786/imdb-score-prediction-for-movies>.

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_unf
 - 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_unf
 - 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

movie_id	color	director_name	num_critc_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_facebook_likes
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

movies_unf 3 x

Read Only

1. View: MoviesWithGenreLanguage

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' pane is open, showing a tree view of the database objects. The 'Stored Procedures' folder is expanded, showing a list of procedures. The 'MoviesWithGenreLanguage' view is selected. The main pane displays the SQL script for creating the view:

```
1 /*CREATE VIEW MoviesWithGenreLanguage AS
2 SELECT m.movie_id, m.movie_title, g.genre, l.language
3 FROM movie m
4 JOIN movie_genre mg ON m.movie_id = mg.movie_id
5 JOIN genre g ON mg.genre_id = g.genre_id
6 JOIN language l ON m.language_id = l.language_id;*/
7 SELECT * FROM MoviesWithGenreLanguage
8
```

The 'Result Grid' pane shows the data returned by the view:

movie_id	movie_title	genre	language
1	Avatar	Action	English
1	Avatar	Adventure	English
1	Avatar	Fantasy	English
1	Avatar	Sci-Fi	English
2	Pirates of the Caribbean: At World's End	Action	English
2	Pirates of the Caribbean: At World's End	Adventure	English
2	Pirates of the Caribbean: At World's End	Fantasy	English
3	Spectre	Action	English
3	Spectre	Adventure	English
3	Spectre	Thriller	English
4	The Dark Knight Rises	Action	English
4	The Dark Knight Rises	Thriller	English
6	John Carter	Action	English

2. View: MovieWithPlotKeywordAndContentRating

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' pane is open, showing a tree view of the database objects. The 'Stored Procedures' folder is expanded, showing a list of procedures. The 'MovieWithPlotKeywordAndContentRating' view is selected. The main pane displays the SQL script for creating the view:

```
1 /*CREATE VIEW MoviesWithPlotKeywordsAndContentRating AS
2 SELECT
3     m.movie_id,
4     m.movie_title,
5     pk.plot_keyword,
6     cr.content_rating
7 FROM
8     movie m
9 JOIN movie_plot_keyword mpk ON m.movie_id = mpk.movie_id
10 JOIN plot_keyword pk ON mpk.plot_keyword_id = pk.plot_keyword_id
11 JOIN content_rating cr ON m.content_rating_id = cr.content_rating_id;*/
12 SELECT * FROM MoviesWithPlotKeywordsAndContentRating;
13
```

The 'Result Grid' pane shows the data returned by the view:

movie_id	movie_title	plot_keyword	content_rating
1	Avatar	avatar	PG-13
1	Avatar	future	PG-13
1	Avatar	marine	PG-13
1	Avatar	native	PG-13
1	Avatar	paraplegic	PG-13
2	Pirates of the Caribbean: At World's End	goddess	PG-13
2	Pirates of the Caribbean: At World's End	marriage ceremony	PG-13
7	Director of the Caribbean: At World's End	pirate	PG-13

3. View: MovieWithGenre

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' pane is open, showing a tree view of the database objects. The 'sowmyako_movie' database is selected. The 'Tables' folder is expanded, showing a list of tables. The 'MovieWithGenre' view is selected. The main pane displays the SQL script for creating the view:

```
1 /*DROP VIEW IF EXISTS MovieWithGenre;
2 CREATE VIEW MovieWithGenre AS
3 SELECT movie_title, genre
4 FROM movie m
5 JOIN movie_genre mg ON m.movie_id=mg.movie_id
6 JOIN genre g ON g.genre_id=mg.genre_id ORDER BY movie_title;
7 */
8 SELECT * FROM MovieWithGenre

```

The 'Result Grid' pane shows the data returned by the view:

movie_title	genre
#Horror	Drama
#Horror	Mystery
#Horror	Horror
#Horror	Thriller
02:13	Horror
02:13	Thriller
10 Cloverfield Lane	Drama
10 Cloverfield Lane	Mystery
10 Cloverfield Lane	Sci-Fi
10 Cloverfield Lane	Horror
10 Cloverfield Lane	Thriller
10 Days in a Madhouse	Drama
10 Things I Hate About You	Drama
10 Things I Hate About You	Comedy

4. View: MovieWithPlotkeyword

The screenshot shows the SQL Studio interface. On the left, the 'Schemas' pane displays the 'sowmyako_AdvanceDatabase' and 'sowmyako_movie' schemas. The 'sowmyako_movie' schema contains tables: actor, director, genre, movie, and movie_actor. The main editor shows the following SQL script:

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

The 'Result Grid' at the bottom displays the data from the view:

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

The screenshot shows the SQL Studio interface. On the left, the 'Schemas' pane displays the 'sowmyako_AdvanceDatabase' and 'sowmyako_movie' schemas. The 'sowmyako_movie' schema contains tables: actor, country, director, genre, imdb, and language. The main editor shows the following SQL script:

```
1 /*CREATE VIEW MoviesWithLanguage AS
2 SELECT m.movie_title, l.language AS language_name
3 FROM movie m
4 JOIN language l ON m.language_id = l.language_id
5 ORDER BY l.language;*/
6 select * from MoviesWithLanguage
7
```

The 'Result Grid' at the bottom displays the data from the view:

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

The screenshot shows the SQL Studio interface. On the left, the 'Schemas' pane displays the 'sowmyako_AdvanceDatabase' and 'sowmyako_movie' schemas. The 'sowmyako_movie' schema contains tables: actor, country, director, genre, imdb, and language. The main editor shows the following SQL script:

```
1 /*CREATE VIEW MoviesWithCountry AS
2 SELECT m.movie_title, c.country AS country_name
3 FROM movie m
4 JOIN country c ON m.country_id = c.country_id
5 ORDER BY c.country;*/
6 select * from MoviesWithCountry
7
```

The 'Result Grid' at the bottom displays the data from the view:

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

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' pane is open, showing a list of objects including 'MoviesCountByActor'. The main pane displays the SQL script for creating the view:

```
1 /*CREATE VIEW MoviesCountByActor AS
2 SELECT a.actor_id, a.actor_name, COUNT(ma.movie_id) AS movie_count
3 FROM actor a
4 LEFT JOIN movie_actor ma ON a.actor_id = ma.actor_id
5 GROUP BY a.actor_id, a.actor_name;*/
6 SELECT * FROM MoviesCountByActor;
```

The 'Result Grid' shows the data returned by the view:

actor_id	actor_name	movie_count
1	CCH Pounder	4
2	Johnny Depp	41
3	Christoph Waltz	5
4	Tom Hardy	11
5	Doug Walker	1
6	Daryl Sabara	2
7	J.K. Simmons	31
8	Brad Garrett	2
9	Chris Hemsworth	15
10	Alan Rickman	8
11	Henry Cavill	7
12	Kevin Spacey	22

8. View: MoviesByActor

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' pane is open, showing a list of objects including 'MoviesByActor'. The main pane displays the SQL script for creating the view:

```
1 /*CREATE VIEW MoviesByActor AS
2 SELECT a.actor_name, m.movie_title
3 FROM movie_actor ma
4 JOIN actor a ON ma.actor_id = a.actor_id
5 JOIN movie m ON ma.movie_id = m.movie_id
6 ORDER BY a.actor_name;*/
7 SELECT * FROM MoviesByActor;
```

The 'Result Grid' shows the data returned by the view:

actor_name	movie_title
50 Cent	Get Rich or Die Tryin'
A.J. Budley	The Good Dinosaur
Aaliyah	Queen of the Damned
Aasif Mandvi	Ghost Town
Abbie Cornish	Legend of the Guardians: The Owls of Ga'Hoole
Abbie Cornish	Seven Psychopaths
Abbie Cornish	Bright Star
Abhishek Bachchan	Dum Maaro Dum
Abhishek Bachchan	Neal 'N' Nikki
Abigail Evans	Butterfly Girl
Abigail Spencer	The Haunting in Connecticut 2: Ghosts of Georgia
Adam Arkin	Life

Stored Procedure 1: GetMoviesByLanguageAndCountry

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' pane displays the 'sowmyako_AdvanceDatabase' with a table named 'imdb'. The table's columns are 'imdb_id' (int(11) AI PK) and 'imdb_score' (decimal(2,1)). The main pane shows the SQL script for the stored procedure 'GetMoviesByLanguageAndCountry'. The script is as follows:

```
1  /*DELIMITER */
2  CREATE PROCEDURE GetMoviesByLanguageAndCountry(IN language_name VARCHAR(10), IN country_name VARCHAR(20))
3  BEGIN
4      SELECT m.movie_title, l.language, c.country
5      FROM movie m
6      JOIN language l ON m.language_id = l.language_id
7      JOIN country c ON m.country_id = c.country_id
8      WHERE l.language = language_name AND c.country = country_name
9      ORDER BY m.movie_title;
10 END;
11 //
12 DELIMITER ;*/
13 CALL GetMoviesByLanguageAndCountry('English', 'USA');
```

The 'Result Grid' at the bottom displays the following data:

movie_title	language	country
#Horror	English	USA
02:13	English	USA
10 Cloverfield Lane	English	USA
10 Days in a Madhouse	English	USA
10 Things I Hate About You	English	USA
102 Dalmatians	English	USA
10th & Wolf	English	USA
11:14	English	USA
12 Angry Men	English	USA

Stored Procedure 2: GetMoviesReleasedAfterDate

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' pane displays the 'sowmyako_AdvanceDatabase' with a table named 'imdb'. The table's columns are 'imdb_id' (int(11) AI PK) and 'imdb_score' (decimal(2,1)). The main pane shows the SQL script for the stored procedure 'GetMoviesReleasedAfterDate'. The script is as follows:

```
1  /*DELIMITER */
2  CREATE PROCEDURE GetMoviesReleasedAfterDate(IN release_date DATE)
3  BEGIN
4      SELECT movie_title, title_year
5      FROM movie
6      WHERE title_year > YEAR(release_date)
7      ORDER BY movie_title;
8  END;
9  //
10 DELIMITER ;*/
11 CALL GetMoviesReleasedAfterDate('2009-04-03');
```

The 'Result Grid' at the bottom displays the following data:

movie_title	title_year
#Horror	2015
10 Cloverfield Lane	2016
10 Days in a Madhouse	2015
12 Years a Slave	2013
127 Hours	2010
13 Hours	2016

Stored Procedure 3: GetMoviesReleasedAfterYearOrdered

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'SCHEMAS' tree with the 'movie' table selected. The right pane shows the SQL Editor with the following code:

```
1  /*DELIMITER */
2  CREATE PROCEDURE GetMoviesReleasedAfterYearOrdered(IN target_year INT)
3  BEGIN
4      SELECT movie_title, title_year
5      FROM movie
6      WHERE title_year > target_year
7      ORDER BY movie_title;
8  END;
9  //
10 DELIMITER ;
11 */
12 CALL GetMoviesReleasedAfterYearOrdered(2015);
```

The 'Result Grid' at the bottom displays the following data:

movie_title	title_year
10 Cloverfield Lane	2016
13 Hours	2016
A Beginner's Guide to Snuff	2016
Airlift	2016
Alice Through the Looking Glass	2016
Allegiant	2016
Alleluia! The Devil's Carnival	2016
Antibirth	2016
Bad Moms	2016
Batman v Superman: Dawn of Justice	2016
Ben-Hur	2016

Stored Procedure 4: GetMoviesByContentRating

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'SCHEMAS' tree with the 'content_rating' table selected. The right pane shows the SQL Editor with the following code:

```
1  /*DELIMITER */
2  CREATE PROCEDURE GetMoviesByContentRating(IN rating VARCHAR(9))
3  BEGIN
4      SELECT movie_title, content_rating
5      FROM movie
6      INNER JOIN content_rating ON movie.content_rating_id = content_rating.content_rating_id
7      WHERE content_rating.content_rating = rating;
8  END;
9  //
10 DELIMITER ;
11 */
12 CALL GetMoviesByContentRating('NC-17');
```

The 'Result Grid' at the bottom displays the following data:

movie_title	content_rating
Showgirls	NC-17
Shame	NC-17
Inside Deep Throat	NC-17
Orgazmo	NC-17
L.I.E.	NC-17
The Evil Dead	NC-17
Pink Flamingos	NC-17

Function 1: GrossAdjustedForInflation

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' pane shows the 'sowmyako_movie' database with tables like 'actor', 'content_rating', 'country', 'director', and 'genre'. The 'Administration' tab is selected, and the 'Function: CalculateAdjustedGrossRevenue' is highlighted. The main pane shows the SQL script for creating the function 'GrossAdjustedForInflation'.

```
1  /*DELIMITER //
2  CREATE FUNCTION GrossAdjustedForInflation(movieId INT) RETURNS BIGINT
3  BEGIN
4      DECLARE inflationRate DECIMAL(5, 2);
5      DECLARE originalGross BIGINT;
6      DECLARE adjustedGross BIGINT;
7
8      -- Get the original gross revenue for the specified movie
9      SELECT gross INTO originalGross
10     FROM movie
11     WHERE movie_id = movieId;
12
13     -- Get the inflation rate
14     SET inflationRate = 0.03; -- Example inflation rate (3% annually)
15
16     -- Calculate the adjusted gross revenue
17     SET adjustedGross = originalGross * POW(1 + inflationRate, (YEAR(CURRENT_DATE()) - (SELECT title_year FROM movie WHERE movie_id = movieId)));
18
19     RETURN adjustedGross;
20 END;
21 //
22 DELIMITER ;*/
23 #SELECT GrossAdjustedForInflation(55);
24
25
```

The 'Result Grid' at the bottom shows the output of the function call:

GrossAdjustedForInflation(55)
493892986

Function 2: AverageIMDBRatingForActor

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' pane shows the 'sowmyako_movie' database with tables like 'actor', 'content_rating', 'country', 'director', and 'genre'. The 'Administration' tab is selected, and the 'Function: AverageIMDBRatingForActor' is highlighted. The main pane shows the SQL script for creating the function 'AverageIMDBRatingForActor'.

```
1  /*DELIMITER //
2  CREATE FUNCTION AverageIMDBRatingForActor(actorName VARCHAR(64)) RETURNS DECIMAL(3, 2)
3  BEGIN
4      DECLARE avgRating DECIMAL(3, 2);
5      SELECT AVG(i.imdb_score) INTO avgRating
6      FROM movie_actor ma
7      JOIN movie m ON ma.movie_id = m.movie_id
8      JOIN actor a ON ma.actor_id = a.actor_id
9      JOIN imdb i ON m.imdb_id = i.imdb_id
10     WHERE a.actor_name = actorName;
11     RETURN avgRating;
12 END;
13 //
14 DELIMITER ;*/
15 SELECT AverageIMDBRatingForActor('Tom Cruise');
16
```

The 'Result Grid' at the bottom shows the output of the function call:

AverageIMDBRatingForActor('Tom Cruise')
7.03

Function 3: TotalGrossByGenre

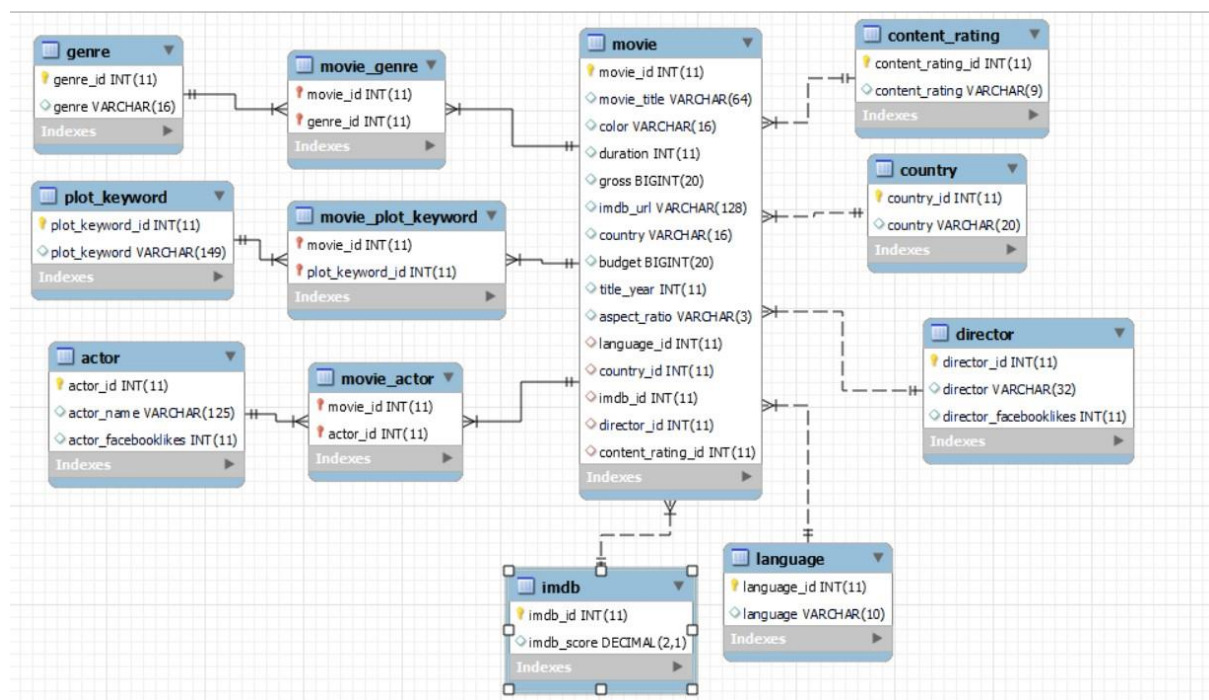
The screenshot shows a database IDE with a SQL editor and a results grid. The SQL editor contains the following code:

```
1  /*DELIMITER */
2  CREATE FUNCTION TotalGrossByGenre(genreName VARCHAR(16)) RETURNS BIGINT
3  BEGIN
4      DECLARE totalGross BIGINT;
5
6      SELECT SUM(m.gross) INTO totalGross
7      FROM movie m
8      JOIN movie_genre mg ON m.movie_id = mg.movie_id
9      JOIN genre g ON mg.genre_id = g.genre_id
10     WHERE g.genre = genreName;
11
12     IF totalGross IS NULL THEN
13         SET totalGross = 0;
14     END IF;
15     RETURN totalGross;
16 END;
17 //
18 DELIMITER ;*/
19 • SELECT TotalGrossByGenre('Action') AS TotalGross;
20
```

The results grid shows the output of the function call:

TotalGross
71039926303

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.