Team 16: Joe Boutte, Anastasia King, Brayan Quezada, Anson Jones, Ian Riley, Alexander Dubrovsky

John Keller

CIS 560/562

Thursday December 6th, 2018

Discount IMDb

Introduction

Our database is a compendium of movies that people can browse or search through. Admins of the system can add, edit, or remove entries as new movies come out, errors in the data are reported, or a movie is deemed inappropriate for a valid reason. Users of the system can search for movies based on any of the search criteria, including: title, director, actor, runtime, and many more. The idea is anyone who is an avid watcher of movies can use our service to search for movies they may like based on similar directors, actors, or genres these movies have in common. Further uses would be to find a movie the user could watch given their time frame or the audience that is watching the movie.

Technical Description of the System

Our system consists of a front-end coded in C# and a back-end coded in SQL, respectively. The front-end is a C# form that allows for the user to search through the available list of movies by any column they want. It uses several C# libraries including: System Data, System Drawing, System Data SQLClient, and System Windows Forms.

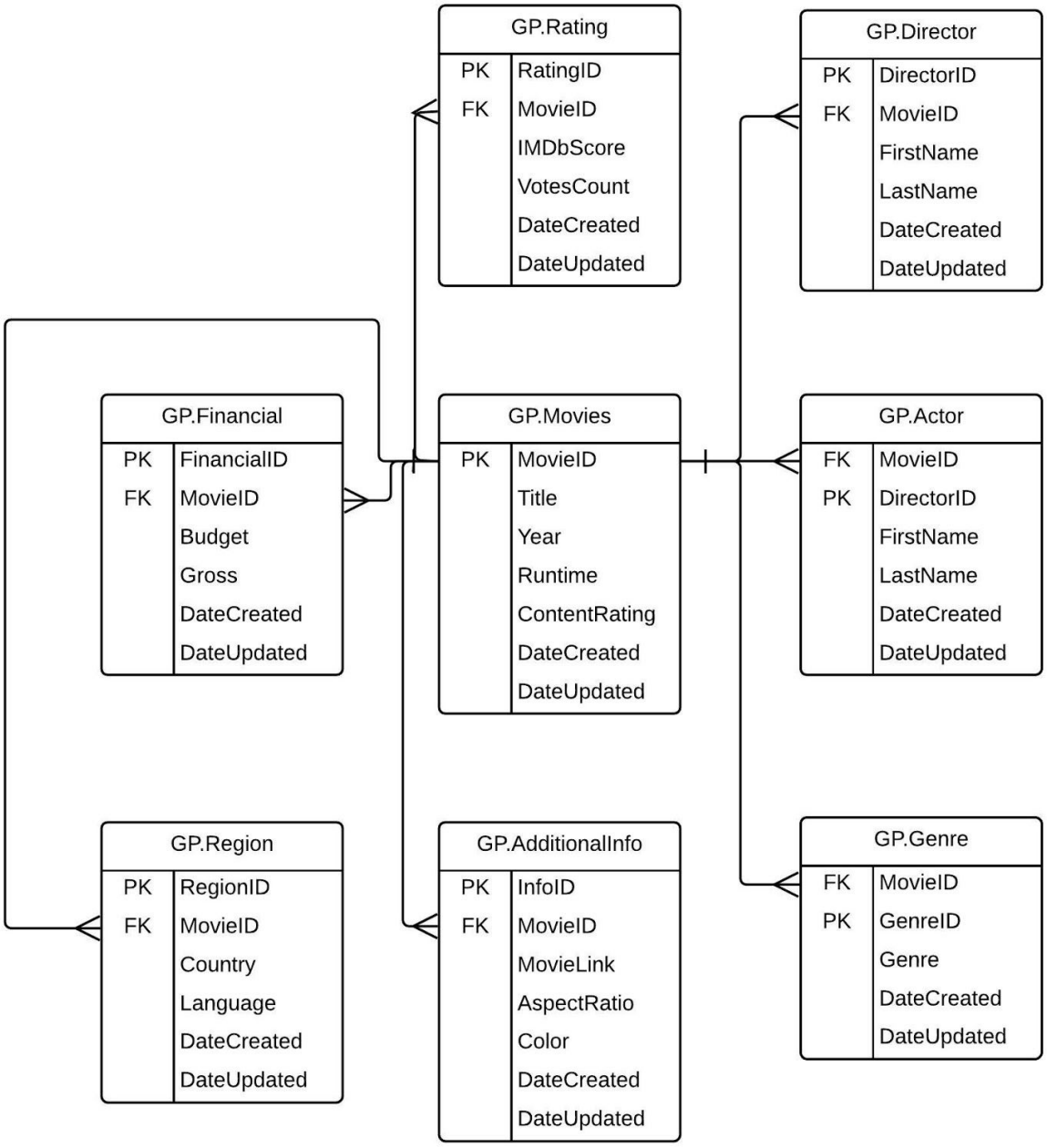
The most important of these is System Data SQLClient, which allows us to interact with the SQL database from our C# program, both to input all of our data and to query the database for our results. Through the SQLClient library, we were able to establish a sql connection to our database that allowed for the application to use specific sql procedures and C# functions designed by our team. Our application provides functionality for adding, updating, removing data as well as data manipulation for user interaction. Each of these functions uses a designed sql procedure. Each procedure is designed to use the variables from the C# application to display defined results. We tried to produce each procedure with enough functionality to enhance the user experience and allow for diverse lookups on tables. The procedure design's were based on what available data we had and what possible lookups might be necessary for each possible use case. Overall, we couldn't define every possible procedure for every possible lookup, but we came up with a few unique procedures that produced interesting results.

Database Design

We had to change our tables for our database multiple times because we had a hard time getting our data imported just right. At first our database multiple tables, but we did not have data to populate all the tables. Later we slimmed it down to only 4 tables because we were able to get data to add to them that we could make working queries for but then we did not have enough relations. Now our database design includes eight tables; movies, rating, genre, region, director, actor, financial, and additional information. These tables each contain different pieces of information about each

movie, all tied together by the primary key that is the movie ID from the main “Movies” table, everywhere else it is a primary, foreign key that references back to the “Movies” table. The tables are designed to group similar data together for easier searching, such as the ratings table including their IMDb score, the number of votes behind that score, and content rating of the movie. Of all the values in all of the tables, only the identity value for the movie ID is set to unique. We did this because the relationship between the movie and any of its attributes is a one to many, where one director can have directed many movies, one actor can have starred in many movies, and many movies can have the same content rating. We included a system date time offset so we can know when an attribute was created and last modified for the purpose of finding the newest entries. User is able to search for movies by specifying multiple criteria, including movie title, release year, director, lead actor, budget, gross and more. We also provided user with ability to filter movies based on the rating range. The rating is from IMDB dataset and the user can define the lowest and highest movie rating which creates a range for movies to fall into for search and filtering. Another interesting feature of our application is the ability to display the top one hundred highest grossing movies. Other more simple query provides the amount the movies by a specific director grossed all together.

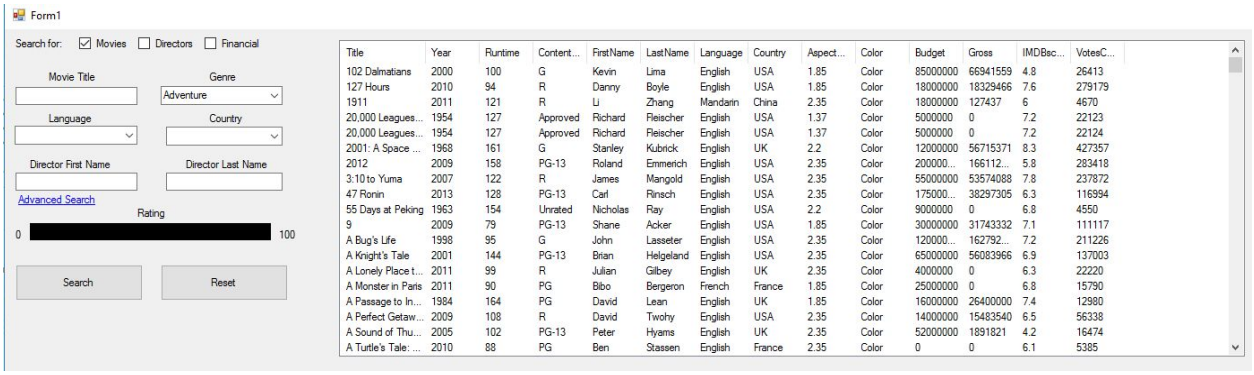
Design:



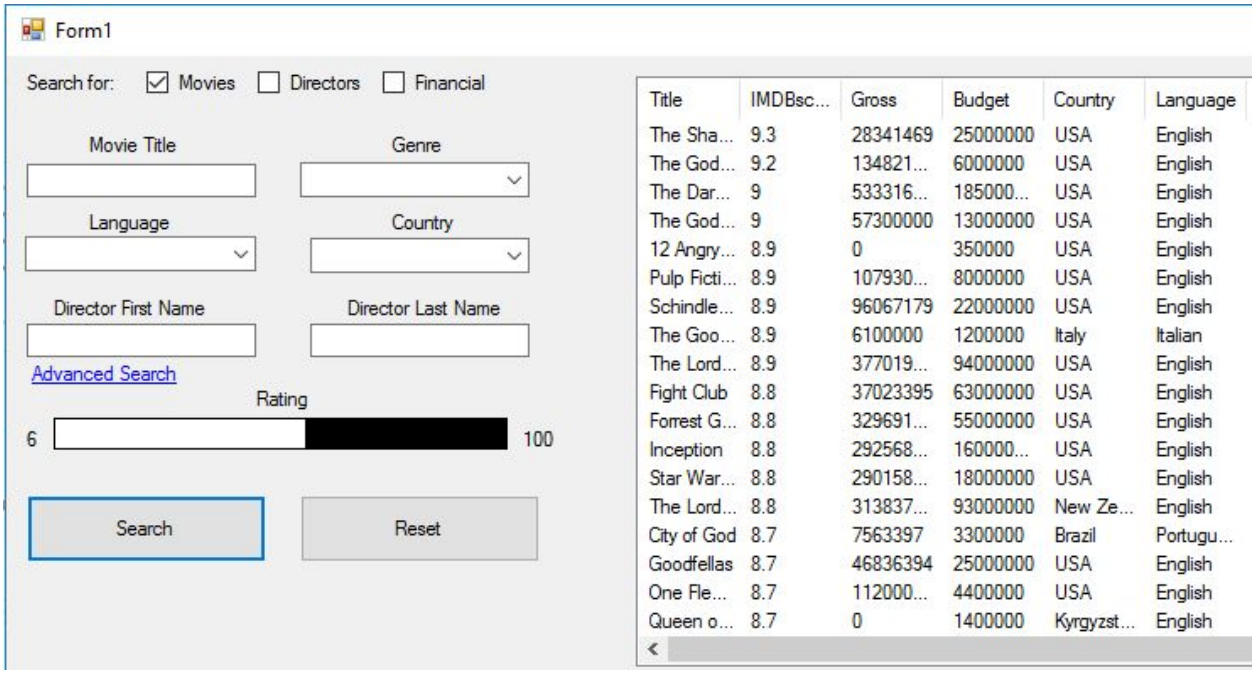
System Design

Searching for movie information based on movie title: This is when a user desires information about a movie, and they know the exact movie that they want.

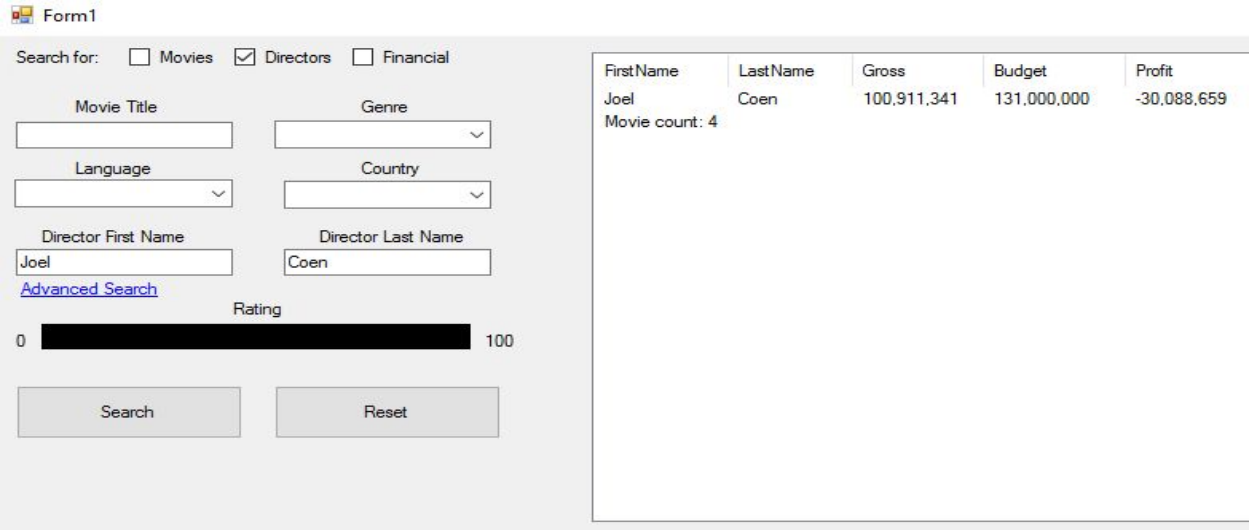
Search for movie information based on the genre: This is used when a user would like information about movies, and they know the genre (it is also possible to use this query on other fields as well, not just genre)



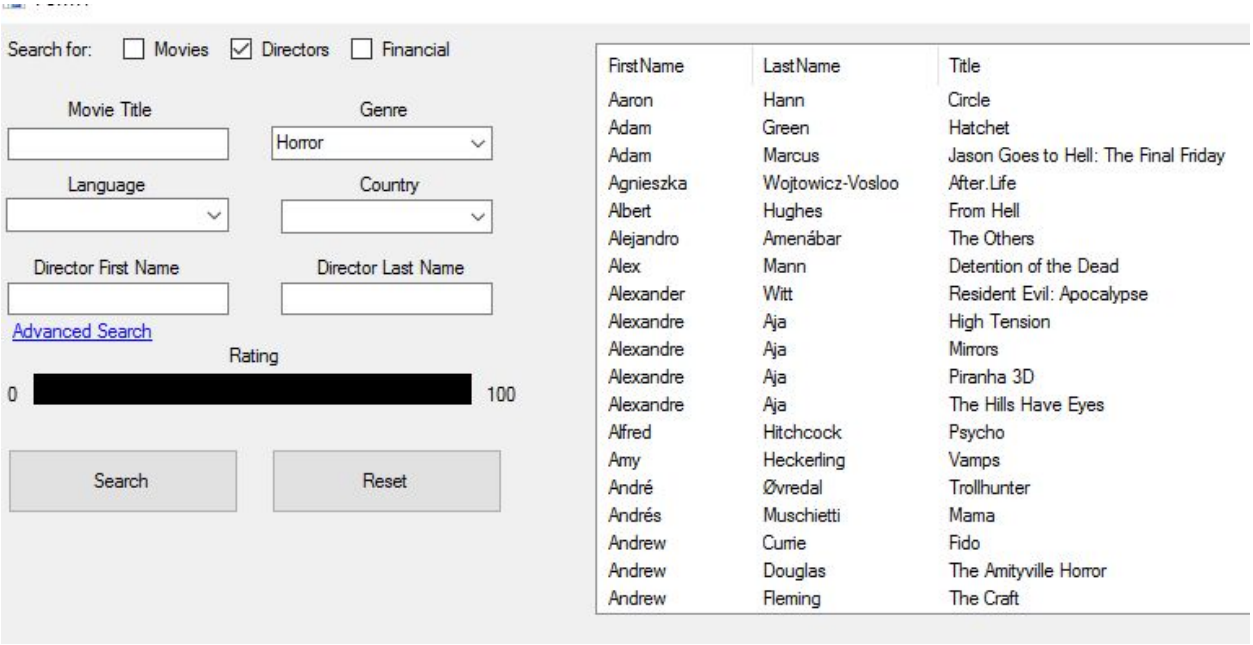
Search for movies based on rating: This is used when a user wants to find movies within a certain rating range. This will filter the movies on the range between 6 and 10.



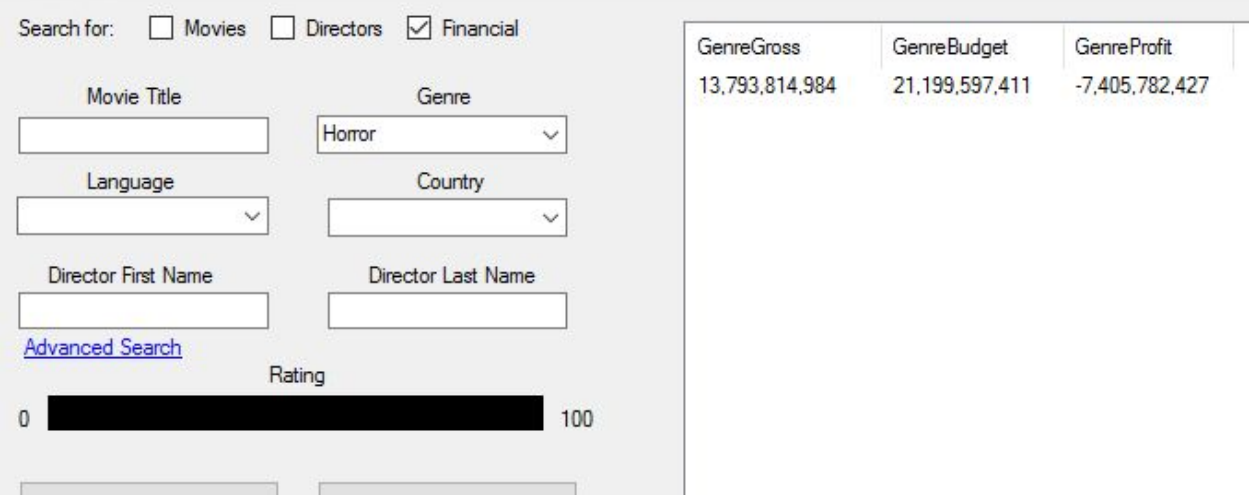
Find director financial information based on that director's name: This query is used when a user knows the director that they are looking for, and they would like to see financial information for that director. For future work, this query would be broken out of this style and users would be able to decide if they want the financial information or the director information.



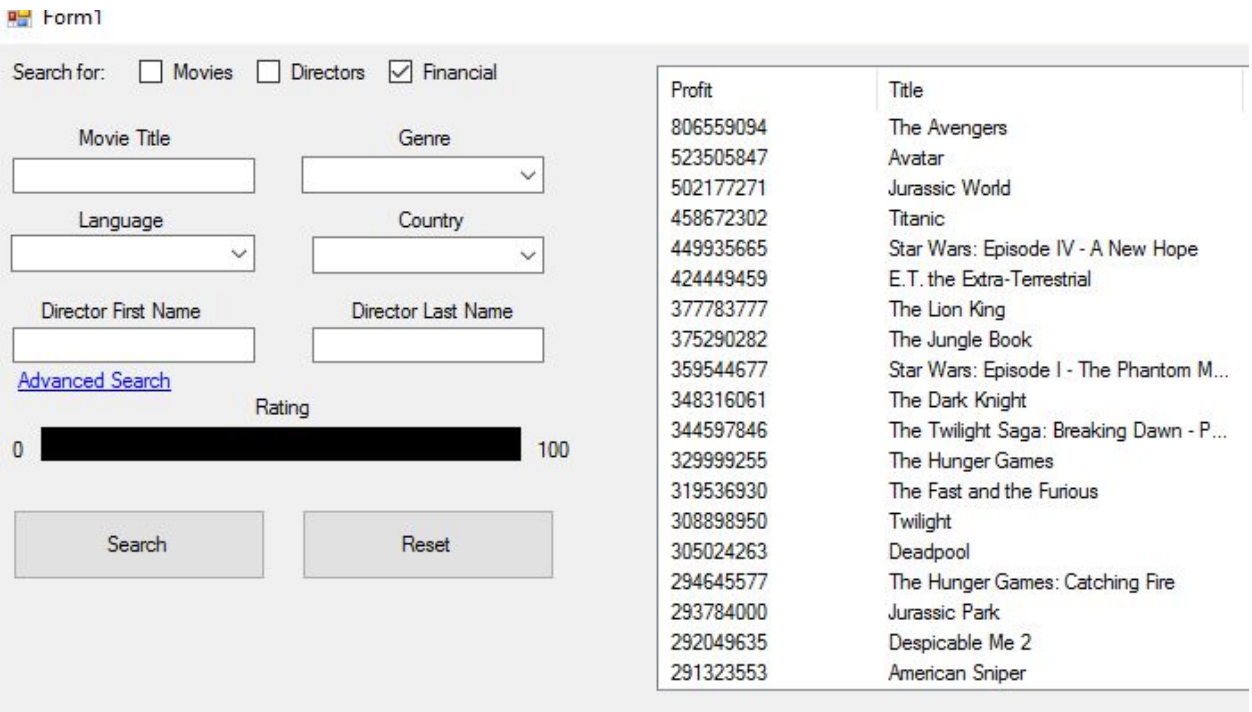
Find director information based on genre: This query is used when the user would like to find a list of all directors and their movies for a certain genre. Like the movie search, this search can be done on the other fields as well as a complete comprehensive search feature.



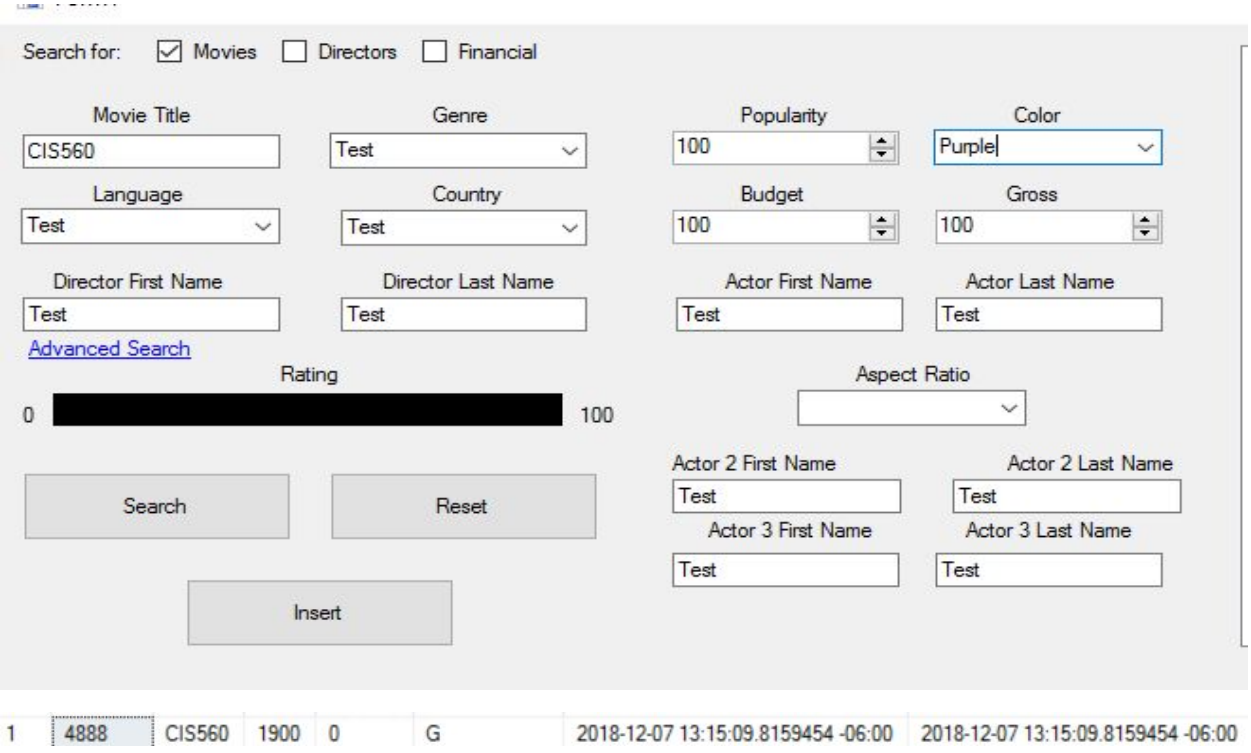
Find financial information based on the genre of movie: This query is used when the user would like to see the gross, budget, and profit per genre.



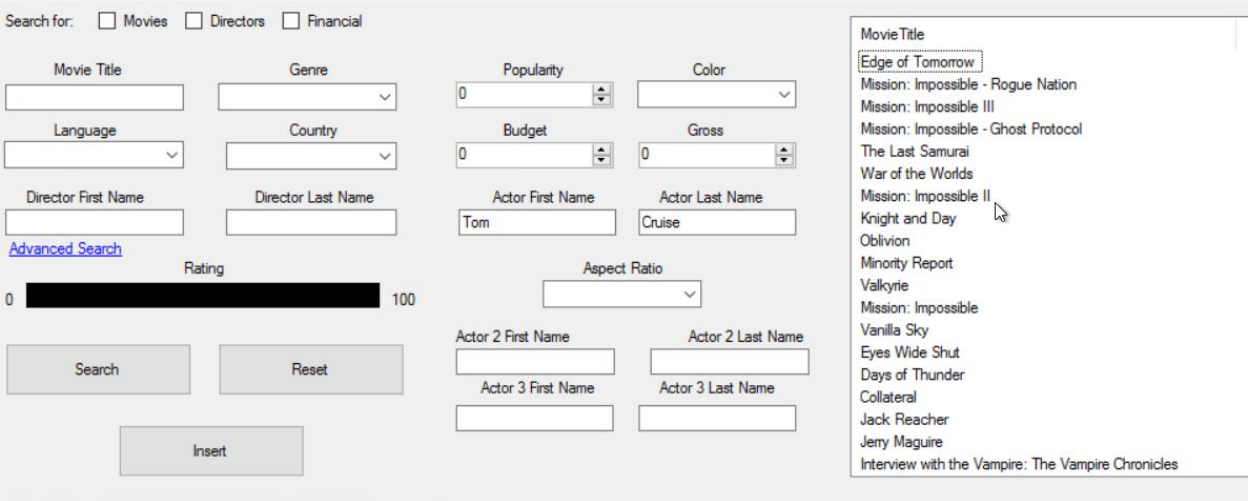
Find the top 100 grossing movies: An empty search field on the financial tab will bring up a list of the top 100 grossing movies and the profit that they made.



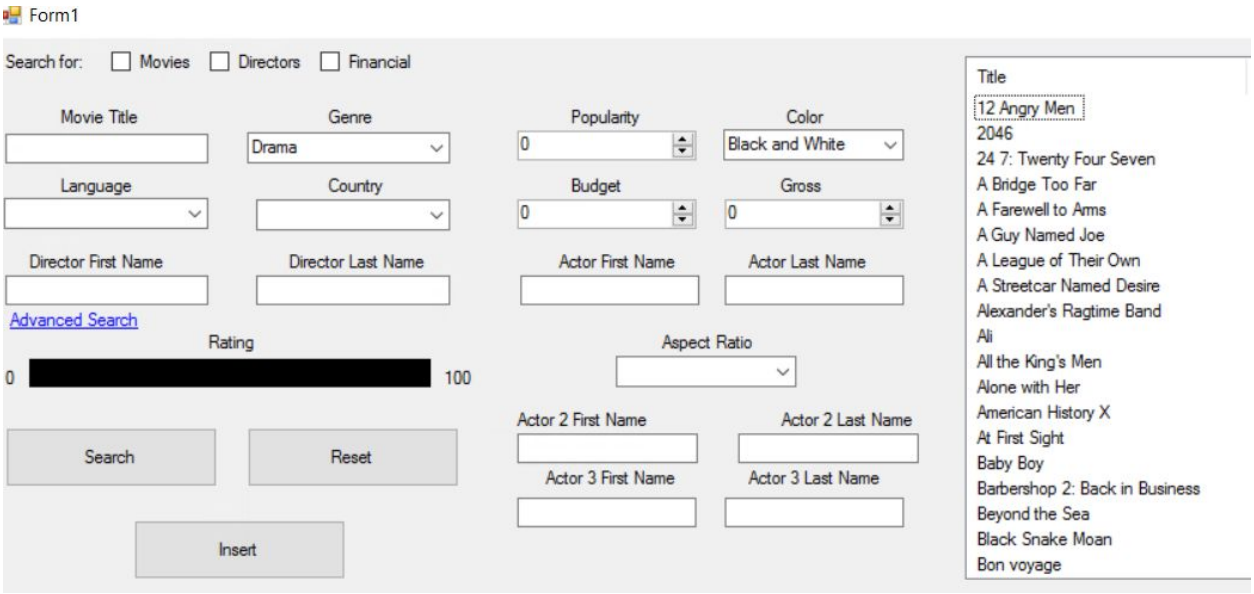
Inserting values into the database: This is the output in the database, (the GUI does not have output upon the input).



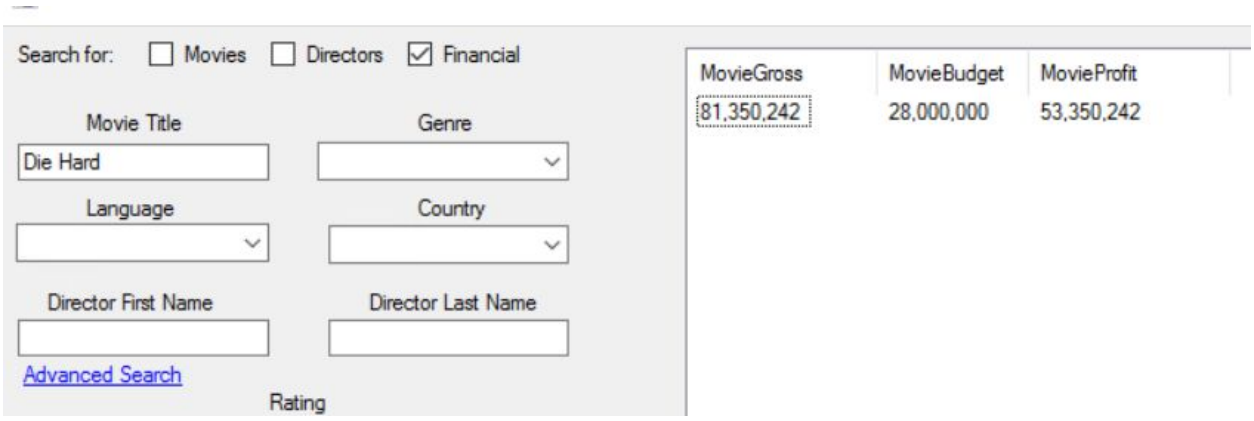
Find list of movies that an actor has appeared in: Using the advanced search function, a user can input a name into the actor first name and actor last name text boxes to get a list of all of the movies where that actor stars in. Currently, the actor 2 and actor 3 name boxes do not work, but in future work, these would be implemented so one could look for movies based on multiple actors.



Find list of movies that match certain criteria: Again using the advanced search, a user can search for a list of all movies that match their desired criteria. In the screenshot, the query was to search for movies based on genre, and if it's black and white.



Find the financial information based on the movie: The user inputs a movie title and has the financial tab clicked and the program will return all of the financial information for that movie.



Queries and Reports

Query 1 (Question Type): Finding a movie based on the movie title. This query takes the movie title as a parameter, and returns all of the pertinent information for that movie title.

```
DROP PROCEDURE IF EXISTS GP.TitleSearch
GO
CREATE PROCEDURE GP.TitleSearch
  @Title NVARCHAR(50)
AS
SELECT M.Title, M.Year, M.Runtime, M.ContentRating, d.FirstName, d.LastName, rE.Language,
rE.Country, aI.AspectRatio, aI.Color, f.Budget, f.Gross, r.IMDBscore, r.VotesCount
FROM GP.Movies M
        INNER JOIN GP.Genre g on g.MovieID = M.MovieID
        INNER JOIN GP.Director d on d.MovieID = M.MovieID
        INNER JOIN GP.Region rE on rE.MovieID = M.MovieID
        INNER JOIN GP.AdditionalInfo aI on aI.MovieID = M.MovieID
        INNER JOIN GP.Financial f on f.MovieID = M.MovieID
        INNER JOIN GP.Rating r on r.MovieID = M.MovieID
        WHERE M.Title = @Title
GROUP BY M.Title, M.Year, M.Runtime, M.ContentRating, d.FirstName, d.LastName, rE.Language,
rE.Country, aI.AspectRatio, aI.Color, f.Budget, f.Gross, r.IMDBscore, r.VotesCount
GO
```

Query 2 (Question Type): Movie information based on genre (or many different other parameters). This query takes any of these parameters and finds a movie that matches

all of the inputted parameters. This query is very versatile and can use many different parameters to find the desired movies.

```
CREATE PROCEDURE GP.MovieSearch
        @MovieTitle nvarchar(MAX) = null,
        @Genre nvarchar(50) = null,
        @Country nvarchar(100) = null,
        @Language nvarchar(100) = null,
        @DirectorFirstName nvarchar(100) = null,
        @DirectorLastName nvarchar(100) = null
AS
SELECT g.Title, g.Year, g.Runtime, g.ContentRating, d.FirstName, d.LastName, rE.Language,
rE.Country, aI.AspectRatio, aI.Color, f.Budget, f.Gross, r.IMDBscore, r.VotesCount
FROM GP.Genre M
        INNER JOIN GP.Movies g on g.MovieID = M.MovieID
        INNER JOIN GP.Director d on d.MovieID = M.MovieID
        INNER JOIN GP.Region rE on rE.MovieID = M.MovieID
        INNER JOIN GP.AdditionalInfo aI on aI.MovieID = M.MovieID
        INNER JOIN GP.Financial f on f.MovieID = M.MovieID
        INNER JOIN GP.Rating r on r.MovieID = M.MovieID
WHERE
        (g.Title = @MovieTitle OR @MovieTitle IS NULL)
        AND (M.Genre = @Genre OR @Genre IS NULL)
        AND (rE.Country = @Country OR @Country IS NULL)
        AND (rE.Language = @Language OR @Language IS NULL)
        AND (d.FirstName = @DirectorFirstName OR @DirectorFirstName IS NULL)
        AND (d.LastName = @DirectorLastname OR @DirectorLastName IS NULL)
```

```
GROUP BY g.Title, g.Year, g.Runtime, g.ContentRating, d.FirstName, d.LastName, rE.Language,
rE.Country, aI.AspectRatio, aI.Color, f.Budget, f.Gross, r.IMDBscore, r.VotesCount
GO
```

Query 3 (Question Type): Search for movies based on ratings. This query takes a range of ints, then searches all of the movies and returns the movies that have an IMBd rating within that range.

```
CREATE PROCEDURE GP.RatingRangeSearch
  @MinRating INT,
  @MaxRating INT
AS
SELECT DISTINCT M.Title, R.IMDBscore, F.Gross, F.Budget, REG.Country, REG.Language
FROM GP.Rating R
        INNER JOIN GP.Movies M ON M.MovieID = R.MovieID
        INNER JOIN GP.Actor A ON A.MovieID = R.MovieID
        INNER JOIN GP.Director D ON D.MovieID = R.MovieID
        INNER JOIN GP.Genre G ON G.MovieID = R.MovieID
        INNER JOIN GP.Region REG ON REG.MovieID = R.MovieID
        INNER JOIN GP.Financial F ON F.MovieID = R.MovieID
        INNER JOIN GP.AdditionalInfo I ON I.MovieID = R.MovieID
WHERE R.IMDBscore BETWEEN @MinRating AND @MaxRating
ORDER BY R.IMDBscore DESC;
GO
```

Query 4 (Report Type): Director financial information based on name. This query takes a director's name, and returns all of the financial information for all of the movie's that the director was a part of.

```
CREATE PROCEDURE GP.DirectorGross
	@DirectorFirstName NVARCHAR(50),
	@DirectorLastName NVARCHAR(50)
AS
SELECT d.FirstName, d.LastName, FORMAT(SUM(F.Gross), '##,##0') AS Gross,
FORMAT(SUM(F.Budget), '##,##0') AS Budget, FORMAT(SUM(F.Gross - F.Budget), '##,##0') AS Profit
FROM GP.Financial F
	INNER JOIN GP.Director D ON D.MovieID = F.MovieID
WHERE D.FirstName = @DirectorFirstName AND D.LastName = @DirectorLastName
GROUP BY D.FirstName, D.LastName
GO
```

Query 5 (Select Type): Find director information based on genre/other topics. This returns all of the director information that matches all of the parameters. This is similar to the MovieSearch procedure in that it is very flexible.

```
CREATE PROCEDURE GP.DirectorSearch
	@MovieTitle nvarchar(MAX) = null,
	@Genre nvarchar(50) = null,
	@Country nvarchar(100) = null,
	@Language nvarchar(100) = null,
	@DirectorFirstName nvarchar(100) = null,
	@DirectorLastName nvarchar(100) = null
AS
Select d.FirstName, d.LastName, m.Title
FROM GP.Movies m
	INNER JOIN GP.Genre g on g.MovieID = m.MovieID
	INNER JOIN GP.Region rE on rE.MovieID = m.MovieID
```

```
        INNER JOIN GP.AdditionalInfo aI on aI.MovieID = m.MovieID
        INNER JOIN GP.Financial f on f.MovieID = m.MovieID
        INNER JOIN GP.Rating r on r.MovieID = m.MovieID
        INNER JOIN GP.Director d on d.MovieID = m.MovieID
WHERE
        (m.Title = @MovieTitle OR @MovieTitle IS NULL)
                AND (rE.Country = @Country OR @Country IS NULL)
                AND (rE.Language = @Language OR @Language IS NULL)
                AND (d.FirstName = @DirectorFirstName OR @DirectorFirstName IS NULL)
                AND (d.LastName = @DirectorLastname OR @DirectorLastName IS NULL)
                AND (g.Genre = @Genre OR @Genre IS NULL)
GROUP BY d.FirstName, d.LastName, m.Title
```

Query 6 (Report Type): Find the financial information for the genre. Report type. This query takes a genre as the parameter and compiles all of the financial information for each movie and presents the user with the different financial information aggregated for all of those movies

```
CREATE PROCEDURE GP.GenreGross
        @GenreName NVARCHAR(50)
AS
BEGIN
        SELECT FORMAT(SUM(F.Gross), '##,##0') AS GenreGross, FORMAT(SUM(F.Budget),
'##,##0') as GenreBudget, FORMAT(SUM(F.Gross-F.Budget), '##,##0') AS GenreProfit
        FROM GP.Financial F
                INNER JOIN GP.Genre G ON G.MovieID = F.MovieID
        WHERE G.Genre = @GenreName
        GROUP BY Genre
```

```
END
```

Query 7 (Report Type): Top 100 Profitable Movies. This query takes no parameters, and returns with a list of the top 100 profitable movies.

```
CREATE PROCEDURE GP.TopMovieProfit_100
AS
SELECT TOP 100 SUM(F.Gross-F.Budget) AS Profit, M.Title
FROM GP.Financial F
        INNER JOIN GP.Movies M ON M.MovieID = F.MovieID
GROUP BY M.Title
ORDER BY Profit DESC
GO
```

Query 8 (Select Type): Get MovieID. This query takes no parameters and returns with the count of ID's in the movie database. This is then used to insert the user data into the database and this value is used as the movieID.

```
CREATE PROCEDURE GP.GetMovieID
AS
SELECT COUNT(m.MovieID) as MovieCount
FROM GP.Movies m
GO
```

Query 9 (Question Type): Find list of movies that an actor has appeared in. This query takes the parameters of the actors first and last name and returns with a list of all movies that actor is a part of.

```
CREATE PROCEDURE GP.ActorSearch
        @FirstName NVARCHAR(50),
        @LastName NVARCHAR(50)
```

```
AS
SELECT M.Title as MovieTitle
FROM GP.Movies M
        INNER JOIN GP.Actor a ON a.MovieID = M.MovieID
WHERE A.FirstName = @FirstName AND A.LastName = @LastName;
GO
```

Query 10 (Select Type): Find a list of movies that match the desired input. Just like the other two versatile queries, this takes many parameters and returns a list of movies with all of the movie data where it matches all of the parameters. You'll notice that some of the parameters are commented out, this is because the queries were not returning with the correct things with these. Future work would include fixing these fields.

```
CREATE PROCEDURE GP.GeneralSearch
        @MovieTitle nvarchar(MAX) = null,
        @Genre nvarchar(50) = null,
        @Country nvarchar(100) = null,
        @Language nvarchar(100) = null,
        @DirectorFirstName nvarchar(100) = null,
        @DirectorLastName nvarchar(100) = null,
        @Actor1FirstName nvarchar(100) = null,
        @Actor1LastName nvarchar(100) = null,
        @Actor2FirstName nvarchar(100) = null,
        @Actor2LastName nvarchar(100) = null,
        @Actor3FirstName nvarchar(100) = null,
        @Actor3LastName nvarchar(100) = null,
        @MinRating INT = null,
        @MaxRating INT = null,
```

```
    @Popularity INT = null,
    @Color nvarchar(100) = null,
    @Budget INT = null,
    @Gross INT = null,
    @AspectRatio DECIMAL = null
AS
Select m.Title as Title
FROM GP.Movies M
    INNER JOIN GP.Genre g on g.MovieID = M.MovieID
    INNER JOIN GP.Director d on d.MovieID = M.MovieID
    INNER JOIN GP.Region rE on rE.MovieID = M.MovieID
    INNER JOIN GP.AdditionalInfo aI on aI.MovieID = M.MovieID
    INNER JOIN GP.Financial f on f.MovieID = M.MovieID
    INNER JOIN GP.Rating r on r.MovieID = M.MovieID
    INNER JOIN GP.Actor a on a.MovieID = M.MovieID
WHERE
    (m.Title = @MovieTitle OR @MovieTitle IS NULL)
    AND (g.Genre = @Genre OR @Genre IS NULL)
    AND (rE.Country = @Country OR @Country IS NULL)
    AND (rE.Language = @Language OR @Language IS NULL)
    AND (d.FirstName = @DirectorFirstName OR @DirectorFirstName IS NULL)
    AND (d.LastName = @DirectorLastName OR @DirectorLastName IS NULL)
    AND (a.FirstName = @Actor1FirstName OR @Actor1FirstName IS NULL)
    AND (a.LastName = @Actor1LastName OR @Actor1LastName IS NULL)
    AND (a.FirstName = @Actor2FirstName OR @Actor2FirstName IS NULL)
    AND (a.LastName = @Actor2LastName OR @Actor2LastName IS NULL)
    AND (a.FirstName = @Actor3FirstName OR @Actor3FirstName IS NULL)
```

```
        AND (a.LastName = @Actor3LastName OR @Actor3LastName IS NULL)
        --AND (r.IMDBscore BETWEEN @MinRating AND @MaxRating)
        --AND (r.VotesCount = @Popularity OR @Popularity IS NULL)
        AND (aI.Color = @Color OR @Color IS NULL)
        --AND (f.Budget = @Budget OR @Budget IS NULL)
        --AND (f.Gross = @Gross OR @Gross IS NULL)
        --AND (aI.AspectRatio = @AspectRatio OR @AspectRatio IS NULL)
GROUP BY m.Title
GO
```

Query 11 (Report Type): Director Count. This query takes the director's name as a parameter and returns with the count of all of the movies that they appear in. The output for this appears in the fourth screenshot of the previous section.

```
CREATE PROCEDURE GP.DirectorCount
        @DirectorFirstName NVARCHAR(100),
        @DirectorLastName NVARCHAR(100)
AS
BEGIN
        SELECT COUNT(DISTINCT M.MovieID) AS MovieCount
        FROM GP.Movies M
                INNER JOIN Gp.Director D ON D.MovieID = M.MovieID
        WHERE D.FirstName = @DirectorFirstName AND D.LastName = @DirectorLastName
END
```

Query 12 (Report Type): Find financial information for a movie. This query takes the movie title as a parameter and returns all of the financial information for that movie, similar to the genre and director query.

```
CREATE PROCEDURE GP.MovieProfit
	@Title NVARCHAR(50)
AS
BEGIN
	SELECT FORMAT(SUM(F.Gross), '##,##0') AS MovieGross, FORMAT(SUM(F.Budget), '##,##0')
	as MovieBudget, FORMAT(SUM(F.Gross-F.Budget), '##,##0') AS MovieProfit
	FROM GP.Movies M
		INNER JOIN GP.Financial F ON F.MovieID = M.MovieID
	WHERE M.Title = @Title
	GROUP BY F.Budget, M.Title
	ORDER BY F.Budget DESC
END
GO
```

Summary and Discussion

While we were able to implement most of the functionality we originally wanted to implement, not all parts of the front end and SQL queries were complete in time. Our finished product ended up being quite different from the original vision in pretty much all aspects. Originally our application was supposed to simulate a functional database for a movie theatre, however, due to very limited availability of the datasets with necessary structure that had to change. Yet, even after we changed the direction for our project, it was still not complete in its final form. The lack of public free data about movie theaters made it extremely difficult to structure tables and an application around the idea. After careful discussion and research, we discovered the IMDB api which gave us csv files of movie data. However, the data was very convoluted and difficult to parse. As such,

finding clean and efficient data was difficult to come by. We had to restructure our table design and application design to fit the data being used. Overall, we tried to communicate as much as possible to coordinate with design and implementation issues while also coordinating each person's role and job in the team.

Changes

Our database design changed significantly since the initial proposal. It was decided to drop the whole structure of the dynamic movie theatre screenings as we were simply not able to find any suitable datasets for such structure. On top of that, due to minimum requirements of the amount of data entries it would just no longer make sense to simulate a movie theatre. A very limited number of movies are usually being run at the movie theatre. Even considering the variation in showtimes, the amount of data entries generated would not meet the requirements of the project. Therefore, it was decided to create an application with similar functionality to IMDB and provide a substantial amount of details about each individual movie. The idea was to create an admin side of the application and the user side, both functioning differently. On the admin side it would be possible to insert, update, delete and search data with advanced parameters. On the user side, it would be possible to search for movies through defined criteria such as movie title, director name, actor name, budget, rating and more.

Thoughts

If we had to change anything about this project, it might be that we start earlier than later. With the very little time we had, it was difficult to piece together both the sql side and c# side of the application to create a moderately functioning program. What we've

learned overall as a team is that we need to put more time into applications like this as it's a bit more robust and requires more finagiling and testing to insure a fully functional project. Something else we would change is possibly the topic. Without a robust library of publicly free data about films in a clean format, the project was made mildly more difficult since parsing convoluted data can become time consuming and extremely frustrating. It would have possibly been better too look at what data sources were widely available and possibly structure our project around this.

Future Work & Improvements

The biggest limitation of this project is the hidden queries. We have two very versatile queries that can find movies based on many different parameters, but to run other queries it's necessary to know how to run those specific queries. In future work, we would change the GUI to work better with the different types of queries that we have. Another future project would be to allow users to login to the application to add reviews for the movies or their own personal opinions about a film and allow them to add movies to their favorites or create lists they can share. A possible improvement could be implementing a full text search procedure that allows the user to have an almost Google-like experience when searching for a movie.