**Final Project: Movie and TV Show Streaming Service Database Project**

By Team Spongebob: Jasper Yeung, Duong Ha, Caitlin Tran, Kenny Leung

# 

**Table of Contents**

[1. Introduction: 3](#_Toc184843024)

[2. Entity Relationship Diagram 4](#_Toc184843025)

[2.1 Design Decisions: 4](#_Toc184843026)

[3. Relational Model 5](#_Toc184843027)

[3.1 Domain: 7](#_Toc184843028)

[3.2 Assumptions: 10](#_Toc184843029)

[3.3 Design Decisions: 10](#_Toc184843030)

[4. Use Case Queries: 11](#_Toc184843031)

[5. Tooling Assessment 12](#_Toc184843032)

[5.1 DBMS: MySQL 12](#_Toc184843033)

[5.2 UI: HTML, Javascript 12](#_Toc184843034)

[5.3 Database Hosting: Microsoft Azure 13](#_Toc184843035)

[5.4 IDE: Visual Studio Code 13](#_Toc184843036)

[5.5 Visual Database Design Tool: MySQL Workbench 13](#_Toc184843037)

[5.6 Mock Data Creation: Mockaroo 13](#_Toc184843038)

[6. Data Sourcing 13](#_Toc184843039)

[7. Data Processing 14](#_Toc184843040)

[8. Code and Testing 14](#_Toc184843041)

[9. Normalization 21](#_Toc184843042)

[9.1 Functional Dependencies: 22](#_Toc184843043)

[9.1.1 STREAMINGSERVICES: 22](#_Toc184843044)

[9.1.2 TVSHOWS: 22](#_Toc184843045)

[9.1.3 MOVIES: 23](#_Toc184843046)

[9.1.4 STREAMINGSERVICETVSHOWS: 23](#_Toc184843047)

[9.1.5 STREAMINGSERVICEMOVIES: 23](#_Toc184843048)

[9.1.6 SEASONS: 24](#_Toc184843049)

[9.1.7 EPISODES: 24](#_Toc184843050)

[9.1.8 STUDIOS: 24](#_Toc184843051)

[9.1.9 TVSHOWGENRES: 25](#_Toc184843052)

[9.1.10 MOVIEGENRES: 25](#_Toc184843053)

[9.1.11 TVSHOWPRODUCEDBY: 25](#_Toc184843054)

[9.1.12 MOVIEPRODUCEDBY: 25](#_Toc184843055)

[9.1.13 TVSHOWAUDIOLANGUAGEOPTIONS: 25](#_Toc184843056)

[9.1.14 MOVIEAUDIOLANGUAGEOPTIONS: 26](#_Toc184843057)

[10. Database Design and Results 26](#_Toc184843058)

[10.1 Retrieve Operations 26](#_Toc184843059)

[10.2 Creation Operations 36](#_Toc184843060)

[10.4 Update Operations 39](#_Toc184843061)

[10.5 Delete Operation 42](#_Toc184843062)

[11 Postmortem Review 43](#_Toc184843063)

[11.1 Effort and Contributions 43](#_Toc184843064)

[11.2 Challenges and Feedback 44](#_Toc184843065)

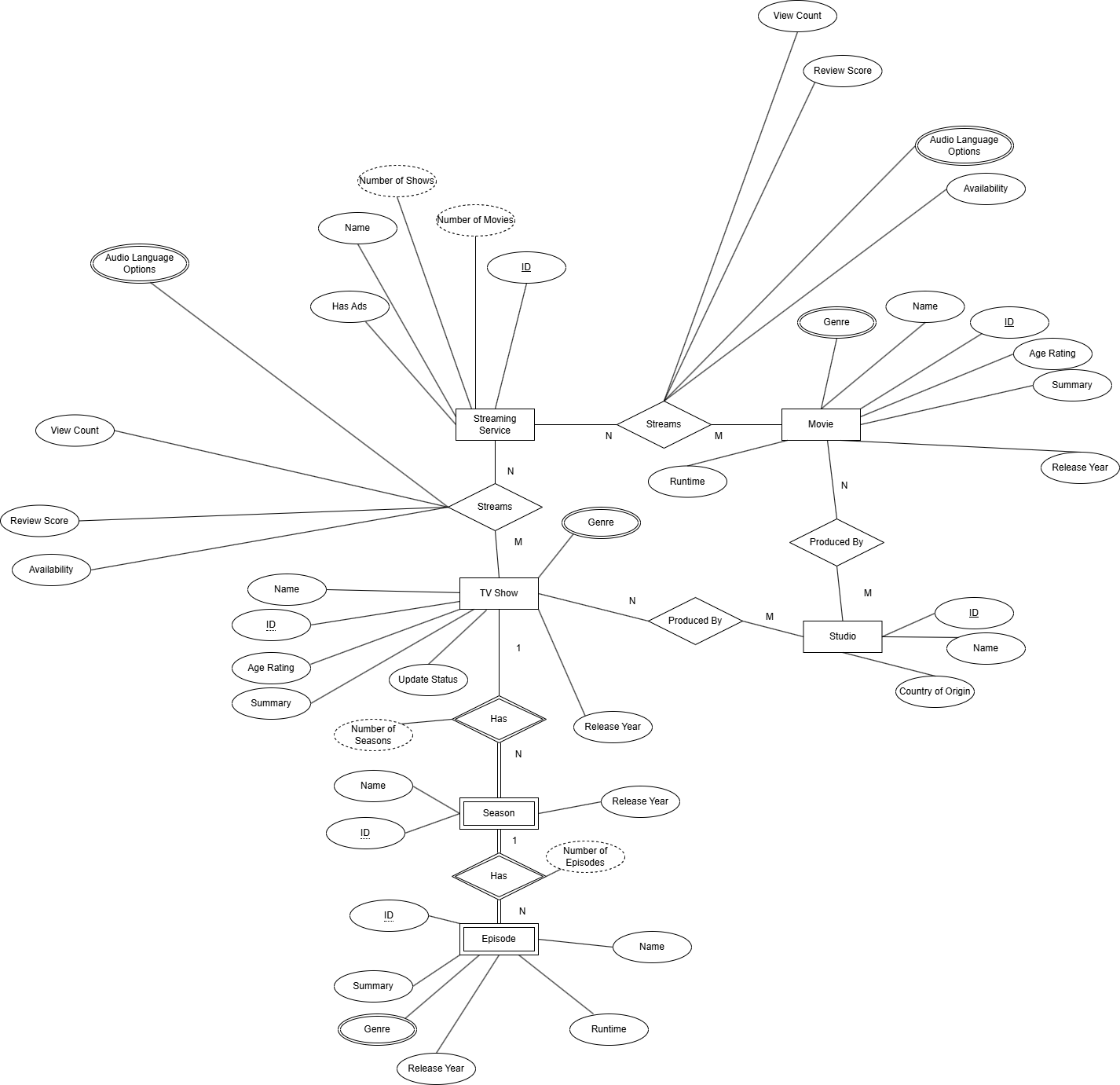
[12. Previous Iteration Updates: 45](#_Toc184843066)

[13. Sources 46](#_Toc184843067)

# 1. Introduction:

The purpose of this project is to create a database that organizes and manages information on TV Shows, movies, and their availability across multiple streaming platforms. In today’s media-rich environment, where numerous streaming services compete for viewer attention, it can be challenging for users to navigate and find content that fits their interests. By structuring data on various media types and relevant categories, this database will allow for efficient information retrieval, and help users discover content based on genre, streaming service availability, review scores, and more. Our database will include key tables such as streaming services, TV shows, movies, seasons, episodes, studios, genres, etc. By incorporating various categories and relationships between tables, this database will be a valuable tool for users, researchers, and businesses to keep insight into the media landscape.

# 2. Entity Relationship Diagram



*Figure 1: ER Diagram for Movie and TV Show Streaming Service Database*

## 2.1 Design Decisions:

* The attributes Number of Movies, TV Shows, Seasons, and Episodes are all derived attributes that would be retrieved through queries and not actually stored in the database.
* TV shows and movies can have more than one genre so they have multivalued attributes.
* TV shows and movies may have different review ratings from different streaming services so review ratings are a relationship attribute.
* Streaming services may not have all language options available for a given TV show or movie. As such, the language options are a multi-value relationship attribute of Streaming services streaming TV shows/movies.
* TV shows and movies have a rating system of multiple categories, so they are shown as a multivalued attribute.
* A TV show and movie may be available at one time on a given streaming service but then be unavailable to view. The data of an unavailable TV show or movie still needs to be stored so availability is a relationship attribute of streaming service streams TV show/movie.
* Seasons and episodes are in a weak relation with TV shows and have a relation with each other.
* TV Shows and movies have multiple audio language options, so they are shown as a multivalued attribute.
* Entity Movies and TV shows are linked to a studio. It allows users to search by studio.

# 3. Relational Model

*Figure 2: Relational Model for Movie and TV Show Streaming Service Database*

**StreamingServices**

|  |  |  |
| --- | --- | --- |
| **SS\_ID** | **SSName** | **HasAds** |

**TVShows**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TVS\_ID** | **TVSName** | **TVSReleaseYear** | **TVSSummary** | **TVAgeRating** | **TVSUpdateStatus** |

**Movies**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **MV\_ID** | **MVName** | **MVReleaseYear** | **MVSummary** | **MVAgeRating** | **MVRuntime** |

**StreamingServiceTVShows**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SS\_ID** | **TVS\_ID** | **TVSViewCount** | **TVSReviewScore** | **MVAvailability** |

**StreamingServiceMovies**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SS\_ID** | **MV\_ID** | **MVViewCount** | **MVReviewScore** | **MVAvailability** |

**Seasons**

|  |  |  |  |
| --- | --- | --- | --- |
| **TVS\_ID** | **SE\_ID** | **SEName** | **SEReleaseYear** |

**Episodes**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TVS\_ID** | **SE\_ID** | **EP\_ID** | **EPName** | **EPReleaseYear** | **EPSummary** | **EPRuntime** |

**Studios**

|  |  |  |
| --- | --- | --- |
| **ST\_ID** | **STName** | **CountryOfOrigin** |

**TVShowProducedBy**

|  |  |
| --- | --- |
| **TVS\_ID** | **ST\_ID** |

**MovieProducedBy**

|  |  |
| --- | --- |
| **MV\_ID** | **ST\_ID** |

**TVShowGenres**

|  |  |
| --- | --- |
| **TVS\_ID** | **TVSGenre** |

**MovieGenres**

|  |  |
| --- | --- |
| **MV\_ID** | **MVGenre** |

**TVShowAudioLanguageOptions**

|  |  |  |
| --- | --- | --- |
| **TVS\_ID** | **SS\_ID** | **TVSLanguage** |

**MovieAudioLanguageOptions**

|  |  |  |
| --- | --- | --- |
| **MV\_ID** | **SS\_ID** | **MVLanguage** |

## 3.1 Domain:

*Figure 3: Table of Attributes Domain for the Relational Model*

|  |  |
| --- | --- |
| DString | A set of all strings |
| DAString | A set of all alphabetic strings |
| DNumber | A non-negative integer |
| DID | A non-negative, non-zero integer of length 1 to 9 |
| DName | A string of max 64-character length |
| DBool | A bool of true or false |
| DYear | A 4-digit number of a valid year up to current year (2024) |
| DParagraph | A string of max 250-character length |
| DDuration | A duration in terms of minutes |
| DDecimal | A non-negative decimal with a format of up to 2 decimal places |
| DCountry | A string following alpha-3 syntax of uppercase alphabetic characters of length 3 associated with a country |
| **Movies** | |
| dMV\_ID | dMV\_ID ∈ DID that is not null |
| dMVName | dMVName ∈ DName that is not null |
| dMVReleaseYear | dMVReleaseYear ∈ DYear that is not null |
| dMVRuntime | dMVRuntime ∈ DDuration that is not null and not 0 |
| dMVSummary | dMVSummary ∈ DParagraph that is not null |
| dMVAgeRating | dMVAgeRating ∈ DString that is not null and has values associated with the movie age ratings from the Motion Picture Association: ‘G’, ‘PG’, ‘PG-13’, ‘R’, or ‘NC-17’ |
| **TVSHOWS** | |
| dTVS\_ID | dTVS\_ID ∈ DID that is not null |
| dTVSName | dTVSName ∈ DName that is not null |
| dTVSSummary | dTVSSummary ∈ DParagraph that is not null |
| dTVSReleaseYear | dTVSReleaseYear ∈ DYear that is not null |
| dTVAgeRating | dTVAgeRating ∈ DString that is not null and has values associated with the TV age ratings from the TV Parental Guidelines Monitoring Board: ‘TV-Y’, ‘TV-Y7’, ‘TV-G’, ‘TV-PG’, ‘TV-14’, or ‘TV-MA’ which may also have none or combinations of ‘D’, ‘L’, ‘S’, ‘V’, ‘FV’ at the end |
| dUpdateStatus | dUpdateStatus ∈ DAString that is not null and has values of either ‘Complete’, ‘Ongoing’, or ‘Canceled’ |
| **SEASON** | |
| dSE\_ID | dSE\_ID ∈ DID that is not null |
| dSEName | dSEName ∈ DName that is not null |
| dSEReleaseYear | dSEReleaseYear ∈ DYear that is not null |
| **EPISODES** | |
| dEP\_ID | dEP\_ID ∈ DID that is not null |
| dEPName | dEPName ∈ DName that is not null |
| dEPReleaseYear | dEPReleaseYear ∈ DYear that is not null |
| dEPSummary | dEPSummary ∈ DParagraph that is not null |
| dEPRuntime | dEPRuntime ∈ DDuration that is not null and not 0 |
| **STREAMINGSERVICES** | |
| dSS\_ID | dSS\_ID ∈ DID that is not null |
| dSSName | dSSName ∈ DName that is not null |
| dHasAds | dHasAds ∈ DBoo that is not null |
| **STREAMINGSERVICETVSHOWS** | |
| dTVSViewCount | dTVSViewCount ∈ DNumber that is not null |
| dTVSReviewScore | dTVSReviewScore ∈ DDecimal that is not null and has a value range of 0 to 10 inclusive |
| dTVSAvailability | dTVSAvailability ∈ DBool that is not null |
| **STREAMINGSERVICEMOVIES** | |
| dMVViewCount | dMVViewCount ∈ DNumber that is not null |
| dMVReviewScore | dMVSReviewScore ∈ DDecimal that is not null and has a value range of 0 to 10 inclusive |
| dMVAvailability | dMVAvailability ∈ DBool that is not null |
| **STUDIOS** | |
| dST\_ID | dST\_ID ∈ DID that is not null |
| dSTName | dSTName ∈ DName that is not null |
| dCountryOfOrigin | dCountryOfOrigin ∈ DCountry that is not null |
| **MOVIEGENRES** | |
| dMVGenre | dMVGenre ∈ DAString that is not null and has a value associated with a media genre |
| **TVSHOWGENRES** | |
| dTVSGenre | dTVSGenre ∈ DAString that is not null and has a value associated with a media genre |
| **MOVIEAUDIOLANGUAGEOPTIONS** | |
| dMVLanguage | dMVLanguage ∈ DAString that is not null and has a value associated with a spoken language |
| **TVSHOWAUDIOLANGUAGEOPTIONS** | |
| dTVSLanguage | dTVSLanguage ∈ DAString that is not null and has a value associated with a spoken language |

## 3.2 Assumptions:

* Not all streaming services may have all audio language options for a particular TV show or movie
* TV show’s genre will also describe the episode genres. Edge cases such as a TV show with all different episode genres would be categorized as ‘Anthology’ along with individual episode genres.
* Shows available to stream at one point on a streaming service may then be unavailable but the only data that changes would be its availability.
* Assumed that while not all streaming services use the TV Parental Guidelines Monitoring Board’s age rating system, there are equivalents to individual cases such as ‘Mature’ = ‘TV-MA’
* Streaming service information is based on the North American region. This means TV shows and movies in the table are ones licensed to the North American region.

## 3.3 Design Decisions:

* Movies and TV shows have separate rating systems which is why TVAgeRating and MVAgeRating have different domains.
* The amount of categories for the age rating is minimal, so it is allowed as an attribute of the movie and TV show table instead of a separate table.
* There are two tables called Streaming Service TV Shows and Streaming Service Movies because they are two separate relations with different foreign keys.
* TV Show Produced By has a unique superkey of TVS\_ID and ST\_ID where there can only be a single combination of the two foreign keys. Even if a TV show was produced by multiple studios, there will be a unique superkey.
* Movie Produced By has a unique superkey of MV\_ID and ST\_ID where there can only be a single combination of the two foreign keys. Even if a movie was produced by multiple studios, there will be a unique superkey.
* Genres is a multivalued attribute in the ER diagram. When mapped to RM, two relations of TV Show Genres and Movie Genres were created.
* TV shows can have multiple genres so in the TV Show Genres relation, the superkey of TVS\_ID and Genre have a unique combination because an instance of a TV show cannot have a duplicate genre in the table.
* Movies can have multiple genres so in the Movie Genres relation, the superkey of MV\_ID and Genre have a unique combination because an instance of a movie cannot have a duplicate genre in the table.
* Audio Language Options attribute of streaming service streams TV show / movie are separate relation from Streaming Service TV Shows and Streaming Service Movies because it is a multivalued attribute in the ER diagram.
* In TV Show Audio Language Options the superkey of TVS\_ID, SS\_ID, and Language have a unique combination. Even if there are multiple tuples with the same TVS\_ID SS\_ID combination, the Language would be different. Similarly, even if TVS\_ID and Language were the same SS\_ID would be different because the show with that language was streamed on a different streaming service.
* In Movie Audio Language Options the superkey of MV\_ID, SS\_ID, and Language have a unique combination. Even if there are multiple tuples with the same MV\_ID SS\_ID combination, the Language would be different. Similarly, even if MV\_ID and Language were the same SS\_ID would be different because the movie in that language was streamed on a different streaming service.

# 4. Use Case Queries:

There are two user roles that may use our database system: Customers and Administrators. Customers encompass the general public of end users who want to find information on movies, TV shows, and streaming services. Customers also include researchers who may want information on trends and data about movies and TV shows on various streaming services.

Some queries that a customer might want to answer are:

* What streaming services is the TV Show HOUSE available on
* What shows have been removed by Netflix
* Sort movies on Disney+ by year
* Popular shows by genre on Netflix
* How many shows and movies are available on a streaming service
* What streaming service has the most TV shows and movies
* What is the highest-rated TV show on Peacock
* What movies have Japanese as an audio language option on Netflix
* The best-rated movie in each genre

These queries are for the purpose of determining what shows and movies a user might want to watch and the information that they may want to know in order to pick. Customers may also have specific shows/movies they want to find or may have access to a specific streaming service and want to find shows available there.

Administrators are the ones who will insert, delete, and update the data of the system to maintain data quality.

Some queries that an administrator may want to use are:

* Update view counts, review scores, and availability of a tv show or movie on a streaming service
* Delete a streaming service from the database and its related data
* Insert new movie into the database
* Insert a new TV show into the database
* Insert a new streaming service into the database.

# 5. Tooling Assessment

## 5.1 DBMS: MySQL

For the database management system, we will use MySQL for our project since its syntax is similar to SQLite and it’s free to use. It also has efficient data management and fast query performance, which is well-suited for the scope of this project. Finally, it integrates well with various tech stacks, making it versatile for many types of applications.

## 5.2 UI: HTML, Javascript

We use HTML, and Javascript to develop a user-friendly interface for the streaming service databases that allow users to search information about TV shows and movies. We will use Node.js to connect the UI to MySQL which facilitates efficient data transactions. Additionally, some of us have previous experience in HTML, and Javascript which makes development easier compared to using other unfamiliar languages.

## 5.3 Database Hosting: Microsoft Azure

For our project we will be using Microsoft Azure as our hosting service. The first reason we are doing so is that Azure readily supports MySQL databases and has a feature dedicated to MySQL. Furthermore, Azure offers SQL database and static web app hosting always free to use. Finally, Azure offers many additional tools if we need them that are free for 12 months which is a time frame where we can use the tools to complete our project. We will also be using VS Code as our IDE which can be connected to Azure for ease of development.

## 5.4 IDE: Visual Studio Code

For our IDE, we’ve decided to use VS Code. VS Code supports MySQL, HTML, and Javascript and has additional extensions that could help streamline and optimize our project. We also have some experience using VS Code from previous classes that require it, so utilizing it should come with little hassle.

## 5.5 Visual Database Design Tool: MySQL Workbench

For our project, we realized that connecting to the database hosted on Azure was lacking when it came to finding extensions to do so in MySQL and was not very intuitive to show changes and queries made in the database. As such we decided to use MySQL Workbench which is designed specifically for creating and maintaining MySQL databases and allows connections to the database server on Azure.

## 5.6 Mock Data Creation: Mockaroo

For our project, we didn’t initially use Mockaroo to generate data for our database. However, we realized we needed much more data to showcase the project so we used Mockaroo to generate around 50 to 100 new rows depending on the table.

# 6. Data Sourcing

The team initially manually obtained the population data by looking through various streaming service websites and IMDB pages to obtain accurate information. This yielded conservatively about 10 to 20 rows for each table in our database. After getting feedback from the professor that our database requires more data, we decided to use Mockaroo to create Excel datasets for each individual table.

# 7. Data Processing

The initial population data processed datasets manually into SQL INSERT statements which took a significant amount of time. After getting feedback and switching to Mockaroo to populate the database, we used those Excel files and automated the cells to create the INSERT statements with the data in the row. This, however, presented a problem for special string characters like single quotes which had to be handled manually by adding a forward slash character. All data was then inserted through MySQL workbench into the database hosted on Microsoft Azure.

# 8. Code and Testing

*Figure 4: Test Queries for STREAMINGSERVICES Table*

**Correct:**

INSERT INTO STREAMINGSERVICES VALUE (100, 'Netflix', True);

INSERT INTO STREAMINGSERVICES VALUE (108, 'Crunchyroll', False);

**Incorrect:**

INSERT INTO STREAMINGSERVICES VALUE (108, ‘Hulu’, 3);

INSERT INTO STREAMINGSERVICES VALUE (-109, NULL, NULL);

In figure 4, the top two statements are correct while the bottom violates the key and integrity constraints. The SS\_ID is a key attribute meaning it cannot be duplicated. The HasAds attribute is a Boolean and so having the value 3 is not valid making it an integrity constraint. HasAds and SSName also cannot be null which violates the integrity constraint. In the second incorrect statement, the SS\_ID is negative which violates the domain constraint needing to be a non-negative integer.

*Figure 5: Test Queries for TVSHOWS Table*

**Correct:**

INSERT INTO tvshows Values (201, 'The Last Airbender', 2005, 'An Avatar named Aang travels around the world to master elemental powers and stop the Fire Nation', 'TV-14', 'Finished');

INSERT INTO TVSHOWS VALUES (202, 'The Boys', 2019, 'A group of vigilantes hunts down corrupted superheroes who abuse their powers', 'TV-MA', 'Ongoing');

INSERT INTO tvshows VALUES (216, 'Star Wars: The Acolyte', 2024, 'A Jedi Warrior investigates a series of crimes that reveal sinister forces underneath', 'TV-14', 'Cancelled');

**Incorrect:**

INSERT INTO TVSHOWS VALUES(-1,'breaking bad', 2008, 'an American crime drama television series', ‘TV-MA', 'Finished)

INSERT INTO TVSHOWS VALUES(216, 'friends', 1994, 'an American television sitcom', ‘TV-14', 'Anticipated')

In Figure 5, the bottom violates the domain and key constraints. TVS\_ID must be greater than 0 which otherwise violates the domain constraint. TVS\_ID is a key attribute meaning that it must be unique which would violate the key constraint. The TVSUpdateStatus has to be the values of Finished, Cancelled, or Ongoing, and if not is otherwise a domain constraint violation.

*Figure 6: Test Queries for MOVIES Table*

**Correct:**

INSERT INTO MOVIES VALUES (301, 'Frozen', 2013, 'A woman named Anna sets out to find her sister to save her kingdom from an everlasting winter', 'PG', 102);

INSERT INTO MOVIES VALUES (302, 'Avengers: Endgame', 2019, 'The remaining Avengers gather to stop Thanos and reverse his actions', 'PG-13', 181);

INSERT INTO MOVIES VALUES (305, 'The Matrix', 1999, 'A man discovers that his world is a simulation', 'R', 136);

INSERT INTO MOVIES VALUES (307, 'The Lion King', 1994, 'A lion cub learns about the circle of life', 'G', 88);

**Incorrect:**

INSERT INTO MOVIES VALUES (307, Tangled, 2010, 'A princess is taken from her tower by a notorious bandit and goes out in search of her way home', 'PG', 100);

INSERT INTO MOVIES VALUES (309, 'The Super Mario Bros. Movie', 2023, 'A plumber travels to the Mushroom Kingdom in hopes of rescuing his brother', 'For Families', 92);

In Figure 6, the correct insertions all pass the key, integrity, and domain constraints while the incorrect insertions do not. The first incorrect insertion violates the key constraint due to 307 already being used as an MV\_ID of another tuple. It also violates the integrity constraint because the MVName is not in quotes making it not a string. The second incorrect insertion violates the domain constraint due to MVAgeRating only allowing values of G, PG, PG-13, R, and NC-17.

*Figure 7: Test Queries for STREAMINGSERVICETVSHOWS Table*

**Correct:**

INSERT INTO STREAMINGSERVICETVSHOWS VALUES (100, 201, 5000000, 9.5, True);

INSERT INTO STREAMINGSERVICETVSHOWS VALUES (109, 204, 2500000, 9.2, False);

**Incorrect:**

INSERT INTO STREAMINGSERVICETVSHOWS VALUES (201, 100, 5000000, 9.5, True);

INSERT INTO STREAMINGSERVICETVSHOWS VALUES (109, 204, 2500000, 9.2, No);

In figure 7, the first incorrect insertion would be rejected because it violates the domain constraint since the bool value only allows True/False and not Yes/No value. The second insertion would also be rejected because it violates the foreign key constraint since it does not reference a valid ID value for streaming services and TV shows.

*Figure 8: Test Queries for STREAMINGSERVICEMOVIES Table*

**Correct:**

INSERT INTO STREAMINGSERVICEMOVIES VALUES (100, 301, 10000000, 8.2, True);

INSERT INTO STREAMINGSERVICEMOVIES VALUES (104, 306, 11000000, 7.8, False);

**Incorrect:**

INSERT INTO STREAMINGSERVICEMOVIES VALUES (999, 999, -10, 11.8, NULL);

INSERT INTO STREAMINGSERVICEMOVIES VALUES (100, 301, 30, 9.0, False);

In figure 8, the first insertions are correct because that are in line with the referential, key, domain, and integrity constraints. The first incorrect insert statement is rejected because 999 is not a value that can be referenced in MV\_ID from MOVIES and SS\_ID from STREAMINGSERVICES which is a referential constraint violation. MVViewCount also cannot be a negative integer with is a domain constraint violation. Similarly, MVReviewScore cannot be negative or greater than 10 which is another domain constraint. MVAvailability cannot be NULL which is an integrity constraint violation. The second insertion is rejected because SS\_ID 100 and MV\_ID 301 are both already a key in the STREAMINGSERVICEMOVIES table which makes it a key constraint violation.

*Figure 9: Test Queries for SEASONS Table*

**Correct:**

INSERT INTO SEASONS VALUES(201, 1, 'Book One: Water', 2005);

INSERT INTO SEASONS VALUES(201, 2, 'Book Two: Earth', 2006);

INSERT INTO SEASONS VALUES(201, 3, 'Book Three: Fire', 2007);

INSERT INTO SEASONS VALUES(202, 1, 'Season 1', 2019);

INSERT INTO SEASONS VALUES(202, 2, 'Season 2', 2020);

**Incorrect:**

INSERT INTO SEASONS VALUES (200, 1, ‘Season 1’, 2021);

INSERT INTO SEASONS VALUES (NULL, 2, ‘Season 2’, 2020)

In figure 9, all of the correct insertions pass the constraints. In contrast, for the incorrect ones, the first violates the referential constraint as there is no record in ‘TVShow’ that has ‘200’ as the primary key. The second violates the integrity constraint as the foreign key value for the TVShows is NULL, which it cannot allow.

*Figure 10: Test Queries for EPISODES Table*

**Correct:**

INSERT INTO EPISODES VALUES (201, 1, 1, 'The Boy in the Iceberg', 2005, 'Katara and Sokka discover Aang in an iceberg', 24);

INSERT INTO EPISODES VALUES (201, 1, 2, 'The Avatar Returns', 2005, 'Aang must prove himself to the villagers', 24);

INSERT INTO EPISODES VALUES (202, 1, 1, 'The Name of the Game', 2019, 'Hughie meets Billy Butcher and learns the truth about superheroes', 55);

**Incorrect:**

INSERT INTO EPISODES VALUES (201, NULL, 3, ‘The Southern Air Temple’, 2005, ‘Aang returns to his home where he learns what happened since he left’, 25);

INSERT INTO EPISODES VALUES (205, 2, 2, ‘Video Killed The Radio Star’, 2024, ‘Alastor’s return shakes up the Vee’s airtime while Angel deals with an annoying guest’, 25);

In figure 10, all of the correct insertions passed, while the incorrect insertions did not. The first violated the integrity constraint, as the foreign key representing Season cannot be NULL, and the second violated the referential constraint as there’s no attribute in Season with a value ‘205, 2’.

*Figure 11: Test Queries for STUDIOS Table*

INSERT INTO STUDIOS VALUES (1, 'Nickelodeon Animation Studio', 'USA');

INSERT INTO STUDIOS VALUES (4, 'Ufotable', 'JPN');

**Incorrect:**

INSERT INTO Studios VALUES (1, 'Studio Ghibli', 'JPN');

INSERT INTO Studios VALUES (5, 'Nickelodeon Animation Studio', 'USA');

INSERT INTO Studios VALUES (-1, 'Another Studio', 'GBR');

In Figure 11, the bottom violates the domain, key, Primary Key and Unique constraint. The ST\_ID must be greater than 0. ST\_Name should be unique, which 'Nickelodeon Animation Studio' already exists. Also, ST\_ID must be unique.

*Figure 12: Test Queries for TVSHOWPRODUCEDBY Table*

**Correct:**

INSERT INTO TVSHOWPRODUCEDBY VALUES (201, 1);

INSERT INTO TVSHOWPRODUCEDBY VALUES (202, 2);

**Incorrect:**

INSERT INTO TVShowProducedBy VALUES (201, 1);

INSERT INTO TVShowProducedBy VALUES (999, 1);

In Figure 12, the bottom violates primary key,and foreign key constraints. TVShowProducedB (1, 'Sci-Fi') already exists. Also, TVS\_ID 999 does not exist in TVShows. violating the foreign key constraint.

*Figure 13: Test Queries for* MOVIEPRODUCEDBY *Table*

**Correct:**

INSERT INTO MOVIEPRODUCEDBY VALUES (301, 7);

INSERT INTO MOVIEPRODUCEDBY VALUES (302, 8);

**Incorrect:**

INSERT INTO MOVIEPRODUCEDBY VALUES (400, 7);

INSERT INTO MOVIEPRODUCEDBY VALUES (302, 100);

In Figure 13, the first insertion would be rejected because it violates the foreign key constraint since it does not reference a valid movie ID. The second insertion would also be rejected because it violates the foreign key constraint since it does not reference a valid ID for the studio value.

*Figure 14: Test Queries for TVSHOWGENRES Table*

**Correct:**

INSERT INTO TVSHOWGENRES VALUES (201, 'Action');

INSERT INTO TVSHOWGENRES VALUES (201, 'Adventure');

INSERT INTO TVSHOWGENRES VALUES (202, 'Drama');

INSERT INTO TVSHOWGENRES VALUES (202, 'Action');

**Incorrect:**

INSERT INTO TVSHOWGENRES VALUES (500, 'Action');

In Figure 14, this insertion would be rejected because it violates the foreign key constraint since it does not reference a valid ID for the TV show.

*Figure 15: Test Queries for MOVIEGENRES Table*

**Correct:**

INSERT INTO MOVIEGENRES VALUES(301, 'Animation');

INSERT INTO MOVIEGENRES VALUES(301, 'Family');

INSERT INTO MOVIEGENRES VALUES(302, 'Action');

**Incorrect:**

INSERT INTO MovieGenres VALUES (301, 'Animation');

INSERT INTO MovieGenres VALUES (999, 'Drama');

In Figure 15, the bottom violates the primary key and foreign key constraints. MVGenre (1, 'Sci-Fi') already exists. Also, There is no movie with MV\_ID 999 in the Movies table, violating the foreign key constraint.

*Figure 16: Test Queries for TVSHOWAUDIOLANGUAGEOPTIONS Table*

**Correct:**

INSERT INTO TVSHOWAUDIOLANGUAGEOPTIONS VALUES(201, 100, 'English');

INSERT INTO TVSHOWAUDIOLANGUAGEOPTIONS VALUES(201, 100, 'Spanish');

INSERT INTO TVSHOWAUDIOLANGUAGEOPTIONS VALUES(202, 101, 'English');

**Incorrect:**

INSERT INTO TVSHOWAUDIOLANGUAGEOPTIONS VALUES(100, 201, 'English' );

INSERT INTO TVSHOWAUDIOLANGUAGEOPTIONS VALUES(201, 100, Yes);

In Figure 16, the incorrect insertion data were rejected as one violates the referential integrity constraint since it does not reference a valid ID value for TV shows and streaming services, while the other violates the domain constraint since it does not reference a valid audio language option.

*Figure 17: Test Queries for MOVIEAUDIOLANGUAGEOPTIONS Table*

**Correct:**

INSERT INTO MOVIEAUDIOLANGUAGEOPTIONS VALUES(301, 100, 'English');

INSERT INTO MOVIEAUDIOLANGUAGEOPTIONS VALUES(301, 100, 'Japanese');

INSERT INTO MOVIEAUDIOLANGUAGEOPTIONS VALUES(302, 101, 'English');

**Incorrect:**

INSERT INTO MOVIEAUDIOLANGUAGEOPTIONS VALUES(100, 301, 'English');

INSERT INTO MOVIEAUDIOLANGUAGEOPTIONS VALUES(301, 100, ‘No’);

In Figure 17, the first insertion would be rejected because it violates the foreign key constraint since it does not reference a valid ID value for movie and streaming services. The second insertion would also be rejected because it violates the domain constraint since it does not contain a valid language option.

# 9. Normalization

We verified all the NFs discussed in class and ensured that each relation in the schema is in 3NF. The attributes in all tables are atomic (1NF). All multi-value attributes in the ER diagram in Iteration 1 were already turned into tables which were MOVIEGENRES, TVSHOWGENRES, TVSHOWPRODUCEDBY, MOVIEPRODUCEDBY, MOVIEAUDIOLANGUAGEOPTIONS, TVSHOWAUDIOLANGUAGEOPTIONS. Additionally, every non-prime attribute in the relations is fully functionally dependent on the primary key, ensuring compliance with 2NF.

STREAMINGSERVICETVSHOWS for example all attributes such as TVSViewCounts and TVSReviewScore are fully functionally dependent on the primary key of SS\_ID and TVS\_ID. There are also no transitive dependencies where a non-prime attribute depends indirectly on the primary key via another non-prime attribute, which satisfies the requirement for 3NF. In our design, the attributes mostly are for the most part entirely functionally dependent on the primary keys with the only exceptions being the genre and streaming-related tables which have a part of the key functionally dependent on a foreign key. This design ensures logical data separation, reduces anomalies, and improves maintainability.

Since the schema is already in 3NF, there would be little gain from further normalization. However, additional normalization could involve storing repeated data in separate tables for highly specific attributes or grouping similar data, which would make the database more efficient. For example, attributes like ViewCount and ReviewScore could be moved into another table for analytics. This change would not affect the structure but could simplify querying and enhance performance for specialized use cases.

## 9.1 Functional Dependencies:

### **9.1.1 STREAMINGSERVICES**:

* FD 1: {SS\_ID} → {SSName}
* FD 2: {SS\_ID} → {HasAds}

The name and ad availability of the streaming services fully depend on the streaming service ID to uniquely identify each service.

### **9.1.2 TVSHOWS**:

* {TVS\_ID} → {TVSName}
* {TVS\_ID} → {TVSReleaseYear}
* {TVS\_ID} → {TVSSummary}
* {TVS\_ID} → {TVAgeRating}
* {TVS\_ID} → {UpdateStatus}

Each TVS\_ID would uniquely identify the name, release year, summary, age rating, and status of each show in the database.

### **9.1.3 MOVIES**:

* {MV\_ID} → {MVName}
* {MV\_ID} → {MVReleaseYear}
* {MV\_ID} → {MVSummary}
* {MV\_ID} → {MVAgeRating}
* {MV\_ID} → {MVRuntime}

Like with TV, MV\_ID uniquely identifies each movie by having it be associated with a specific name, year, summary, age rating, and runtime.

### **9.1.4 STREAMINGSERVICETVSHOWS**:

* {SS\_ID, TVS\_ID} → {TVSViewCount}
* {SS\_ID, TVS\_ID} → {TVSReviewScore}
* {SS\_ID, TVS\_ID} → {TVSAvailability}

This would be in 3NF as SS\_ID and TVS\_ID uniquely identify the attributes on the right side, and there exists no transient dependencies.

### **9.1.5 STREAMINGSERVICEMOVIES**:

* {SS\_ID, MV\_ID} → {MVViewCount, MVReviewScore, MVAvailability}
* {SS\_ID, MV\_ID} → {MVReviewScore}
* {SS\_ID, MV\_ID} → {MVAvailability}

Like with StreamingServiceTVShows, this table is also in 3NF (or perhaps BCNF) as the left side (SS\_ID and MV\_ID) uniquely identifies the right side (view count, review score, and availability) without any transient dependencies.

### **9.1.6 SEASONS**:

* {TVS\_ID, SE\_ID} → {SEName}
* {TVS\_ID, SE\_ID} → {SEReleaseYear}

The names of the season and the year they were released is fully dependent on the IDs of the tv show and season it is associated with.

### **9.1.7 EPISODES**:

* {TVS\_ID, SE\_ID, EP\_ID} → {EPName}
* {TVS\_ID, SE\_ID, EP\_ID} → {EPReleaseYear}
* {TVS\_ID, SE\_ID, EP\_ID} → {EPSummary}
* {TVS\_ID, SE\_ID, EP\_ID} → {EPRuntime}

The IDs of the TV show, season, and episode all uniquely identify the name, release year, summary, and runtime.

### **9.1.8 STUDIOS**:

* {ST\_ID} → {STName}
* {ST\_ID} → {CountryOfOrigin}

The Studios table is in 3NF since all the attributes are fully dependent on the ST\_ID and there are no transitive dependencies.

### **9.1.9 TVSHOWGENRES**:

* {TVS\_ID} → {TVSGenre}

The TvShowGenres table is in 3NF since TVSGenre is fully dependent on TVS\_ID and there are no transitive dependencies.

### **9.1.10 MOVIEGENRES**:

* {MV\_ID} → {MVGenre}

The MovieGenres table is in 3NF since MVGenre is fully dependent on MV\_ID and there are no transitive dependencies.

### **9.1.11 TVSHOWPRODUCEDBY:**

* {TVS\_ID} → {ST\_ID}

The TvShowProducedBy table is in 3NF since ST\_ID is fully dependent on TVS\_ID and there are no transitive dependencies.

### **9.1.12 MOVIEPRODUCEDBY:**

* {MV\_ID} → {ST\_ID}

The MovieProducedBy table is in 3NF since ST\_ID is fully dependent on MV\_ID and there are no transitive dependencies.

### **9.1.13 TVSHOWAUDIOLANGUAGEOPTIONS:**

* {TVS\_ID, SS\_ID} → {TVSLanguage}

The TvShowAuddioLanguageOptions table is in 3NF since TVSLanguage is fully dependent on TVS\_ID and SS\_ID, and there are no transitive dependencies.

### **9.1.14 MOVIEAUDIOLANGUAGEOPTIONS:**

* {MV\_ID, SS\_ID} → {MVLanguage}

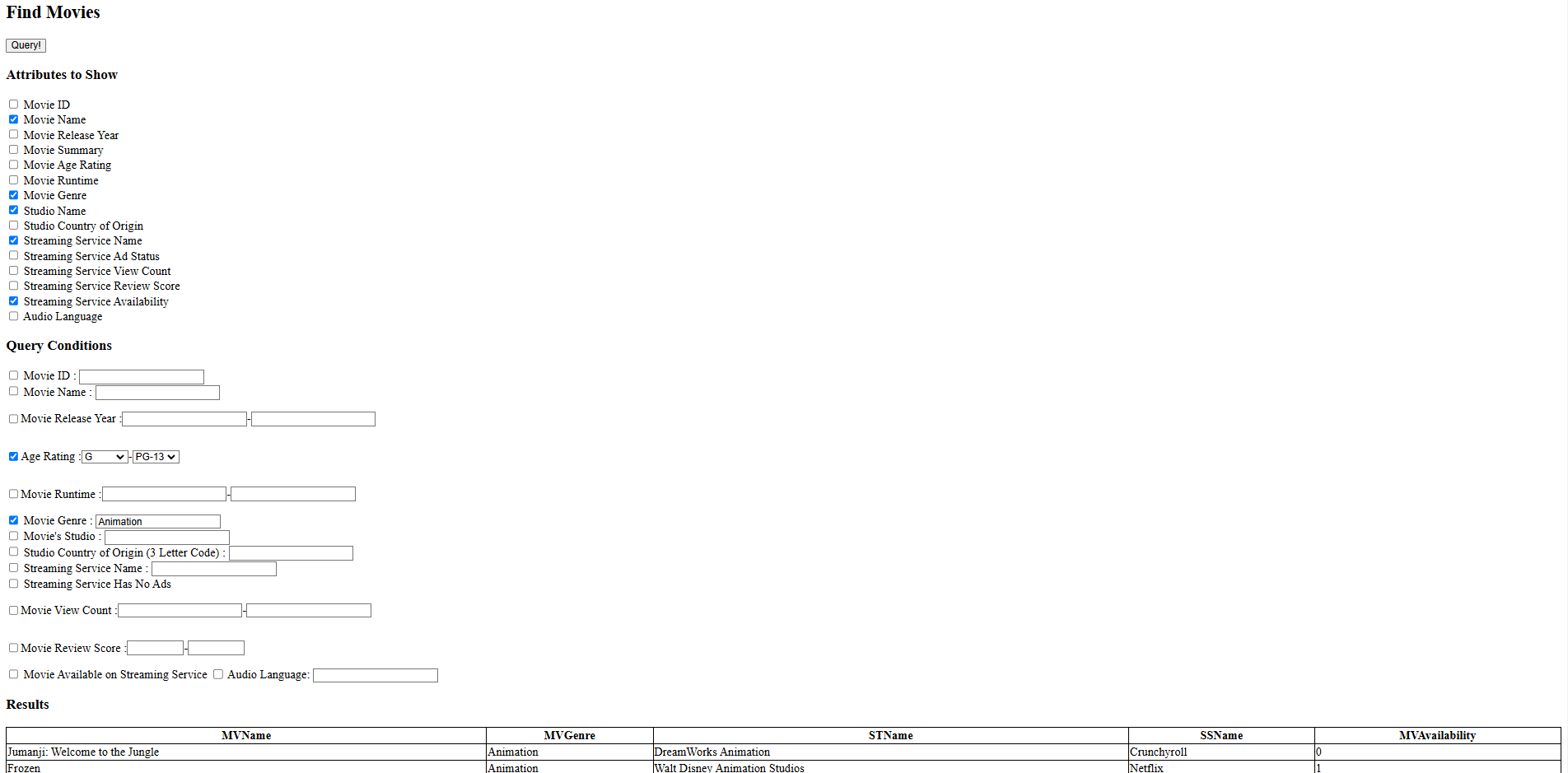
The MovieAudioLanguageOptions table is in 3NF since MVLanguage is fully dependent on MV\_ID and SS\_ID, and there are no transitive dependencies.

# 10. Database Design and Results

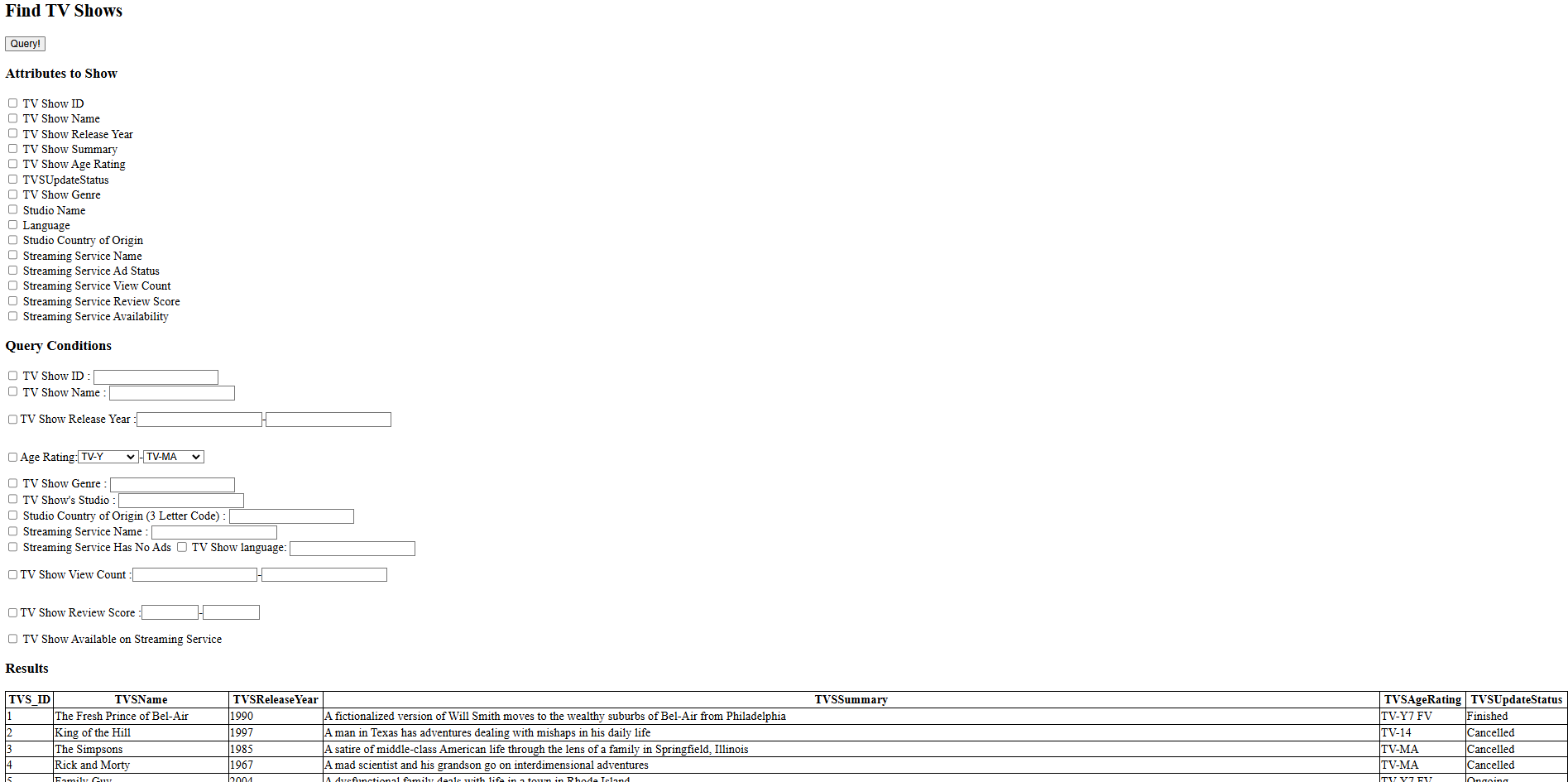
The result of our project was a website page that has four sections focusing on specific queries for information on movies, TV shows, episodes, and streaming services. There were an additional three sections that focused on admin CRUD functions which were the data inserts, updates, and delete operations. The website and database are both hosted on Microsoft Azure. A design priority for the UI in this project was allowing users to customize queries in many combinations of conditions to find TV shows and movies they are looking for. The specific use cases and implementations of each part of the UI will be discussed in the following sections.

## 10.1 Retrieve Operations

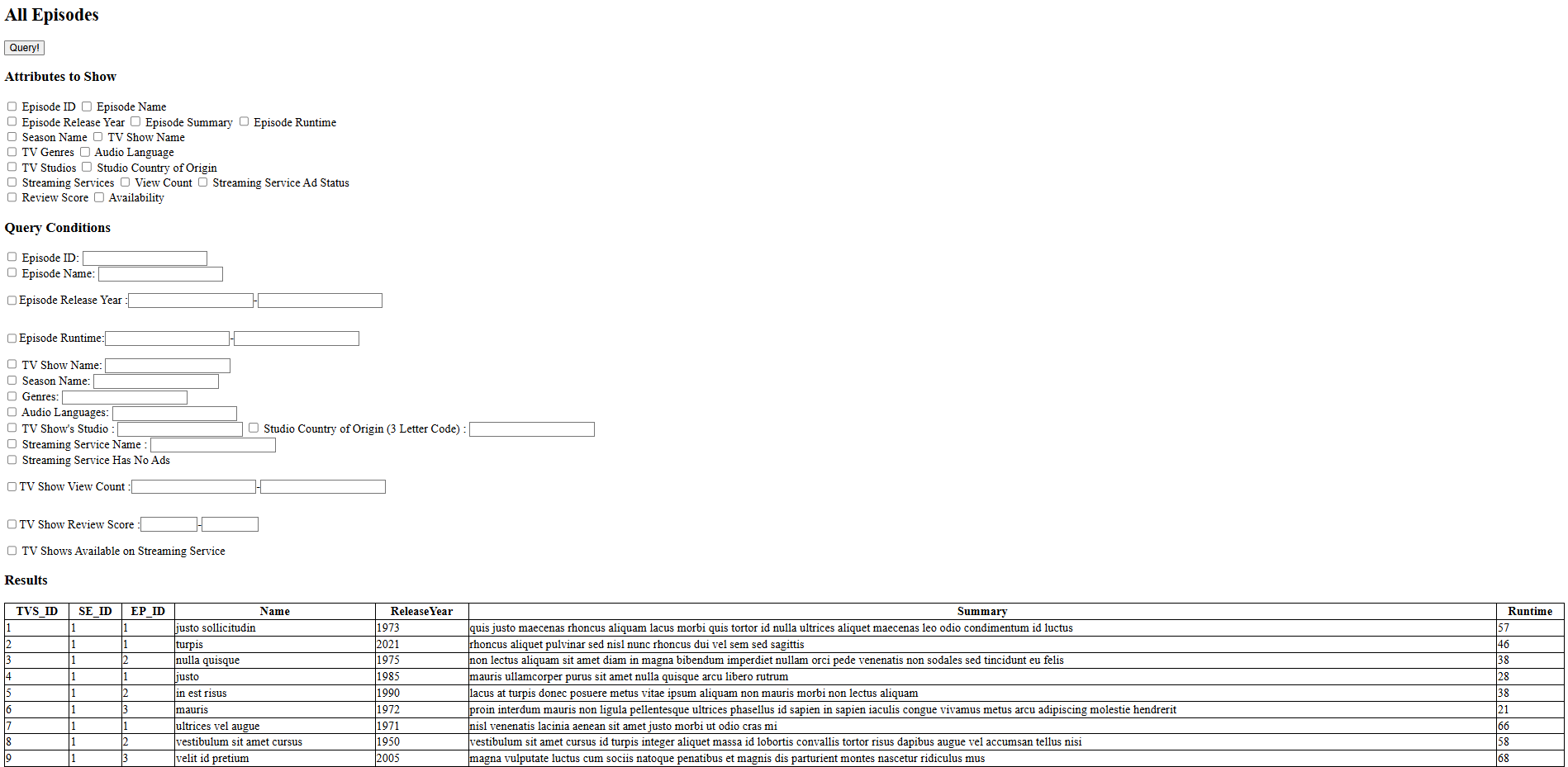
*Figure 18: Find Movies Section*



*Figure 19: Find TV Shows Section*



*Figure 20: All Episodes Section*

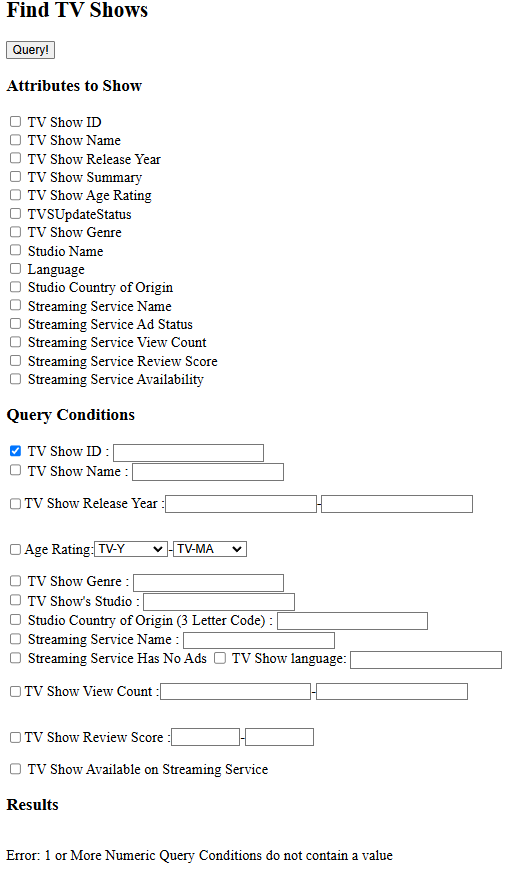
**

*Figure 21: SQL Queries and Use Cases for Movies, Episodes, and TV Shows*

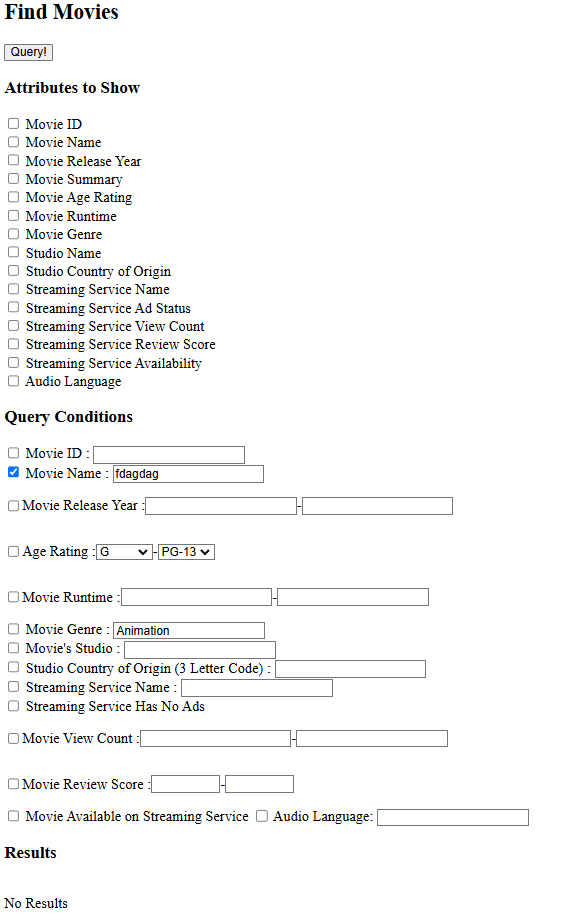
| **SQL Queries** | **Use Case** |
| --- | --- |
| SELECT M.MVName, SS.SSName FROM MOVIES M  INNER JOIN STREAMINGSERVICEMOVIES SSM ON M.MV\_ID = SSM.MV\_ID  INNER JOIN STREAMINGSERVICES SS ON SS.SS\_ID = SSM.SS\_ID  WHERE MVName=’Iron Man’ AND SSM.MVAvailability = true; | A customer may want to find what streaming services are streaming a specific movie such as Iron Man only if it is currently available to be streamed. |
| SELECT M.MVName, SS.SSName, SSM.MVReviewScore FROM MOVIES M INNER JOIN STREAMINGSERVICE MOVIES SSM ON M.MV\_ID = SSM.MV\_ID INNERJOIN STREAMINGSERVICES SS ON SS.SS\_ID = SSM.SS\_ID  WHERE SSM.MVAvailability = true AND SSM.MVReviewScore >= 8 AND SS.SSName = ‘Netflix’; | A customer may want to find all the movies available in a streaming service such as on Netflix that has a high review score like 8. |
| SELECT M.MVName, G.MVGenre, SS.SSName FROM MOVIES M  INNER JOIN STREAMINGSERVICE MOVIES SSM ON M.MV\_ID = SSM.MV\_ID  INNER JOIN STREAMINGSERVICES SS ON SS.SS\_ID = SSM.SS\_ID  INNSER JOIN MOVIEGENRES G ON G.MV\_ID = M.MV\_ID  WHERE SSM.MVAvailability = true AND M.Genre = ‘Action’ AND SS.SSName = ‘Netflix’; | A customer may want to find all movies available on a streaming service like Netflix that is in the Action genre. |
| SELECT M.MVName, L.MVLanguage, SS.SSName FROM MOVIES M  INNER JOIN STREAMINGSERVICE MOVIES SSM ON M.MV\_ID = SSM.MV\_ID  INNER JOIN STREAMINGSERVICES SS ON SS.SS\_ID = SSM.SS\_ID  INNER JOIN MOVIEAUDIOLANGUAGEOPTIONS L ON L.MV\_ID = M.MV\_ID AND SS.SS\_ID = L.SS\_ID  WHERE SSM.MVAvailability = true AND L.MVLanguage = ‘Spanish’ AND SS.SSName = ‘Netflix’; | A customer may want to find all movies available on a streaming service like Netflix that has the Spanish audio language option. |
| SELECT M.MVName, M.MVAgeRating, SS.SSName FROM MOVIES M  INNER JOIN STREAMINGSERVICE MOVIES SSM ON M.MV\_ID = SSM.MV\_ID  INNER JOIN STREAMINGSERVICES SS ON SS.SS\_ID = SSM.SS\_ID  WHERE SSM.MVAvailability = true AND M.MVAgeRating IN (PG, PG-13) AND SS.SSName = ‘Netflix’; | A customer may want to find all movies available on a streaming service like Netflix that has an age rating from PG to PG-13. |
| SELECT E.Name, TV.TVSName FROM EPISODES E JOIN TVSHOWS TV ON E.TVS\_ID = TV.TVS\_ID WHERE TVSName = “The Last Airbender”; | A customer may want to see all of the episode names from a TV series like “The Last Airbender”. |
| SELECT E.Name, TV.TVSName, SSName FROM EPISODES E INNER JOIN TVSHOWS TV ON E.TVS\_ID = TV.TVS\_ID INNER JOIN STREAMINGSERVICETVSHOWS SSTV ON SSTV.TVS\_ID = TV.TVS\_ID INNER JOIN STREAMINGSERVICES SS ON SSTV.SS\_ID = SS.SS\_ID WHERE TVSName="Demon Slayer: Kimetsu no Yaiba" AND SSName="Amazon Prime"; | A customer may want to see all the episodes of a show like Demon Slayer but on a specific streaming service like Amazon Prime |
| SELECT E.Name, TV.TVSName, Language FROM EPISODES E INNER JOIN TVSHOWS TV ON E.TVS\_ID = TV.TVS\_ID INNER JOIN TVSHOWAUDIOLANGUAGEOPTIONS AL ON TV.TVS\_ID = AL.TVS\_ID | A customer may want to see all the language options for the TV show and episode they want to watch. |
| SELECT E.Name, TV.TVSName, TVSViewCount FROM EPISODES E INNER JOIN TVSHOWS TV ON E.TVS\_ID = TV.TVS\_ID INNER JOIN STREAMINGSERVICETVSHOWS SSTV ON SSTV.TVS\_ID = TV.TVS\_ID WHERE TVSViewCount >= 100000 AND TVSViewCount <= 2500000 | A customer may want to see all of the TV shows and their episodes with their view count between 100K and 2.5M |
| SELECT TVS.TVName, TVG.Genre FROM TVShows AS TVS JOIN TVShowGenres AS TVG ON TVS.TVS\_ID = TVG.TVS\_ID WHERE TVS.TVSName LIKE ‘%The%’; | A customer may want to see the names of TV shows that start with the word ‘The’ along with their corresponding genres |
| SELECT SS\_ID, AVG(ReviewScore) FROM StreamingServiceTVShows GROUP BY SS\_ID HAVING AvgReviewScore > 5; | A customer may want to calculate the average review score for streaming services and list services with scores above 5. |
| SELECT Language, COUNT(\*) AS LanguageCount FROM TVShowAudioLanguageOptions GROUP BY Language ORDER BY LanguageCount DESC; | A customer may want to know the number of TV Shows available in each audio language. |
| SELECT TVG.Genre, TVS.TVSName, SSTVS.TVSReviewScore  FROM TVShowGenres AS TVG  JOIN TVShows AS TVS ON TVG.TVS\_ID = TVS.TVS\_ID  JOIN StreamingServiceTVShows AS SSTVS ON TVS.TVS\_ID = SSTVS.TVS\_ID  JOIN StreamingServices AS SS ON SSTVS.SS\_ID = SS.SS\_ID  WHERE SS.SSName = ?  AND SSTVS.TVSReviewScore = (SELECT MAX(SSTVS2.TVSReviewScore)  FROM StreamingServiceTVShows AS SSTVS2  WHERE SSTVS2.SS\_ID = SS.SS\_ID)  ORDER BY TVG.Genre, SSTVS.TVSReviewScore DESC; | A customer may want to find the best-rated TV shows by genre available on a specific streaming service. It connects data from four tables and demonstrates the use of advanced SQL techniques by finding the highest rated tv show for the selected service |

Figure 18, 19, and 20, showcase the UI display about three sections with use case queries related to movies, TV shows, and episodes. As with all our retrieve operations in the project, we have subsections for selecting what attributes to show and what conditions need to be fulfilled with checkboxes and additional input fields for certain query conditions users can click to include in their query. The user will then click the “Query!” button which sends the checkboxes and condition values into the URL as parameters to the backend Node.js code. The code in particular houses logic that decides what attributes to show in the SELECT, tables to join in FROM, as well as WHERE conditions to add. All three of these sections can utilize 0 or more joins. Movies can have 6 tables, TV shows 6 tables, and Episodes 8 tables joined. This allows for many combinations of queries and ways to get information on these three pieces of media. The results section is where a table with the query results from the database is displayed to the user. Figure 21 showcases some example important use cases and SQL queries considering the conditions and attributes a customer may choose for their query.

*Figure 22: Error Message for Condition Inputs*

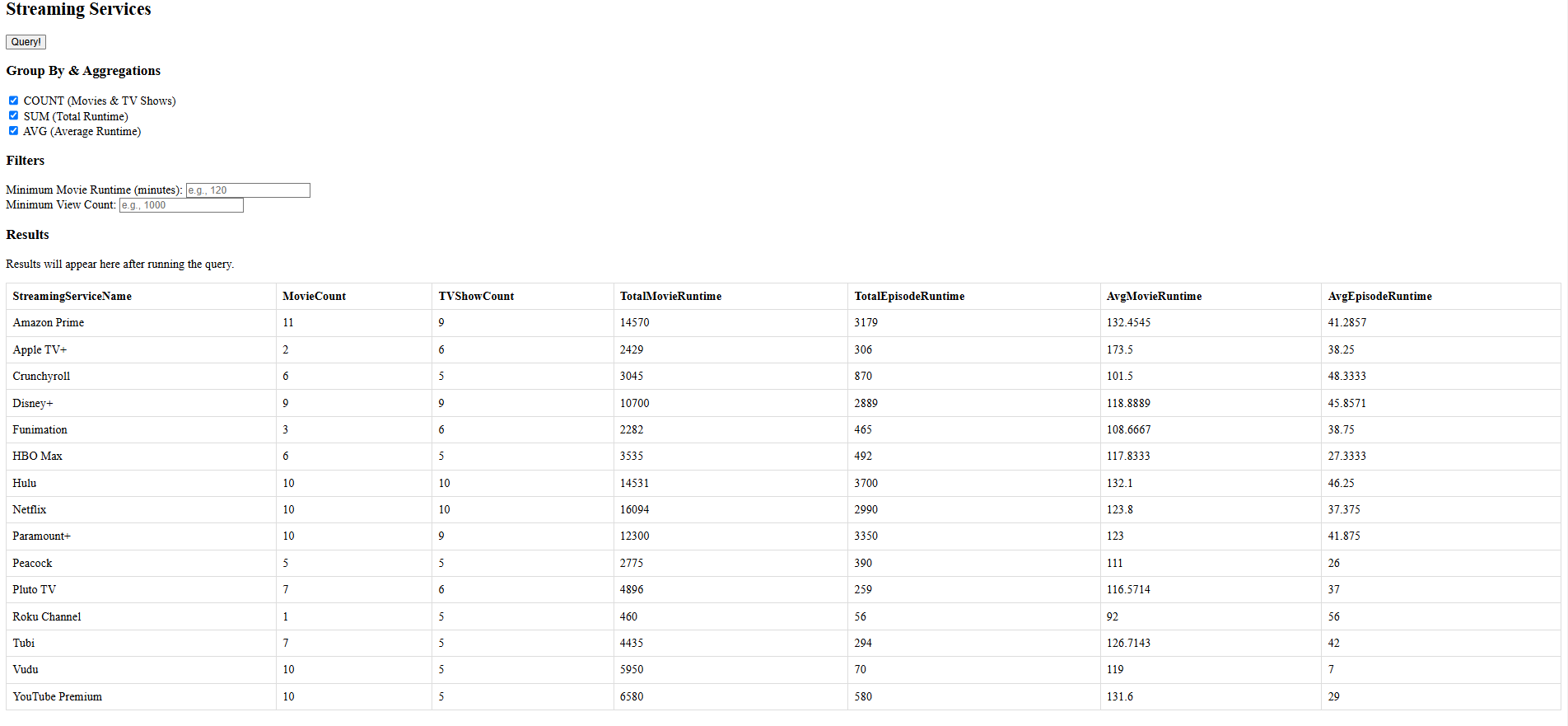
**

*Figure 23: No Results Message*

**

Additionally, we added error and no result messages in the UI. In figure 22, it shows a message whenever numeric condition fields that are checked are not inputted. This is to prevent the UI from giving improper SQL statements to MySQL which would stop the system from working after an error throw. However, we did not implement a message for empty string fields as even an empty string in a SQL query would not yield issues and give a no results display. Figure 23 shows that no results display which was implemented to notify the user that their query was inputted and there were no results.

*Figure 24: Streaming Service Section*

**

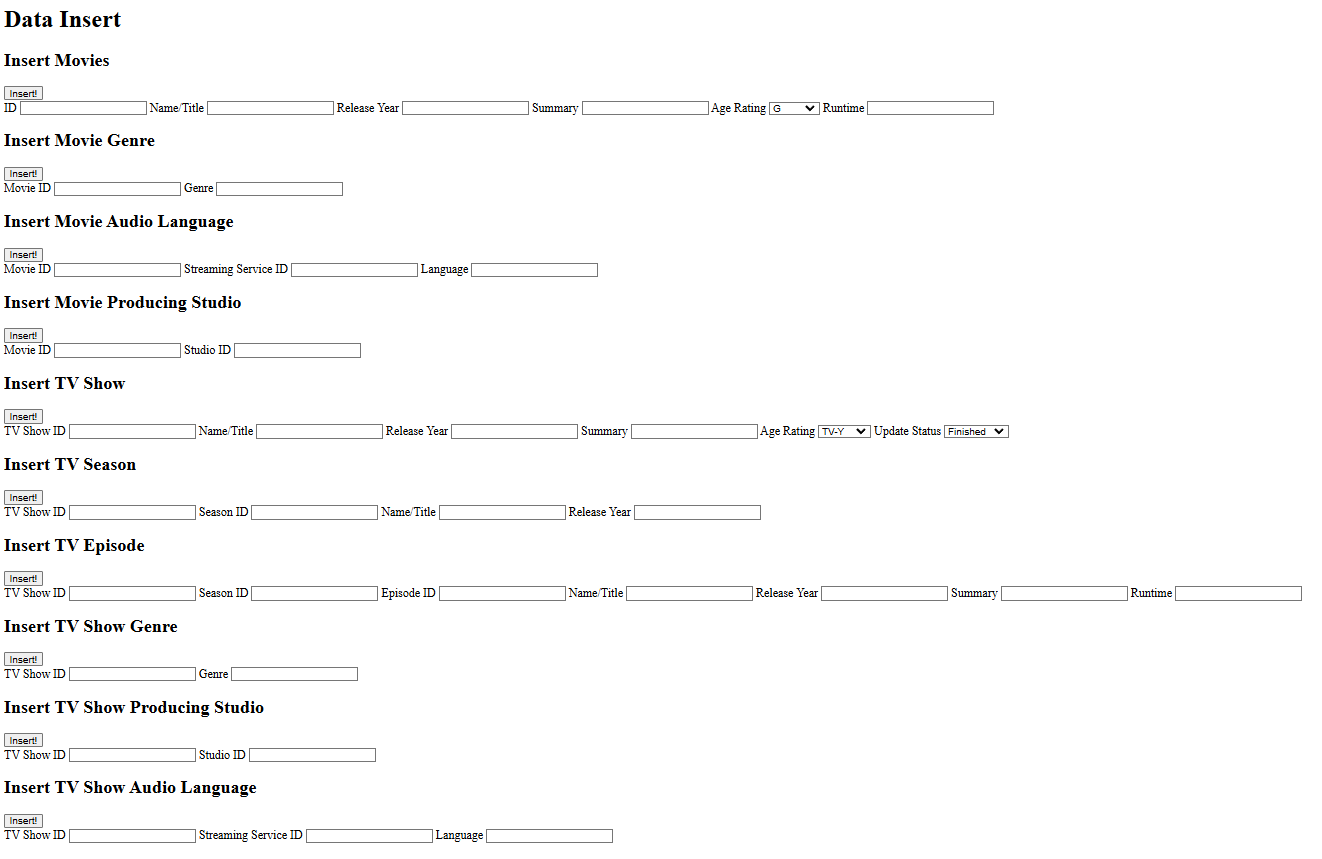
*Figure 25: SQL Queries and Use Cases for Streaming Services*

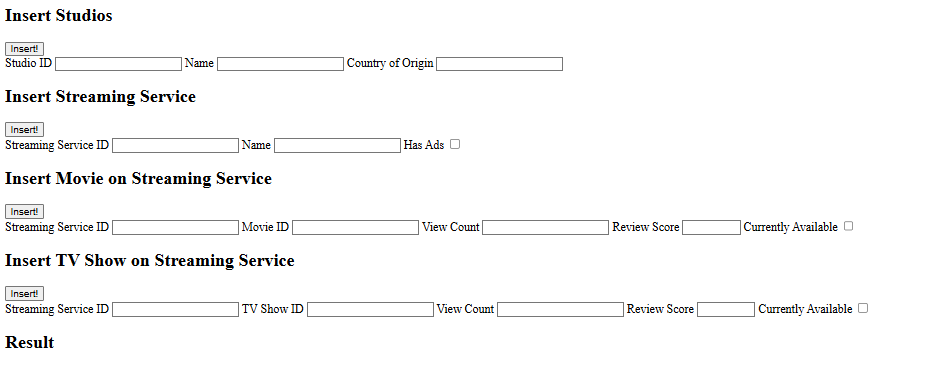
| **SQL Queries** | **Use Case** |
| --- | --- |
| SELECT SS.SSName, COUNT(SST.TVS\_ID) as TVShowCount, COUNT(SSM.MV\_ID) as MovieCount FROM STREAMINGSERVICES SS  LEFT JOIN STREAMINGSERVICEMOVIES SSM ON SS.SS\_ID = SSM.SS\_ID  LEFT JOIN STREAMINGSERVICETVSHOWS SST ON SS.SS\_ID = SSM.SS\_ID  WHERE SSM.MVViewCount >= 2000 AND SSM.TVSViewCount >= 200 AND GROUP BY SS.SS\_ID; | A customer may want to find the total number of movies and TV shows on a streaming service that have a minimum view count of 20000 views. |
| SELECT SS.SSName, SUM(EP.EPRuntime) as TotalEpisodeRuntime, SUM(M.MVRuntime) as TotalMovieRuntime FROM STREAMINGSERVICES SS  LEFT JOIN STREAMINGSERVICEMOVIES SSM ON SS.SS\_ID = SSM.SS\_ID  LEFT JOIN MOVIES M.MV\_ID = SSM.MV\_ID  LEFT JOIN STREAMINGSERVICETVSHOWS SST ON SS.SS\_ID = SSM.SS\_ID  LEFT JOIN EPISODES EP ON SST.TVS\_ID = EP.TVS\_ID  GROUP BY SS.SS\_ID; | A customer may want to find the total runtime of episodes and movies on a streaming service. |
| SELECT SS.SSName, AVG(EP.EPRuntime) as AvgEpisodeRuntime, AVG(M.MVRuntime) as AvgMovieRuntime FROM STREAMINGSERVICES SS  LEFT JOIN STREAMINGSERVICEMOVIES SSM ON SS.SS\_ID = SSM.SS\_ID  LEFT JOIN MOVIES M.MV\_ID = SSM.MV\_ID  LEFT JOIN STREAMINGSERVICETVSHOWS SST ON SS.SS\_ID = SSM.SS\_ID  LEFT JOIN EPISODES EP ON SST.TVS\_ID = EP.TVS\_ID  GROUP BY SS.SS\_ID; | A customer may want to find the average runtime of episodes and movies on each streaming service. |

The last retrieve section to cover is the streaming services section shown in figure 24. The complex query implemented in this section is the use of group by and aggregation functions. The use case being that information on streaming services should be about total sums, counts, and averages of TV show and movie information on streaming services. There are also two additional filters that change the calculations for counts, sums, and averages based on movie runtime and minimum view count. Figure 25 shows the SQL queries and their use cases which display the large number of joins required for this subsection.

## 10.2 Creation Operations

*Figure 26: Data Insert Section*



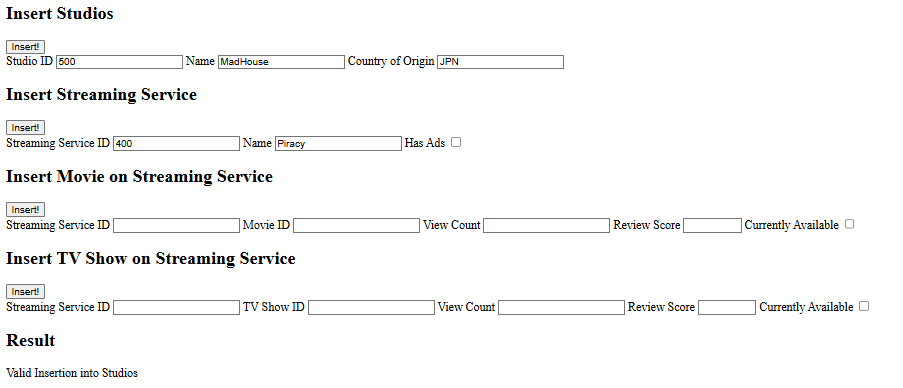


*Figure 27: SQL Queries for Insert Operations*

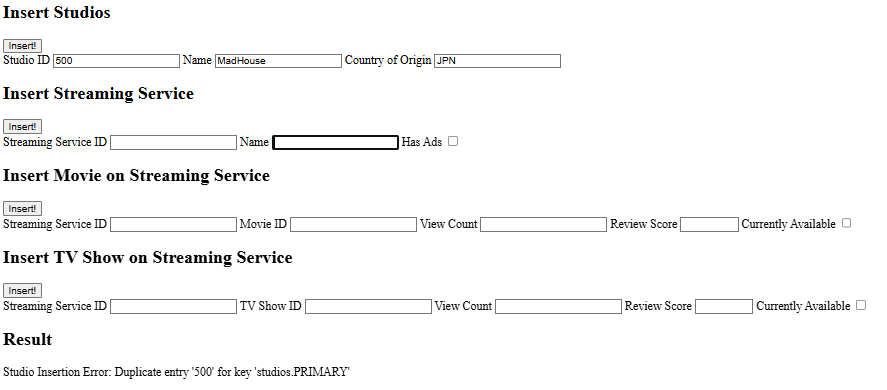
| **SQL Queries** | **Use Case** |
| --- | --- |
| INSERT INTO MOVIES VALUES (1, ‘Avatar’, 2010, ‘Armies and blue people’, ‘PG-13’, 125); | An admin may want to insert a movie into the database. |
| INSERT INTO MOVIEGENRES VALUES (1, ‘Action’); | An admin may want to assign a movie with a genre such as action. |
| INSERT INTO MOVIEAUDIOLANGUAGES VALUES (1, 1, ‘Spanish’); | An admin may want to assign a movie with a language such as Spanish that is streaming on a streaming service. |
| INSERT INTO MOVIEPRODUCEDBY VALUES (1, 1); | An admin may want to assign a movie with its production studio. |
| INSERT INTO TVSHOWS VALUES (1, ‘Gravity Falls’, 2016, ‘Twins solve encounter folk tales and hoaxes’, ‘TV-14’, ‘Finished’); | An admin may want to add a TV show into the database. |
| INSERT INTO SEASONS VALUES (1, 1, ‘Season 1’, 2016); | An admin may want to add a season to an existing TV show in the database. |
| INSERT INTO EPISODES VALUES (1, 1, 1, ‘The Mystery Shack’, 2016, ‘Twins visit their weird uncle over the summer’, 25); | An admin may want to add an episodes to an existing TV show and season in the database. |
| INSERT INTO TVSHOWGENRES VALUES (1, ‘Animation’); | An admin may want to assign a TV show with a genre. |
| INSERT INTO TVSHOWPRODUCEDBY VALUES (1, 1); | An admin may want to assign a movie with its production studio. |
| INSERT INTO TVSHOWAUDIOLANGUAGEOPTIONS VALUES (1, 1, ‘Japanese’); | An admin may want to add a language option for a TV show that is on a streaming service. |
| INSERT INTO STUDIOS VALUES (1, ‘Pixar’, ‘USA’); | An admin may want to add a production studio into the database. |
| INSERT INTO STREAMINGSERVICEMOVIES VALUES (1, 1, 2000, 9, true); | An admin may want to associate a movie to a streaming service with its view count, review score, and availability. |
| INSERT INTO STREAMINGSERVICETVSHOWS VALUES (1, 1, 3000, 5.1, false); | An admin may want to associate a TV show to a streaming service with its view count, review score, and availability. |

Figure 26 shows all the creation operations which utilize SQL insert statements. The use cases are meant for the database admins who can add data about TV shows, movies, and streaming services. There are individual insert subsections for all tables due to the admin having use cases for inserting data into all the tables. Figure 27 shows example SQL queries and use case for insertions into all tables in the project.

*Figure 26: Valid Insertion Message*

**

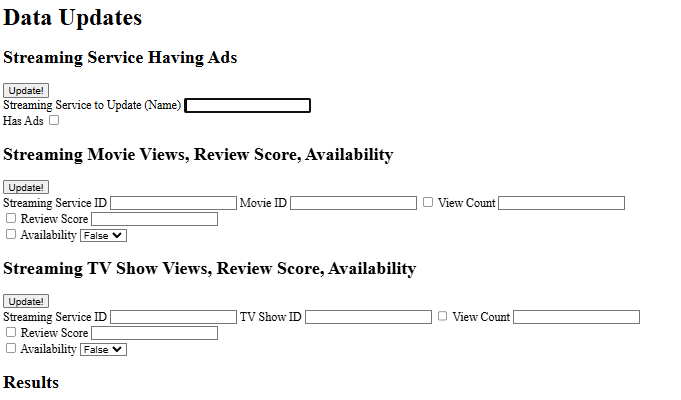
*Figure 27: Invalid Insertion Message*

**

Furthermore, we added messages to confirm the validity of whether inserts were successful. Figure 26 shows a valid insertion in the results subsection while figure 27 shows an invalid insertion and the reason why. The implementation for the error message utilizes the JSON object return from the database which contains a SQL message key. The backend accesses the value of this message and displays it to the admin. We did not implement non-technical error messages but instead just showed the MySQL error messages due to time constraints which would have required extra implementation of substring functions and looking up all error codes that could appear.

## 10.4 Update Operations

*Figure 28: Data Updates Section*

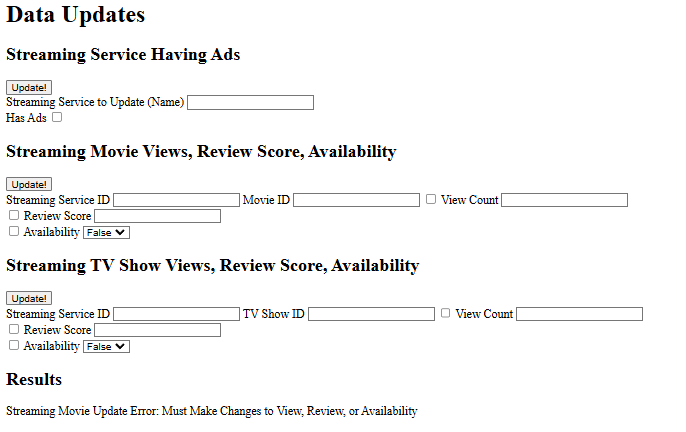
**

*Figure 29: SQL Queries and Use Cases for Update Operations*

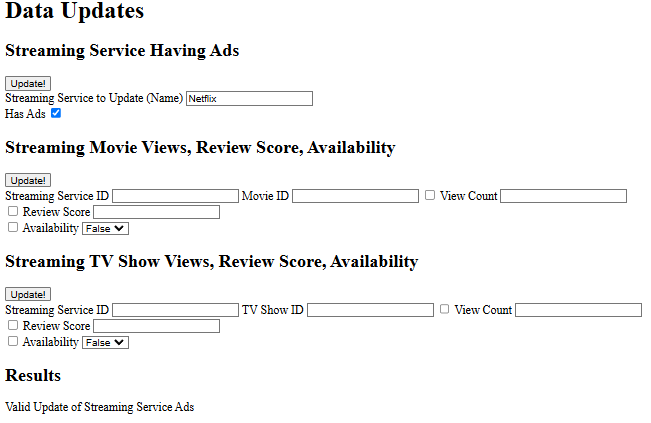
|  |  |
| --- | --- |
| **SQL Query** | **Use Case** |
| UPDATE STREAMINGSERVICEMOVIE SET MVReviewScore = 9.9, MVViewCount = 20000, MVAvailabilityTrue WHERE MV\_ID = ‘100’ AND SS\_ID=’300’; | An admin may want to update a specific movie’s review score, view count, or availability on a streaming service. |
| UPDATE STREAMINGSERVICETVSHOW SET TVSReviewScore = 9.9, TVSViewCount = 20000, TVSAvailabilityTrue WHERE TVS\_ID = ‘100’ AND SS\_ID=’300’; | An admin may want to update a specific TV show’s review score, view count, or availability on a streaming service. |
| UPDATE STREAMINGSERVICES SET HasAds=true, SSName=’Max’ WHERE SS\_ID=’300’; | An admin may want to update a streaming service’s ad status and name due to a change in the terms of service, base subscription tier, or rebranding. |

As can be seen in figure 28, the number of subsections of the update operations is significantly less than the insertion subsections. This is due to the use cases of update where it is assumed that the initial insertions of movies and TV shows are stable and inherent which would not need to be changed. On the other hand, streaming services related tables are the most likely to have data changes made to maintain data accuracy and have use cases shown in figure 29. However, not all the attributes of these tables would need to be changed. For example, a streaming service’s ID would not need to be changed while other aspects like having ads would need to be updated due to terms and conditions changing. Additionally, as will be later discussed, the streaming movies and TV shows on streaming services would not need to be deleted if they are removed from the streaming service because there is a possibility they will return after a renewed streaming license. This is why the availability can be changed for movies and TV shows being streamed.

*Figure 30: Update Error Message*

**

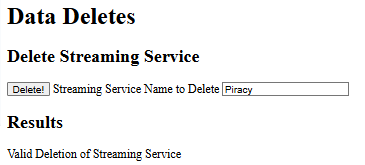
*Figure 31: Valid Update Message*

**

As shown in figure 30 and 31, we implemented messages to indicate the validity of updates. Figure 30 shows the attributes to change need at least one thing to change which prevents MySQL syntax errors.

## 10.5 Delete Operation

*Figure 32: Data Delete Section*

**

*Figure 33: SQL Query and Use Case for Delete Operation*

|  |  |
| --- | --- |
| **SQL Query** | **Use Case** |
| DELETE FROM STREAMINGSERVICES WHERE SSName = ‘PlutoTV’; | An admin may want to delete a streaming service and its associated information in connected tables due to the streaming service’s end of service. |

In the current implementation there is only one delete operation present which is deleting a streaming service as shown in figure 32 with its associated use case in figure 33. This is due to the assumption that movies and TV shows may be delisted from streaming services but always have the possibility to be brought back and streamed which would make deleting require more insertions in the event that happens which should be avoided. Instead, streaming services are the only data that can be deleted because they can go out of service. In our design, we specify a name of the streaming service and when it is deleted, the MySQL database has foreign key constraints of data using the streaming service ID cascade and be deleted. Once again, we also implemented a message indicating the validity of the deletion.

# 11 Postmortem Review

## 11.1 Effort and Contributions

The group spent a significant amount of effort on the project. While we didn’t keep exact time logs of our contributions, we estimate that each member worked on the project for 1-2 hours individually for each of the three iterations and always met together for those weeks 2 hours each week. For the creation of the UI and final data population, we estimate that each individual spent at least 4 hours for their own portion of the work and met together twice for a total of 4 hours of meetings.

In addition, the group delegated work to efficiently finish the project.In iteration one, we created the entity relationship diagram individually in order to merge them together for a more comprehensive ER diagram which we translated into the relational model. Our RM was then created based on the merged ER diagram. For iteration 2, team members each looked at possible tools to use individually and came together and shared their recommendations and findings which we then discussed and chose our preferred tools. For iteration three, the team split the work down in two with one team being Jasper and Kenny creating the SQL table creation code and Caitlin and Duong creating insert data. For iteration 4, we had to learn how to use Node.js and Azure which none of us had experience in in order to create the UI. To streamline the work process, Jasper figured out the Azure and Node.js portion and created a javascript and HTML template for the rest of the members to follow. This reduced the time required for group members to learn how to implement the project. We separated the work in terms of queries by table which were Jasper doing movies, Kenny doing TV shows, Caitlin doing episodes, and Duong doing streaming services. However, the insert, delete, and update operations also needed to be included but were more difficult to implement so Jasper did that portion of the work. During that same week, to balance out the workload, Duong, Caitlin, and Kenny were assigned to use Mockaroo to generate additional data to insert into the database.

The end product showcases our prowess of not only planning the workload for the entire design and creation process but also the delegation required for a fair and efficient implementation of the project.

## 11.2 Challenges and Feedback

During the ER to RM translation process, we realized that in our old ER design, we had so many multi-value attributes and many-to-many relationships that we had over 20 tables in the relational model which would have included tables like star actors, caption languages, and directors. Since the project only required 7 tables and the time needed to complete 18 tables and the related UI functions and data population would have been immense, we cut down on these multi-value attributes and relationships to reduce the relational model to our current version with 14 tables which is still a significant number but which facilitated our more important use cases such as genre, language options, and where TV shows and movies were streaming.

While we initially used information from IMDB and streaming service websites for generating our data, after receiving feedback that we did not have enough data from the graders and professor we used Mockaroo to further populate our database. Mockaroo allowed us to create test data to be generated by specifying the attributes and the type of parameter for our schema in a quick and easy way which could easily be translated into INSERT statements.

Additionally, during the final week before the due date, there was a complication that we didn’t realize we needed CRUD operations of INSERT, UPDATE, and DELETE. However, as written in the section *9.6 Effort and Contributions* Jasper ended up implementing them due to the workload differences and knowledge of Node.js.

If we had more time, we would have liked to change the RM to allow for episodes to have view counts and review scores. In our final version, the table STREAMINGSERVICETVSHOWS had view counts like the equivalent movie table but that doesn’t make as much sense as a TV show itself is not watchable, but its episodes are. We should have added an additional table of STREAMINGSERVICEEPISODES. However, we were too far into the UI coding process to catch this, and it would have taken significant time to fix. We would have also liked to include a separate page for admin CRUD operations because as of now they are all on the same page without any authentication systems.

Finally, the biggest challenges were learning Azure and Node.js. Azure in particular has many different configurations and options which are not as easy to understand as a student without any cloud experience. Setting the firewall, admin logins, SSL certificates, and deploying the code were all difficult to learn. The firewall especially was an issue because we delegated functionality to each group, we had to configure the firewall to allow everyone to access the database from different IP addresses to be able to test our individual code. Node.js required many dependencies and installations on our own computers. It also had many different options for how we could implement our project such as database connections through pool or connection modules and dynamic app functionality with express or server modules. There was also difficulty learning URL parameters which we had to implement due to the many combinations of attributes and conditions of our queries users can have. For string inputs in particular, some symbols such as ‘+’ common in streaming service names like ‘Disney+’ could not be entered as URL parameters normally but had to be encoded to be accurately sent to the backend code.

# 12. Previous Iteration Updates:

| **Document** | **Change** | **Description** | **Date** |
| --- | --- | --- | --- |
| Iteration 2 | Added Section for MySQL Workbench | Added to better reflect the tools we are using for the project and why we are using MySQL Workbench. | 11/27/24 |
| Iteration 3 | Added a Previous Iteration Updates section | Added due to feedback from grader’s that we lacked this section for the original Iteration 3 assignment. | 12/9/24 |
| Iteration 3 | Moved the “Sources” section to the bottom of the document. | Moved due to the need to accommodate for the “Data Sourcing” section and to fix formatting issues based on grader feedback. | 12/9/24 |
| Iteration 3 | Added “Data Sourcing” section | Added to better explain how our process of getting data changed from the original iteration three to the now iteration 4. | 12/9/24 |
| Iteration 3 | Modified “Data Processing” section | Added more explanations on how we translated datasets from online sources into INSERT statements through manual translation and Excel automation. | 12/9/24 |

# 13. Sources

“Apple TV+.” *Apple*, 2024, tv.apple.com. Accessed 12 Dec. 2024. Accessed 27, Nov. 2024.

“Crunchyroll - Watch Popular Anime & Read Manga Online.” *SONY PICTURES*, crunchyroll.com.

“Free Movies & TV Fewer Ads than Cable No Subscription Required” *Tubi,* tubitv.com. Accessed 27, Nov. 2024.

“Max | the One to Watch.” *WarnerMedia Direct*, max.com. Accessed 27, Nov. 2024.

“Mockaroo – Random Data Generator and API Mocking Tool” *Mockaroo*, mockaroo.com. Accessed 12, Dec. 24.

“Paramount Plus: Stream Movies, Shows & Live TV.” *Paramount*, 2019, paramountplus.com. Accessed 27, Nov. 2024.

“Peacock: Stream TV and Movies Online, Watch Live News and Sports.” *Peacock TV*, peacocktv.com. Accessed 27, Nov. 2024.

“Prime Video.” *Amazon*, primevideo.com. Accessed 27, Nov. 2024.

“Ratings, Reviews, and Where to Watch the Best Movies & TV Shows.” *IMDb*, IMDb.com, www.imdb.com/. Accessed 27 Nov. 2024.

“Stream Disney, Marvel, Pixar, Star Wars, National Geographic | Disney+.” *Disney*, disneyplus.com. Accessed 27, Nov. 2024.

“Stream TV and Movies Live and Online.” *Hulu*, hulu.com. Accessed 27, Nov. 2024.

“The Roku Channel” *Roku*, therokuchannel.roku.com. Accessed 27, Nov. 2024.

“Unlimited Movies, TV Shows, and More.” *Netflix,* netflix.com. Accessed 27, Nov. 2024.

“Vudu - Watch Movies.” *Fandango*, vudu.com. Accessed 27, Nov. 2024.

“Welcome To A Whole New World Of TV.” *Pluto,* pluto.tv/about-us. Accessed 27, Nov. 2024.

“YouTube Premium.” *YouTube*, [www.youtube.com/premium](http://www.youtube.com/premium). Accessed 27, Nov. 2024.