**Introduction**

During the last few weeks we have had the opportunity to create a database modeling a collection of movies and their attributes such as actors, directors, genres and year released. Our tables were populated using the data from public IMDB tables. The primary user for our application is the administrator. The administrator is able to look up a movie by its name, director, writer, character name, actor/s, genre, or year released. The administrator’s job is to preserve the history of movies, add genres associated with a move, and updated information associated with a person such as a name change. The secondary users, or movie enthusiast, are able to use the application to retrieve information about specific movies. The application also has the programmed ability to search a person (actor/director/writer) and see a list of movies that are associated with them.

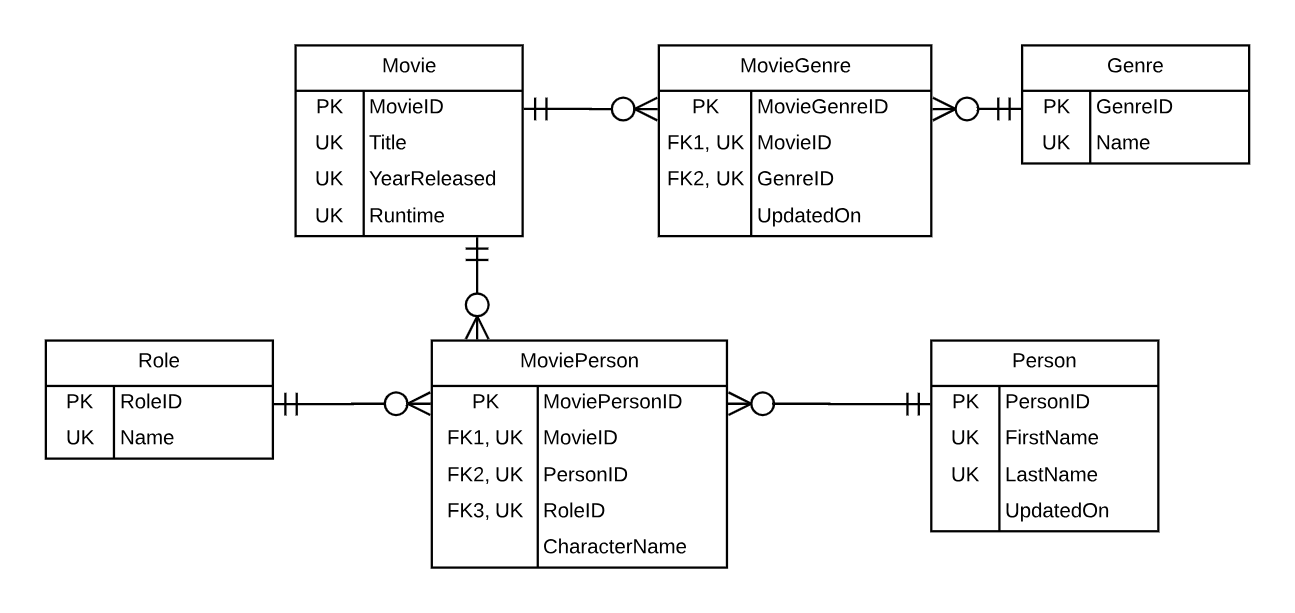
The database’s functions allow all consumers to expand their knowledge about movie history using specific information about movies and their actors. Through the database design, attached below, the programmable SQL is able to compile information on whichever subject our user’s wants.

Additional functionality of our database includes seeing how many movies were released in a given genre in a certain year, the cast size for a movie, and seeing how many movies a certain person has appeared in. When a new movie is released, the administrator can insert the information about the movie into their respective tables. For example, they can relate the people that worked on that movie and their role while they worked on the movie.

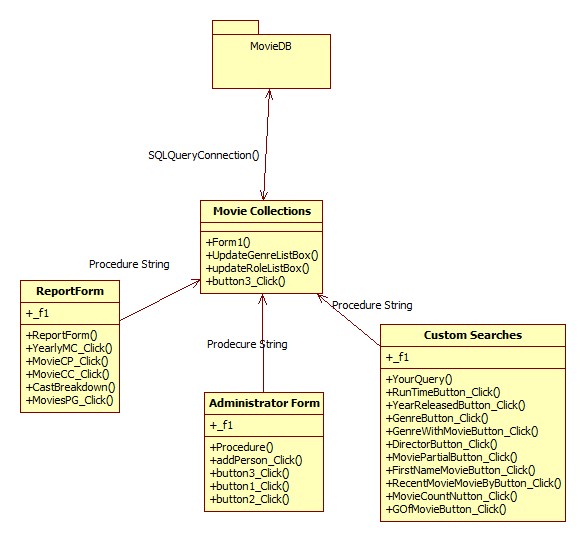
**Technical Description**

The front-end of our project was built using visual studio and C# code. Using multiple forms including Custom Searches, Administrator, and Reports each programmed with specific tasks. C# was chosen because of its ability to connect and query easily to an SQL Server. The database was constructed in My SQL Server Management Studio 2017 under the Schema MovieDB in the cis560\_team10 folder. Initially, we designed our database relation and keys in LucidChart and imported our data from IMDB. The UML diagram for the C# application was made in WhiteStarUML.

**Database Design**

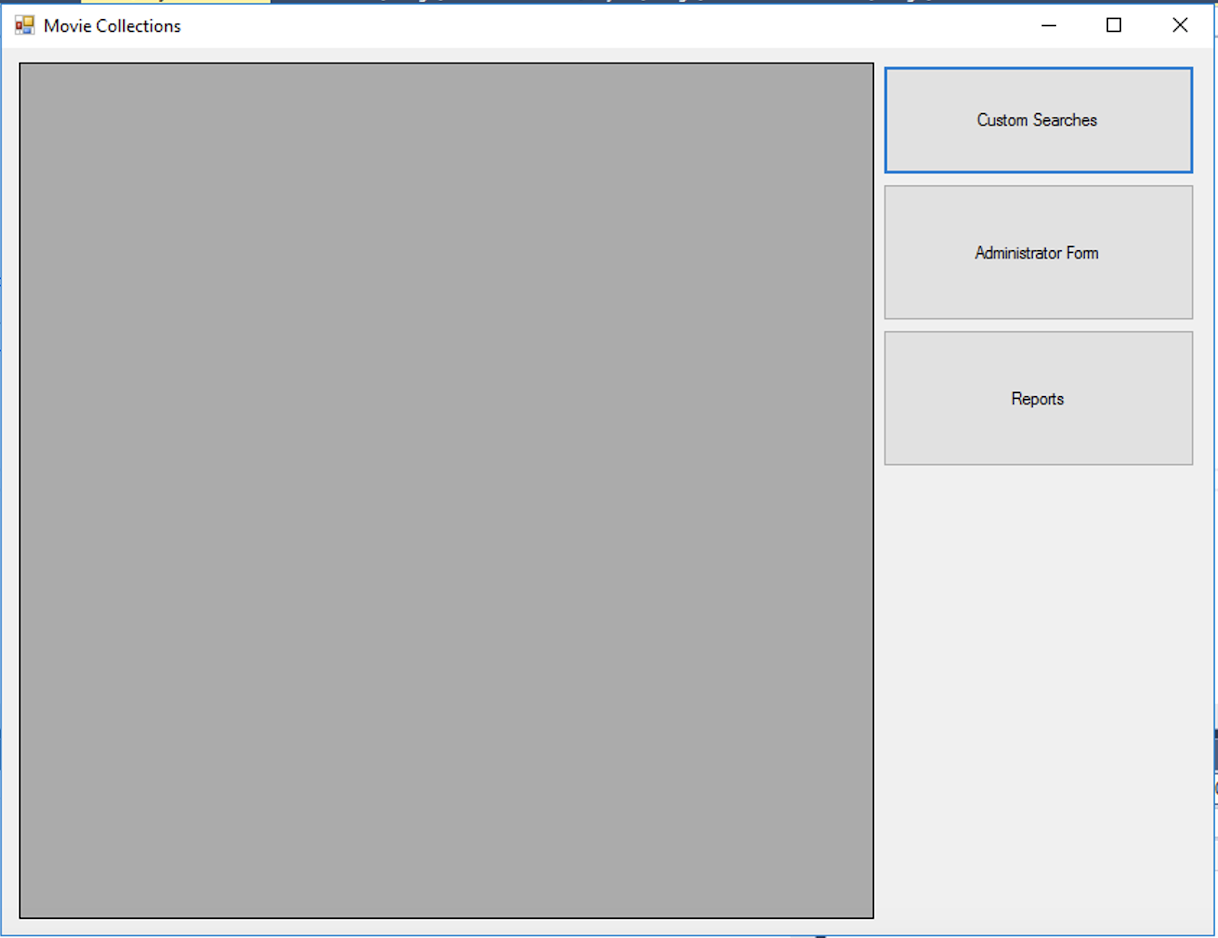


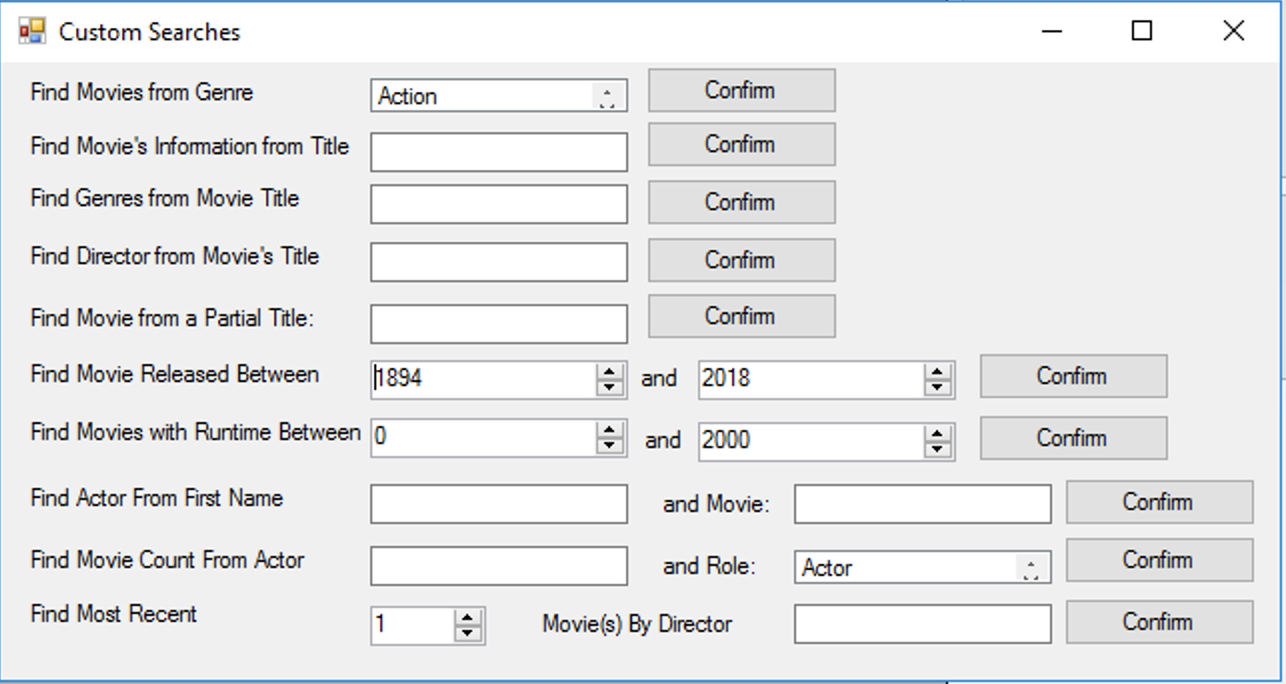
**System Design**

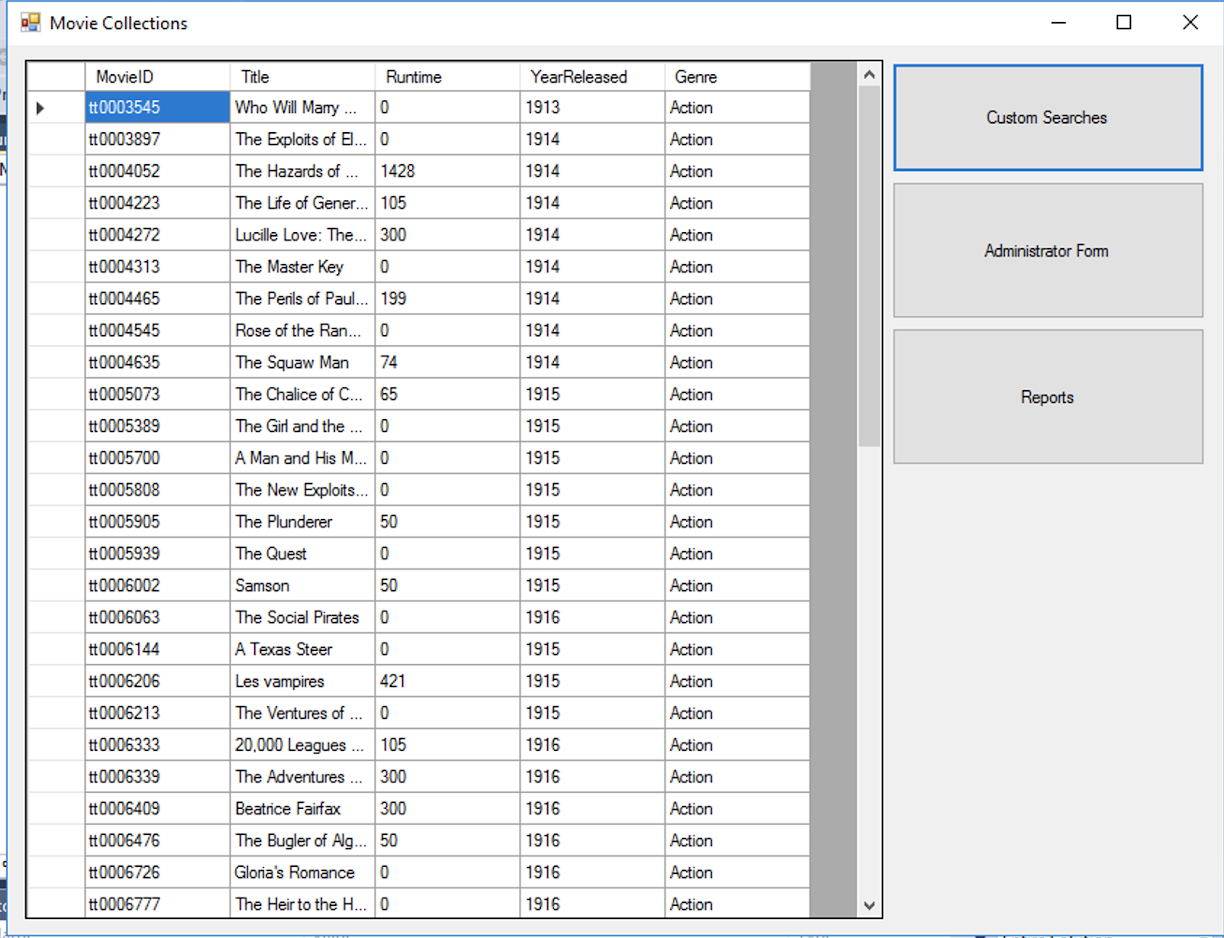
****

**System Features and Usage**

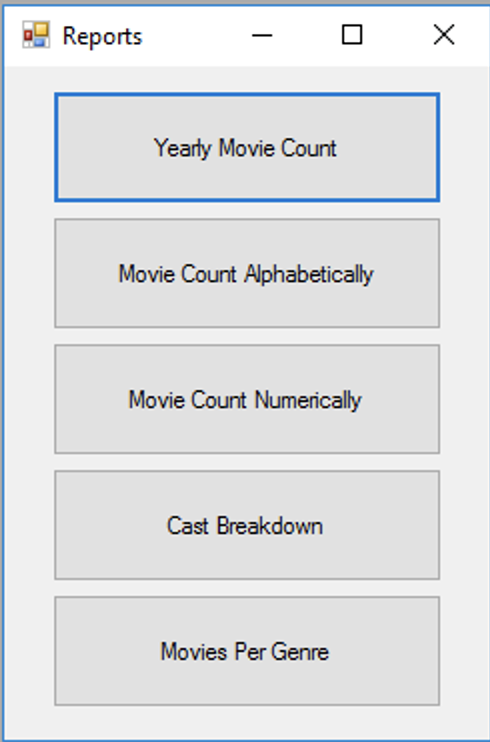
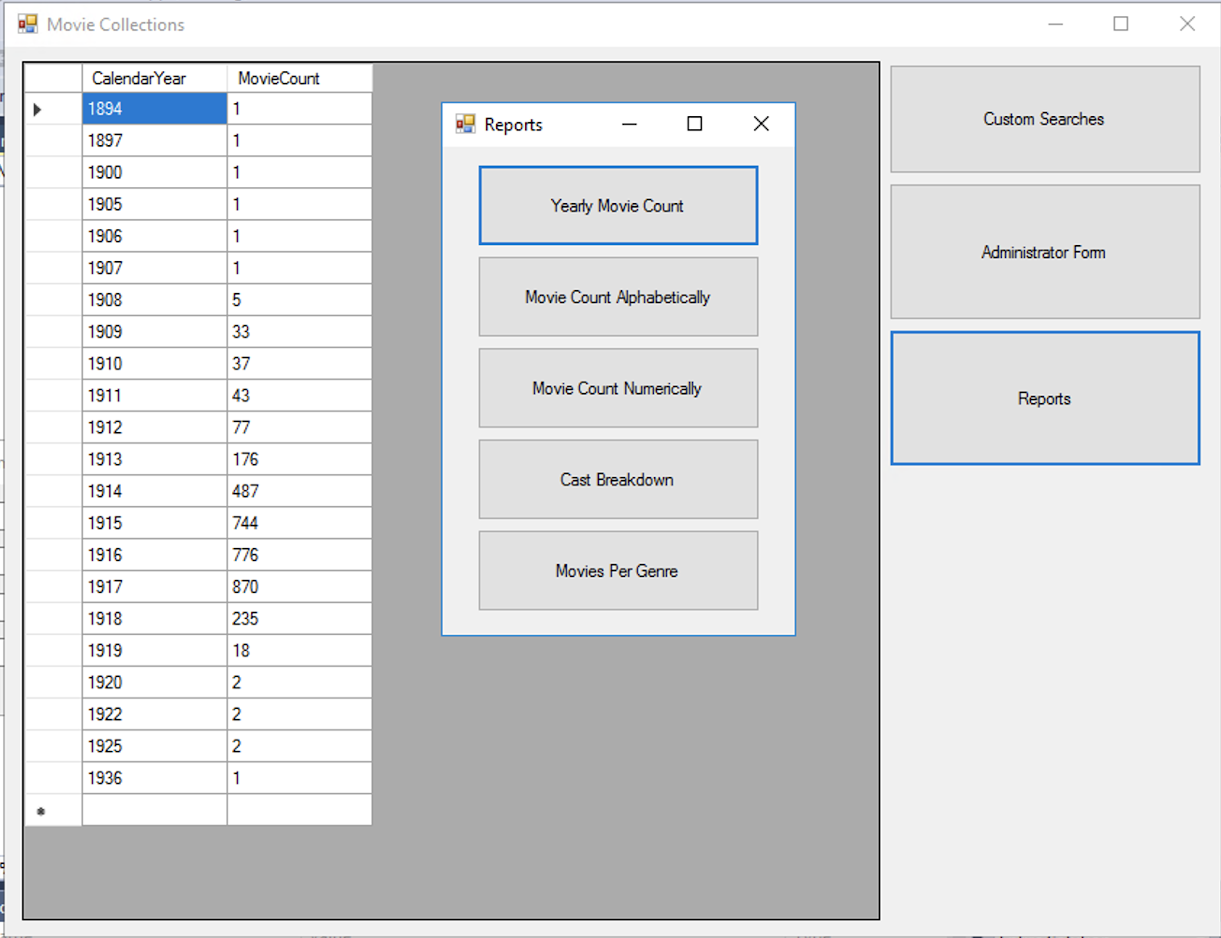
There are 3 different forms that a user can interact with while using this application. A few scenarios include, when a user wants to create a custom search to look through the database. First, the user would click the button *Custom Searches* and a new window is created for them. After looking through the different possible search options, the user is then able to decide what attributes of a movie to search for. Once the user enters the required information, they are able to press the “Confirm” button and the application will pass and EXEC SQL string to the database for processing. The form *Custom Searches* closes and the *Movie Collection* form is populated with information returned from the database.



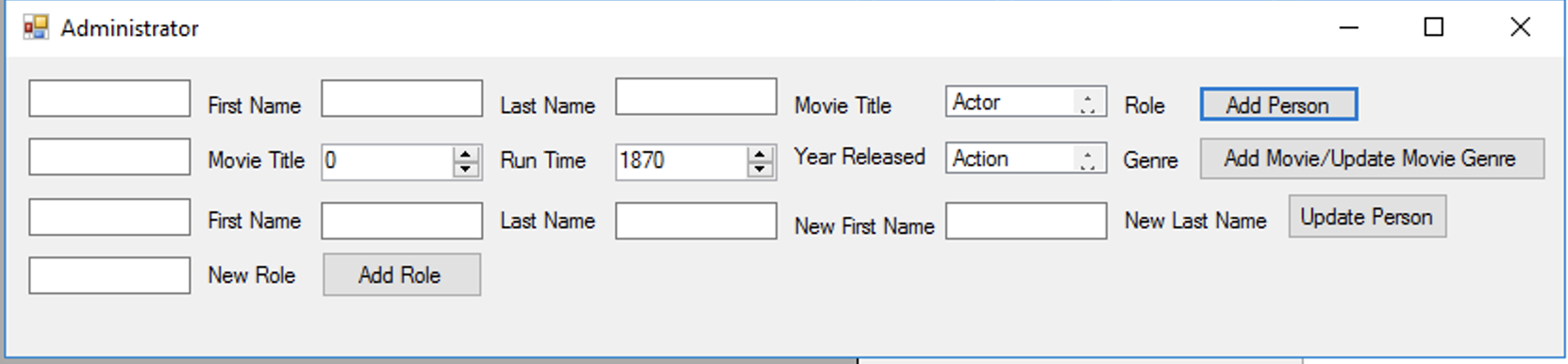


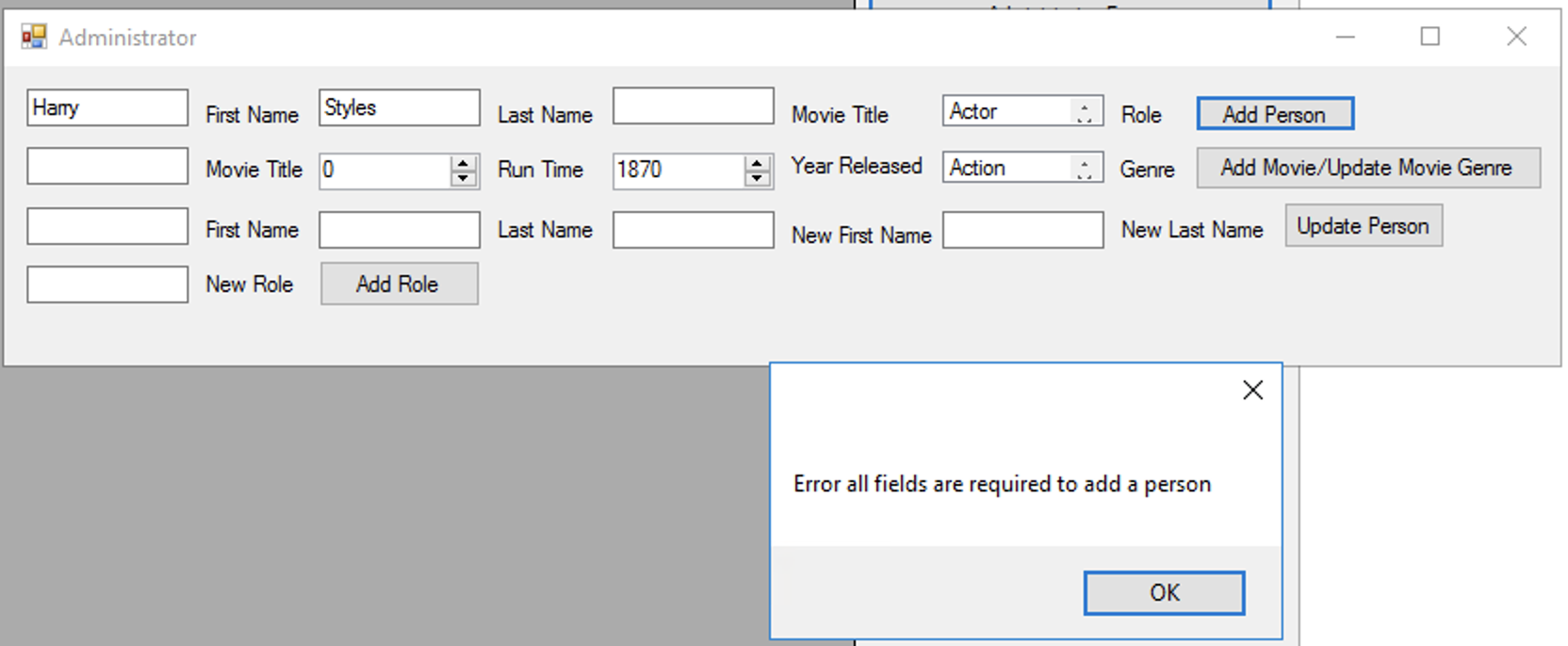


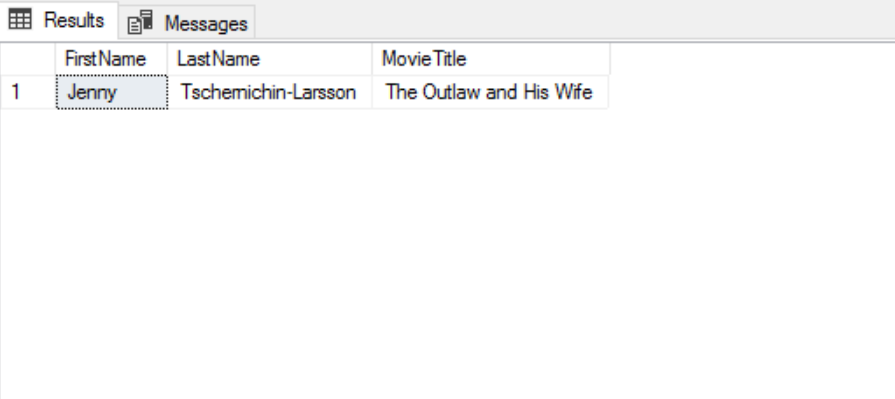
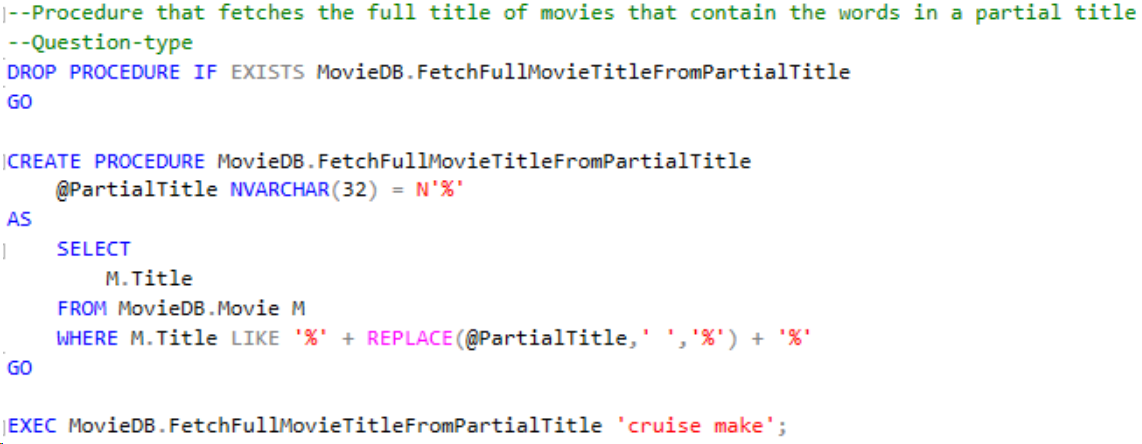
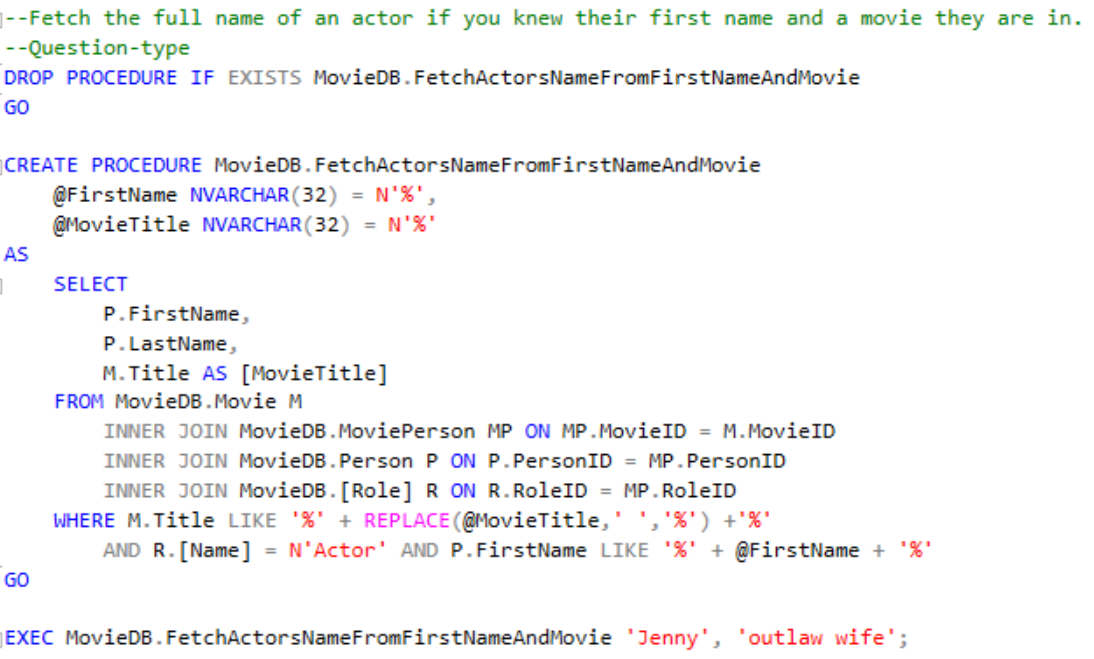
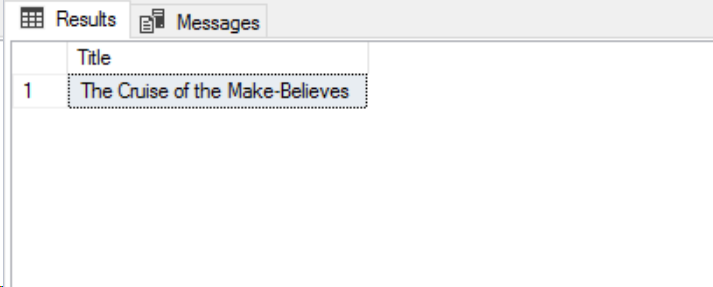
The second use-case is when a user wants to look at different reports that show all of the baseline information from the database. Users will click the *Reports* button and a new form will open. From here, they will decide which kind of reports spark their interest. Since these reports are all static aggregations, all the user needs to do to retrieve the information is select the columns they are interested in. Once they click one of the five options, the window will close and the *Movie Collections* form will be populated with the information from the database.

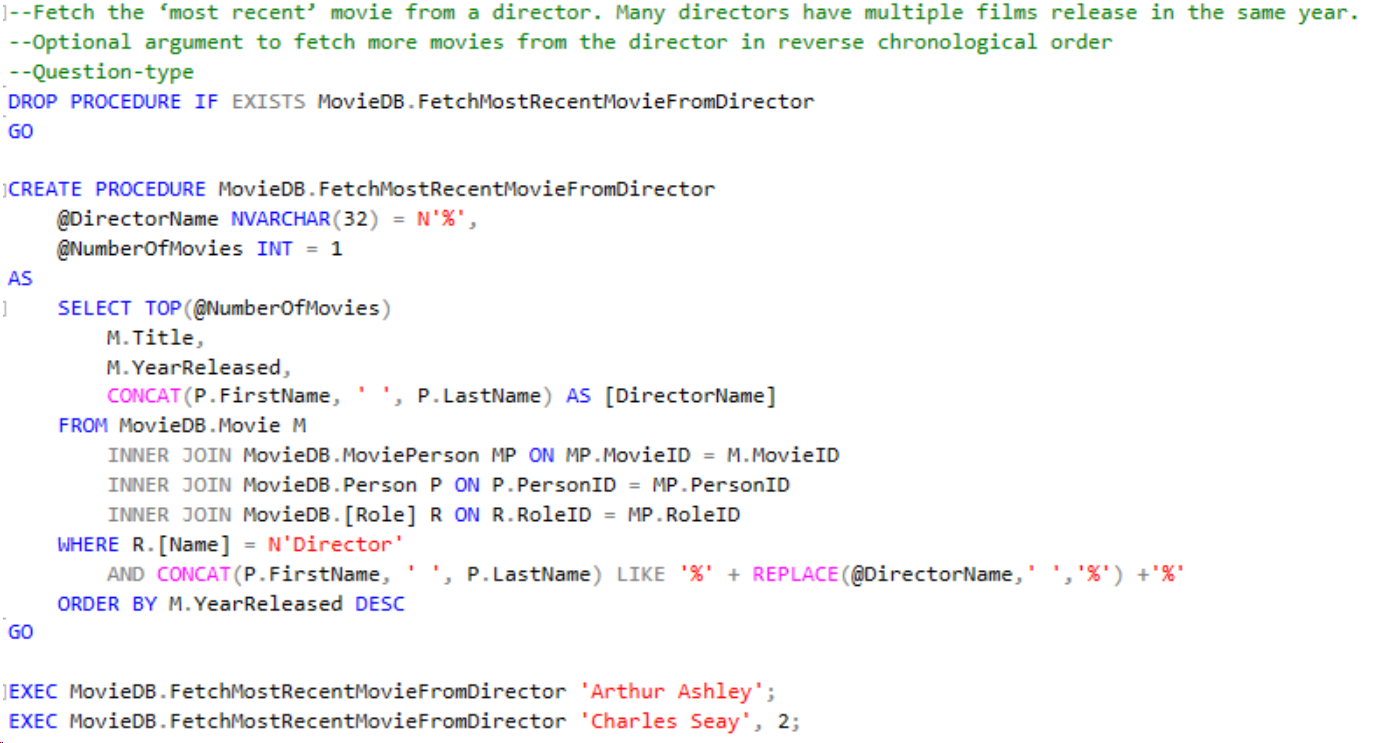
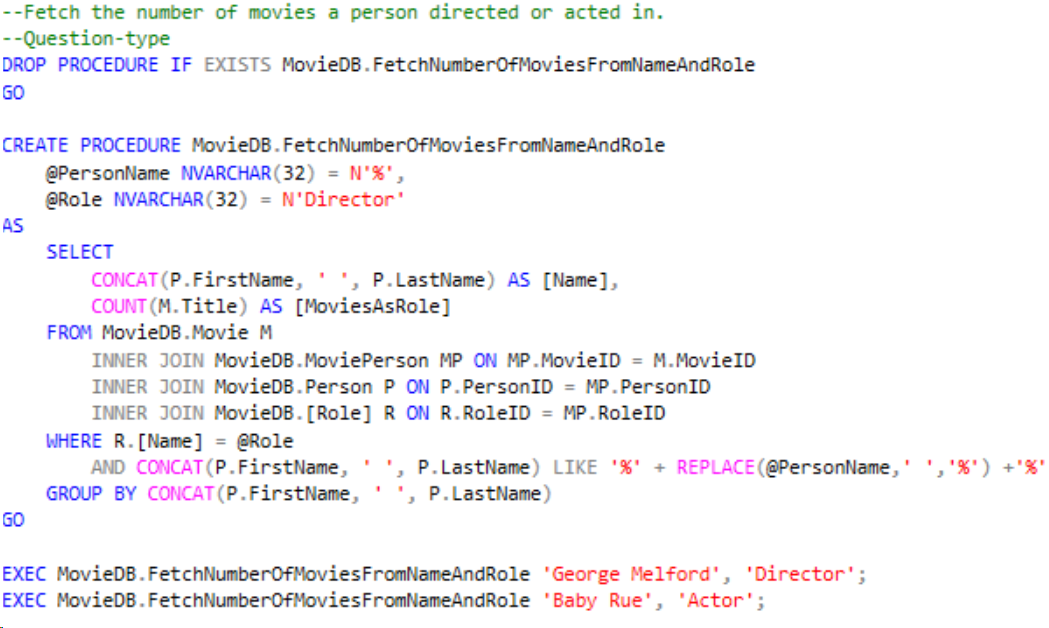
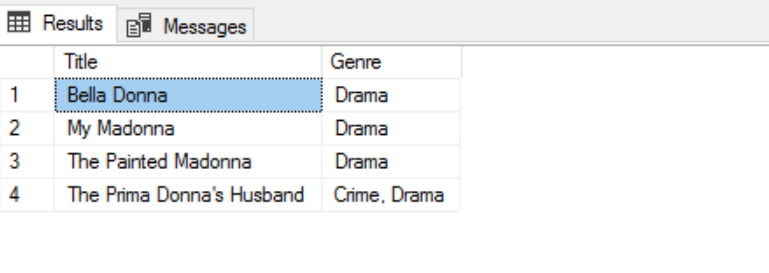
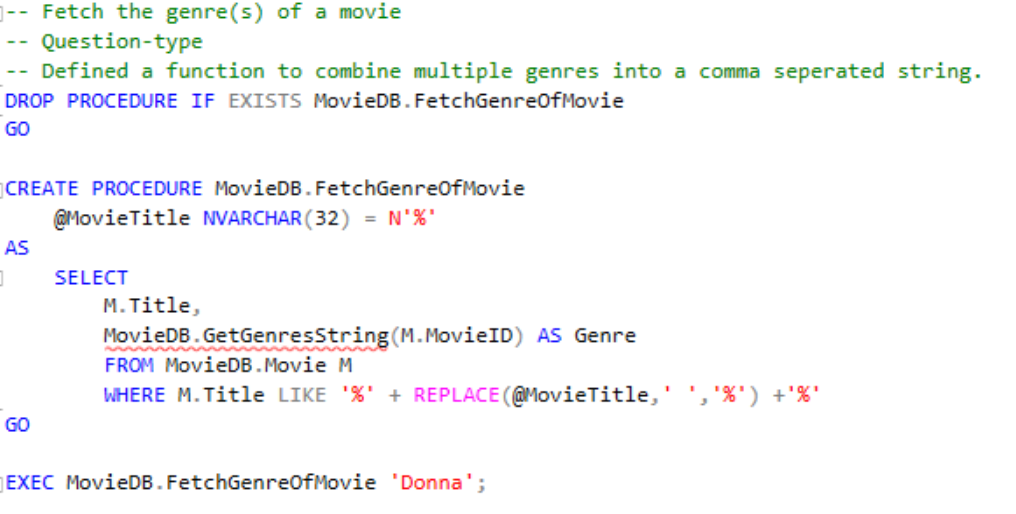
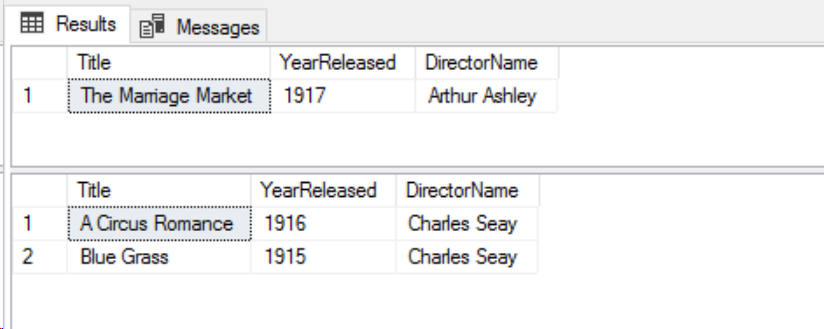
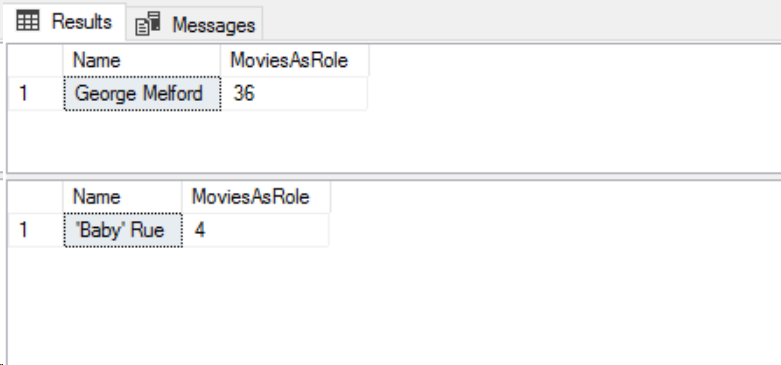


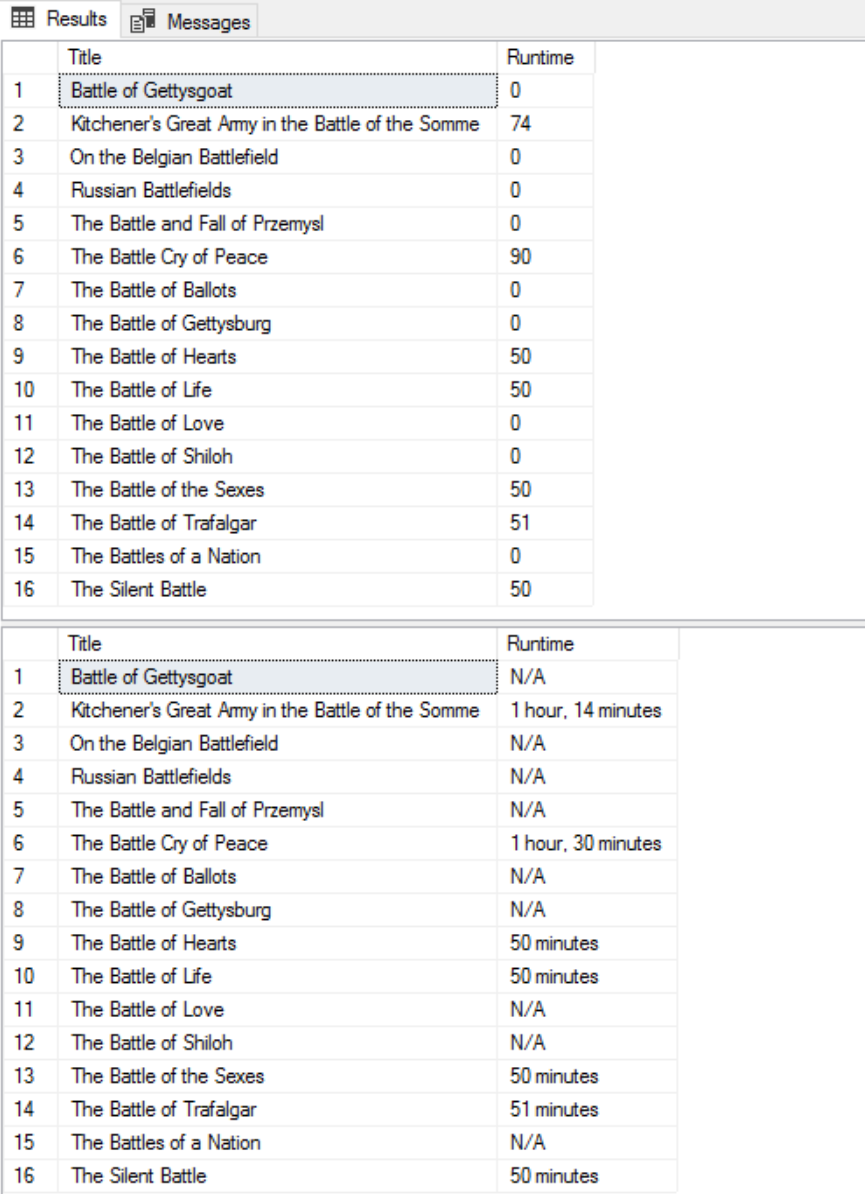
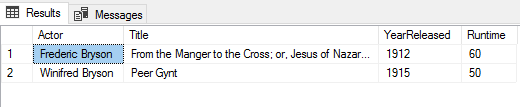
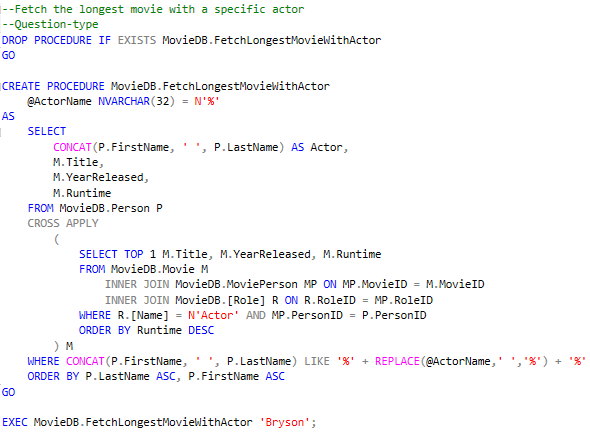
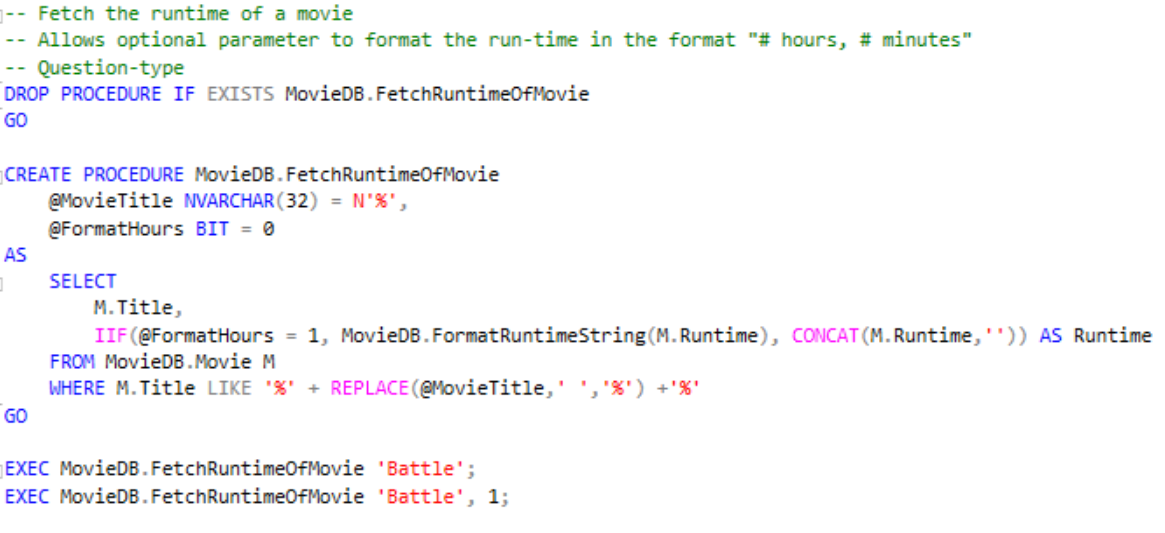
The final use-case discussed in this report is when an administrator wants to change the database while using the application. They will click the *Administrator* form and a new window will open. The administrator will then decide what need to be updated or added to the tables. In the case of adding a person, they will insert the person’s information, including the First Name, Last Name, Movie Title they are associated with, and finally their role in the movie. Finally, they will click the *Add Person* button. If any of the fields haven’t been filled in, then an error will be displayed on screen reminding the administrator that all fields must be filled in order to add a person. Otherwise, a message box will tell the user that the information has been added successfully.

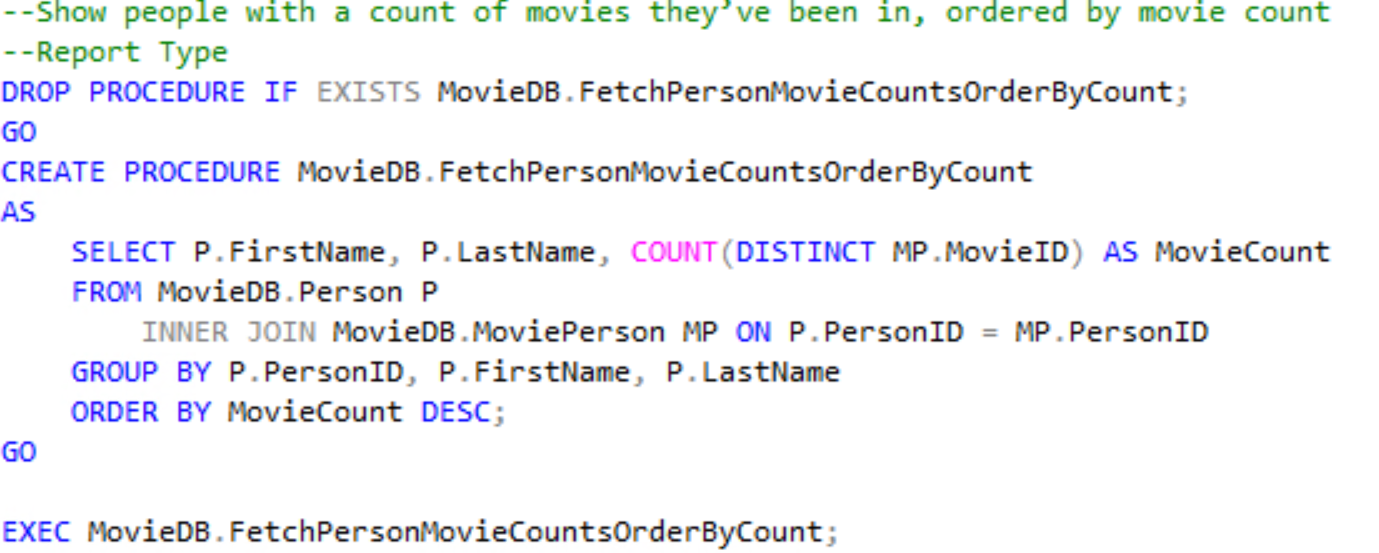
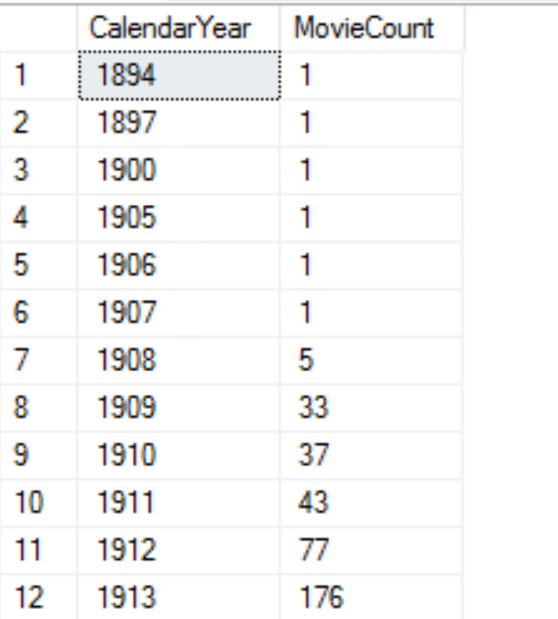
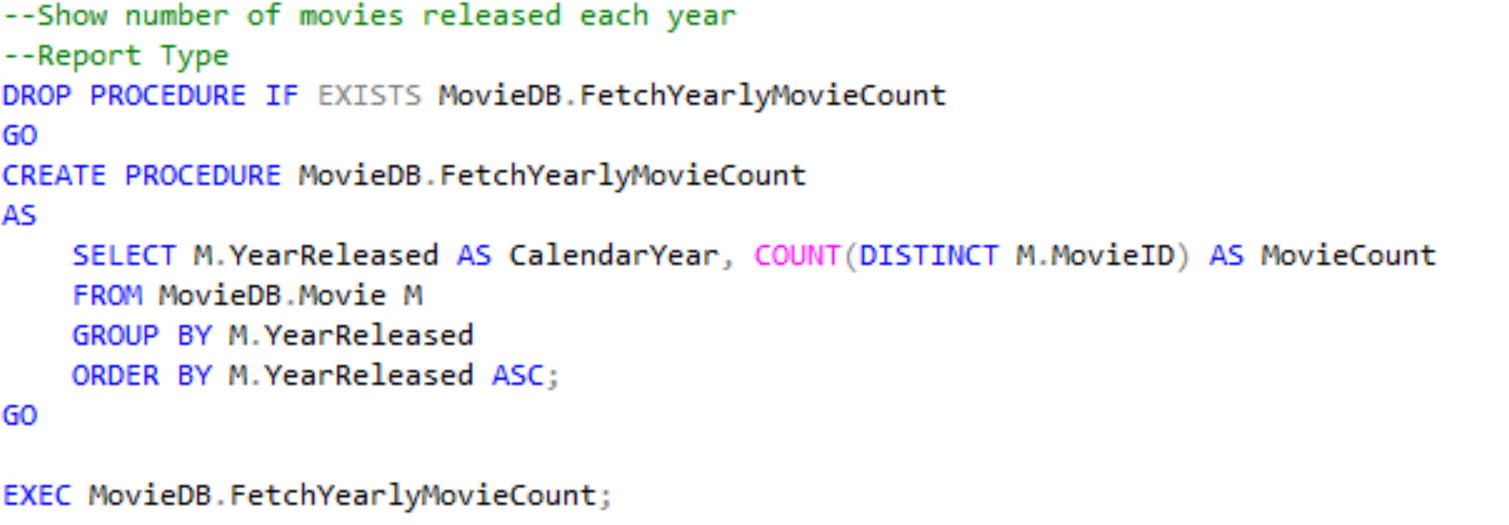
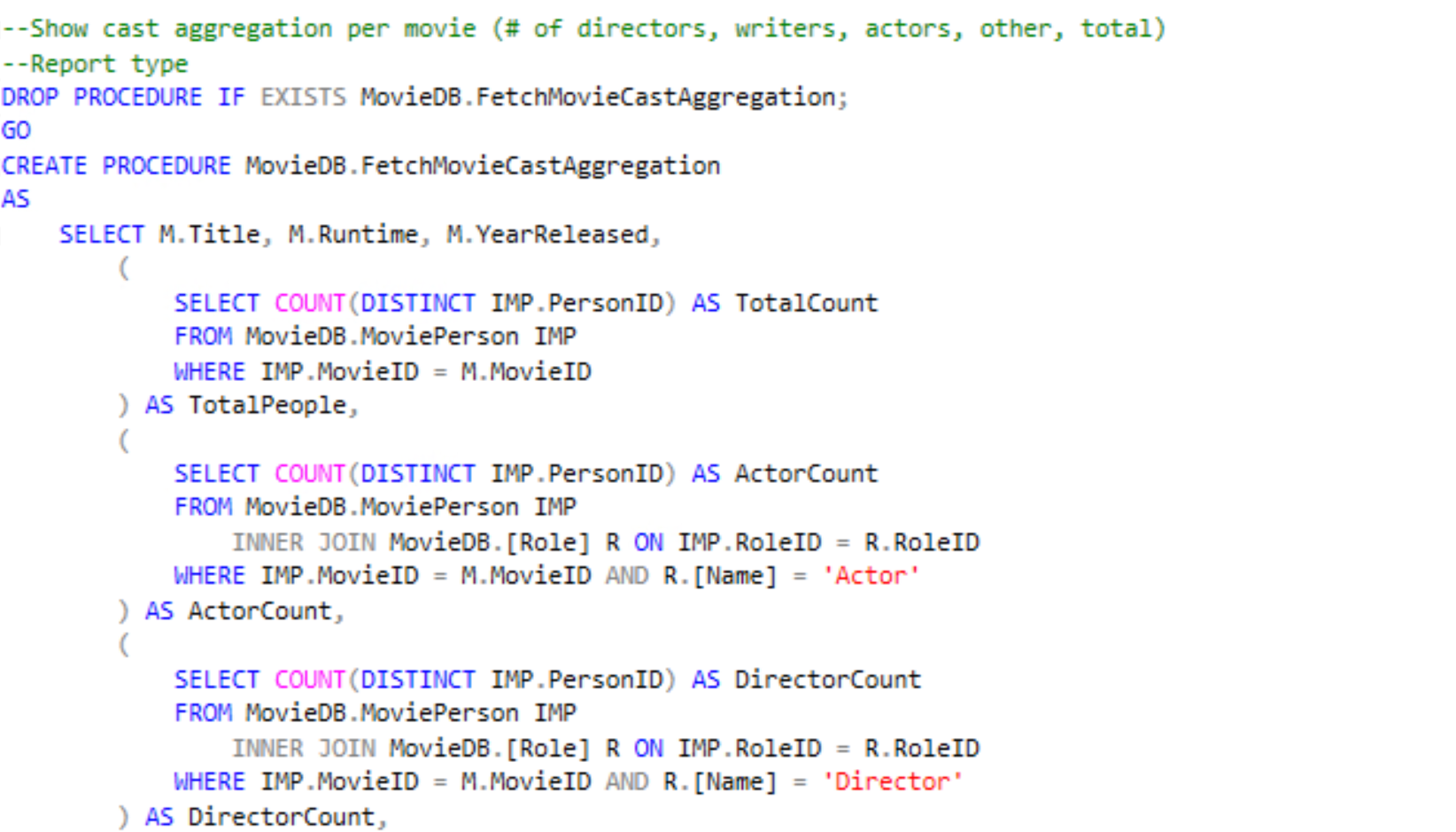
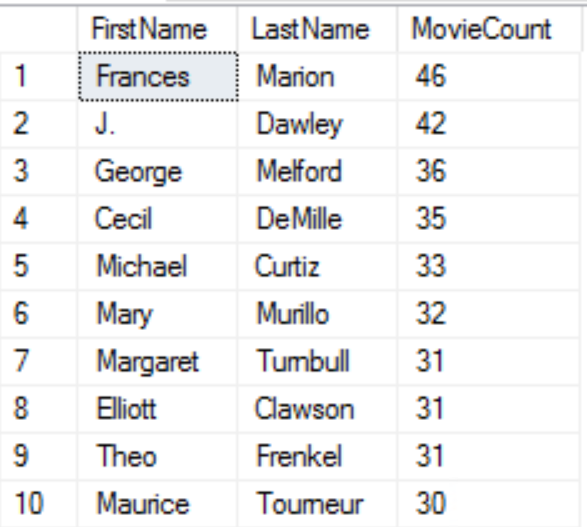


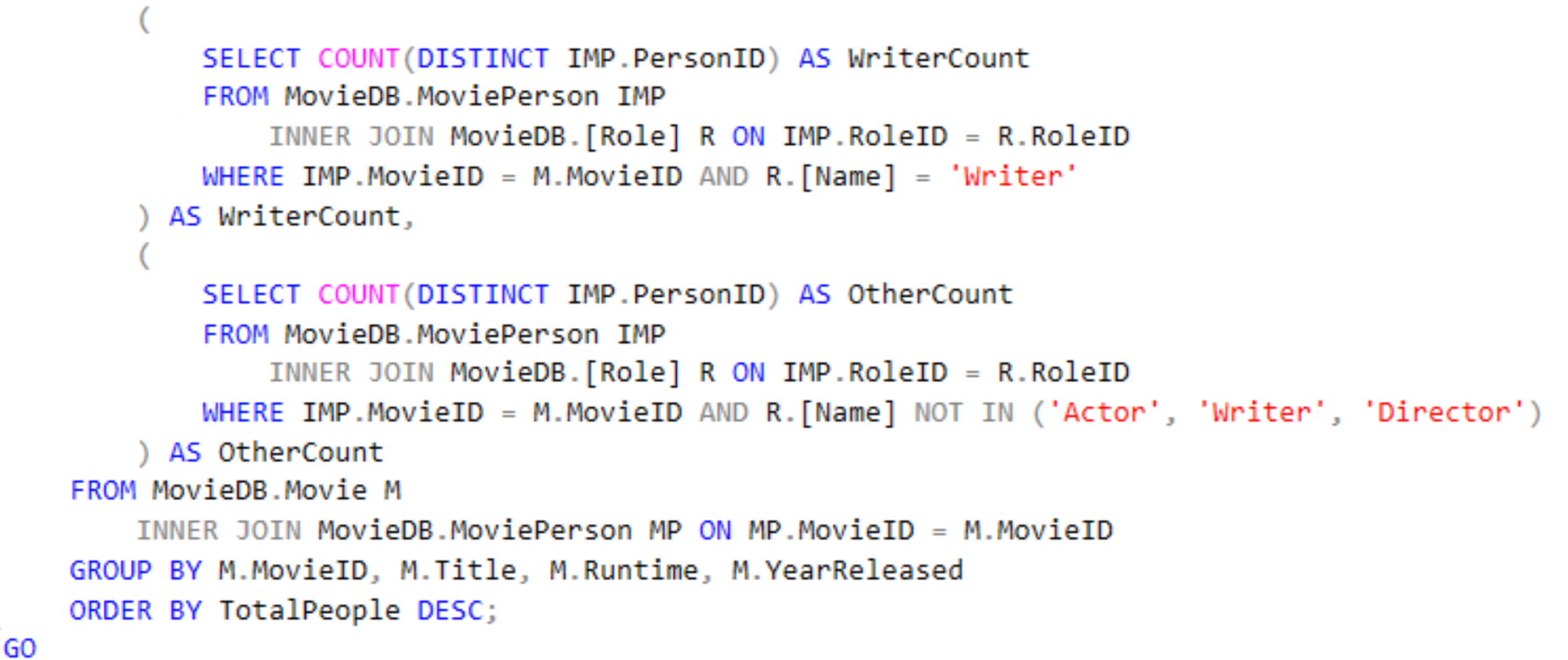
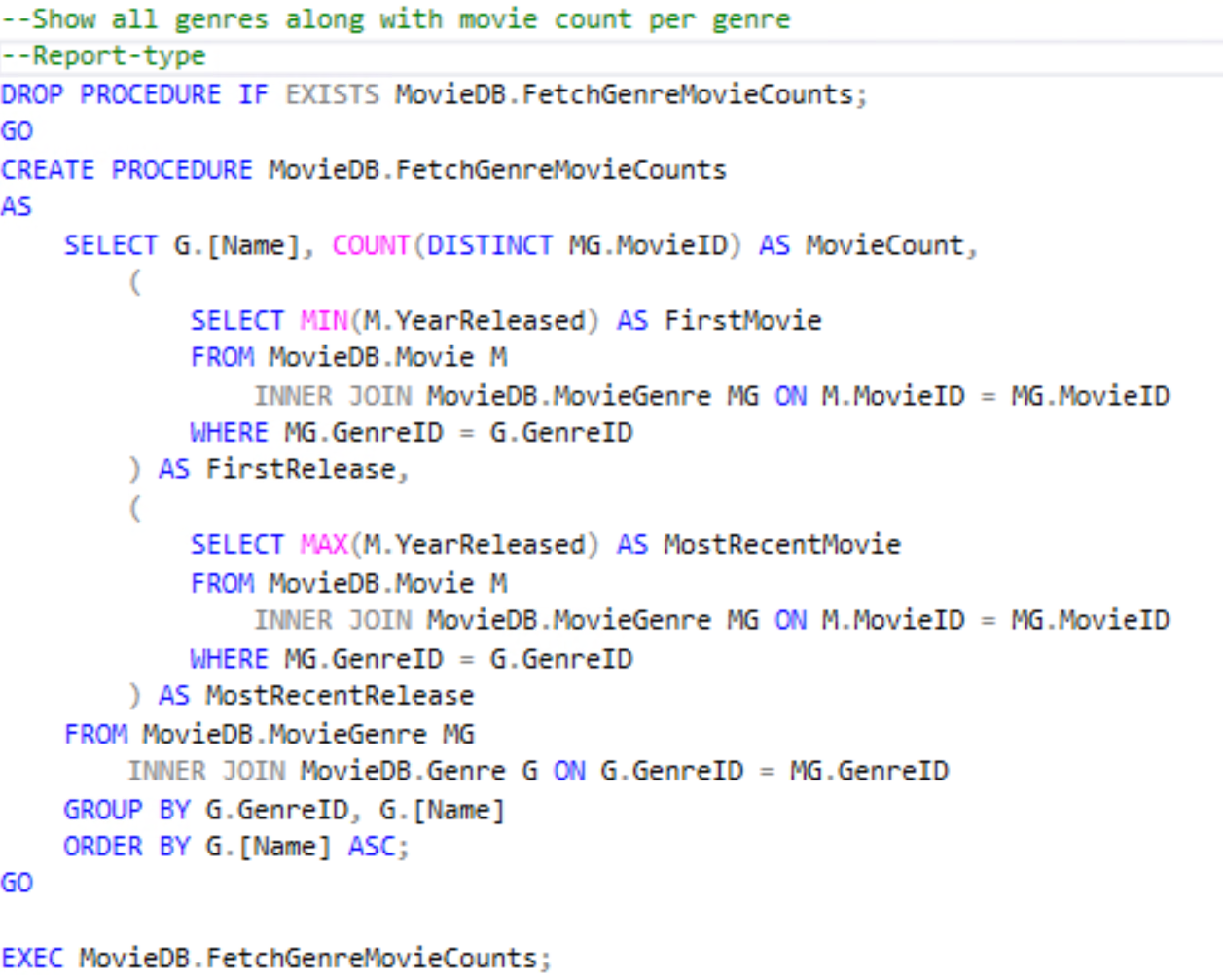
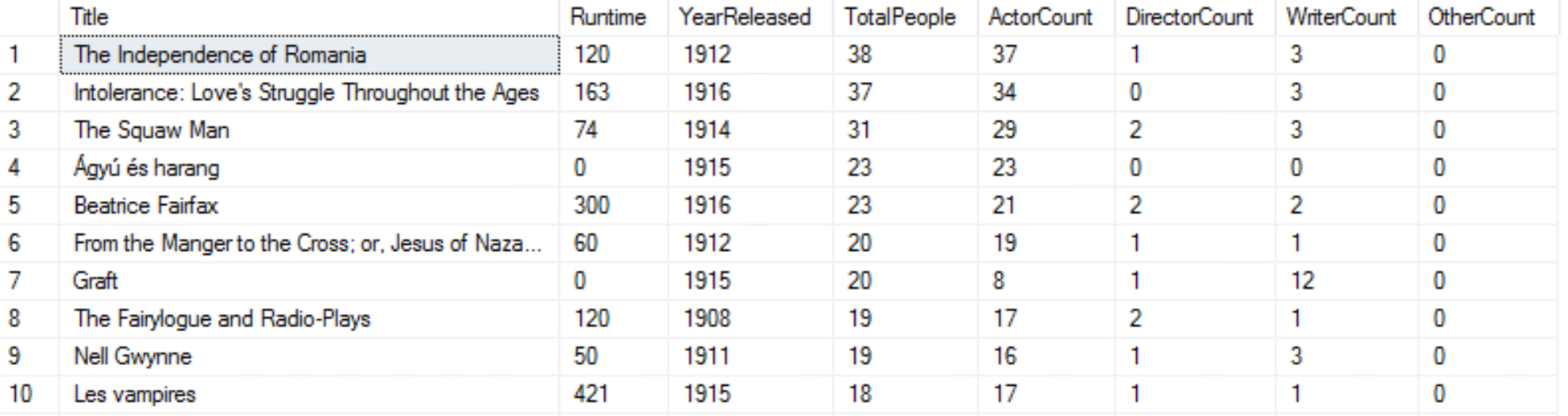
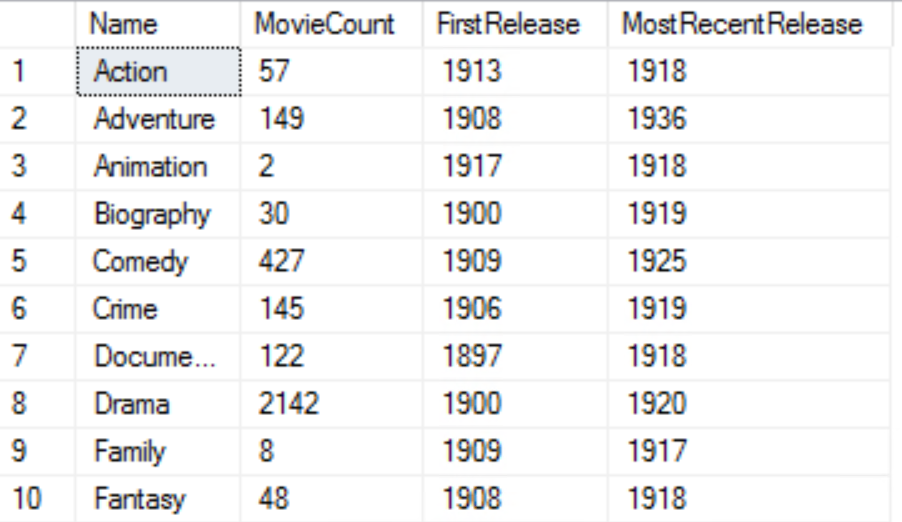


**Queries and Reports**









**Summary and Discussion**

Our project is fairly complete compared to the original vision. We did not implement an average cast size, instead we implemented the cast size of each movie. Additionally, we changed being able to see how many times a certain person has appeared within a Hollywood movie to seeing how many movies a person has appeared in. We were able to connect the SQL with C# to allow users the functionality to retrieve information with the click of a button. Finally, we updated our database design to include unique keys. This was something we had missed during the project proposal, but was necessary to maintain the integrity of our database.

Through this project we realized that we populated our database with data that did not contain any recent movies. However, our biggest challenges came from topics that had not been discussed in class. This includes how to separate comma-separated values into multiple rows and connecting the C# language to SQL. It took us a significant amount of time to learn how to make the application we had created interact with the database. Additionally, the data we utilized in our database originally was not in a workable format, such as, having unsupported characters. This required us to figure out how to merge the table format of the data into the table format we had created. To overcome this challenge we had to create a program to update the .csv file in order to remove characters that were not supported in MS SQL Server. This meant changing the original data into readable NVARCHAR variables. We also had many challenges when creating the procedures to update and add information into the database. The original data tables included anywhere from nine to fifteen million rows; therefore, the queries took anywhere from five to twenty minutes depending on the processing power of your computer.

As for possible improvements, the biggest improvement we would make would be having more up-to-date data. In the future, we could find a more recent data set and merge it into our database allowing us to have both old and new data. We would also want to implement more flexible searches and a more friendly UI design. Having more individual information on each movie and person would also allow us to provide our users more information with an immersive experience within our application. Providing more information on movies and people associated would also expand our intended audience. We could also expand our database to include theaters and show times as another addition to our history database.