Documentation

- What is the dataset?

My dataset is about movies. It contains basic detailed information about movies (one artificial id, name, release year, the genre, country, the main actor, the production company, the number of viewers, the box office), the details of the main actor (one artificial id, the name, the home country, number of movies filmed, gender), the information about the genre (one artificial id , the genre name, one example movie, number of movies in the genre), information of the production company (one artificial id, company name, number of movies produced, the origin country), with other information about the country (one artificial id, the country name, population, abbreviation, the capital city, continent name).

- The sources of the data:

My information comes from web searched websites: Worldometers, The spread sheet guru, The numbers, Washington's Top News. The rest of the information is generated from AI tools like ChatGPT.

Country:

Worldometers: https://www.worldometers.info/geography/alphabetical-list-of-countries/

Thespreadsheetguru: https://www.thespreadsheetguru.com/list-countries-capitals-abbreviations/

Genre:

The numbers: https://www.the-numbers.com/market/genres

Washington's Top News: https://wtop.com/wp-content/uploads/2018/09/Best-Movies-in-Every-Genre.pdf

Company:

ChatGPT: https://chat.openai.com/c/2129cd35-5d41-43bb-8b20-ba69a9494115

Actor:

ChatGPT: https://chat.openai.com/c/f4ec2311-5b5b-4caa-83d2-9119bce0ec98

Movie:

ChatGPT: https://chat.openai.com/c/9a661af1-032c-4fe0-8a65-412c43108e82

- The license information:

I can’t find the license information on the websites. But ChatGPT said “As an AI language model, I don't own the generated data or have the authority to grant licenses for it. The generated table is based on fictional movie data created for the purpose of this interaction. You are free to use, modify, and distribute the generated data for personal or commercial purposes without any restrictions.”. My data is from Worldometers, Thespreadsheetguru, The numbers, Washington's Top News and ChatGPT. I am using the data educationally and you are not publishing anything to the public.

- Provide details of your database:

There are 5 tables:

Country: there are 6 attributes and 195 rows.

Genre: there are 4 attributes 15 rows.

Company: there are 4 attributes 53 rows.

Actor: there are 5 attributes 80 rows.

Movie: there are 9 attributes 137 rows.

- Business rules

1. One country can have one or more actors; one actor must have exactly one country.

This can be achieved with foreign keys. We have a foreign key actor.home_country on the actor table referencing country.country_name. Since we want one country to have one or more actors and one actor must have exactly one home country, and the home country must be found in the country table. We can implement it using foreign keys. Therefore, we can set the home country value in the actor table as the foreign key which reference to country name in the country table. The implementation is like: FOREIGN KEY(home_country) REFERENCES country(country_name).

2. One genre can have one or more movies; one movie must have exactly one genre.

This can be achieved with foreign keys. We have a foreign key movie.movie_genre on the movie table referencing genre.genre_name. Since we want one genre can have one or more movies and one movie must have exactly one genre, plus the movie genre must be found in the genre table. We can implement it using foreign keys. Therefore, we can set

the movie genre value in the movie table as the foreign key which reference to genre name in the genre table. The implementation is like: FOREIGN KEY(movie_genre) REFERENCES genre(genre_name).

3. One company can have one or more movies; one movie can have exactly one company.

This can be achieved with foreign keys. We have a foreign key movie. production_company on the movie table referencing company.company_name. Since we want one company to have one or more movies and one movie must have exactly one company, plus the movie company must be found in the company table. We can implement it using foreign keys. Therefore, we can set the production company value in the movie table as the foreign key which reference to company name in the company table. The implementation is like: FOREIGN KEY (production_company) REFERENCES company(company_name).

- Queries

1.

Select the name, year, viewers and box-office of the movie, calculated money earned per person by dividing box-office by viewers. Where the movie is filmed later than year 2000.

2.

Select the actor’s name, the home country of actor from actor table, the continent name from country table, join the tables on home country and country name. Where the country is in north America.

Select the actor’s name, the home country, and the continent name of actors from north America.

3.

Select the number of actors from the actor table, the continent name from country table, join the tables on home country and country name. Group them by continent.

Select the number of actors from each continent.

4.

Select 50 movie name, movie year, movie viewers, box-office, actor name, genre name from the 3 most popular movie genres (most number of movies), where the movie is filmed later than year 2000.

5.

Create a view of the top 50 most money-earned per person on their movie name, movie year, money per person, actor name, actor home country and continent name. where the movie is filmed later than year 2000. Then showed the result.

Updated the movie table and changed all these 50 movies filmed year to 1999. Then showed the result to see the difference.

- Stored procedures

1.

It is used to create a new value in the movie table. It has 6 parameters: @new_movie_name is the name of the new movie; @new_movie_year is the year of the new movie; @new_movie_genre is the genre of the new movie; @new_movie_country is the country of the new movie; @new_movie_actor is the main actor of the new movie; @new_movie_comp is the production company of the new movie.

Every attribute except year is checked in this procedure: the name of the movie should not be null, and the other attributes should be found in other tables since they are foreign keys. If the checks did not pass, we will receive error messages: "Enter the name of the movie!" or "Not valid!".

Once all the checks pass, an insert should be made into the movie table, and the number of movies is added by 1 to the other tables specified by the attributes, then we receive a result shown.

There is no returned values for this stored procedure. A call example:

```
SET @new_movie_name := "Random good thing";

SET @new_movie_year := 2010;

SET @new_movie_genre := "Adventure";

SET @new_movie_country := "Afghanistan";

SET @new_movie_actor := "Tom Hanks";

SET @new_movie_comp := "Paramount Pictures";

CALL new_movie(@new_movie_name, @new_movie_year, @new_movie_genre, @new_movie_country,@new_movie_actor,@new_movie_comp);
```

CALL can be used with a variable or use values directly. In my SQL script I prepared two test cases, the first one works and the second one returns an error message.

2.

It is used to update the movie table with some regular behaviors. It has 4 parameters: @movie_title is the name of the movie updated; @num_viewers is the number of viewers of the movie; @box_office is the box office of the movie; @trash is a password to delete the movie in case of accident inputs.

This stored procedure first checks for the movie title. If the name of the movie is null or not in the table, we will receive error messages: "Enter the name of the movie!" or "Not valid!".

Once the checks pass, a delete should be made if the trash password is correct, and the number of movies is subtracting by 1 to the other tables specified by the attributes, then we receive a deleted result shown.

Else if the password is not correct, the stored procedure will check the num_viewers and box_office, if the data is not valid, we will receive an error message: “Please provide valid data”, else it will update the attributes and then we receive a updated result shown.

There is no returned values for this stored procedure. A call example:

```
SET @movie_title := "A Quiet Place Part II";

SET @num_viewers := 8000000;

SET @box_office := 200000000;

SET @trash := 0;

CALL movie_update(@movie_title,@num_viewers,@box_office,@trash);
```

CALL can be used with a variable or use values directly. In my SQL script I prepared three test cases, the first one works and the second one returns an error message. The third one is from stored procedure 1. Just to complete it.