Dream to Stream
Stage 6 Project Report

Developers and Authors:

Anthony Doan - UIUC MCS Student

Divya Manirajan - UIUC MCS Student

Mohammad Zayyad - UIUC City Scholar Student

Rohan Sreenivasan - UIUC City Scholar Student

Overview:

Dream to Stream is an application that combines the Shows and Movie tables from Netflix, Hulu, Disney+, and Prime Video. With the provided data, users now have a centralized area to see what's available to them. Using our application dreamToStream, users can create, retrieve, update, and delete their Show and Movie ratings, see what Movie/Show are available to them in their country, and do a mesh with a fellow user to see comparisons between Movies and Shows both have rated.

Technologies Used:

The application is all hosted within Google Cloud Platform (GCP) with the following architecture:

- Backend database:
  - Datasets from Netflix, Hulu, Disney+, and Prime Video loaded into GCP's MySQL instance.
- Frontend:
  - Using Node.JS Framework and Javascript to render out the server and pages
  - Virtual Machine and Network configuration hosted in GCP's Virtual Machine Compute Engine.
- Codebase and Source Control:
  - Hosted in Github: https://github.com/cs411-alawini/sp23-cs411-cmp-team001-dreamToStream

Reflection:

- List of changes in direction of Final Project from the Original Proposal in Stage 1:
    - Using separate platforms (Netflix, Hulu, Prime, Disney+) initially as individual tables → Combine them all together for Movies and Shows tables with similar columns - initial columns: show_id, title, date_added, release_year, genre, length, type, rating, rotten_tomatoes, duration.
    - Being able to view other users reviews → Being able to view your own reviews based on your user id. People could technically use this, but they would need to know other people's ID.
    - Wanted to add a field for a user add a write up for each movie or show watched. For example, a user could write a review on a show they watched for all other users/friends to see.

- Discussion of what our application achieved or failed to achieve regarding its usefulness:

    The application was able to achieve basic functions such as creating a user, signing in an existing user, creating and viewing all show/movie ratings for the viewer signed in, updating a rating for either a movie or show, deleting a rating for either a movie or show, implementing 2 advanced queries, and integrating a stored procedure and trigger within the SQL backend.

    In terms of usefulness, the application had all of the necessary functions to provide a well-rounded user experience with creating an account, logging in, seeing movies or shows available in their country, adding/viewing/updating their movie/show ratings and also meshing their comparison with another user. The main function to add is a "view ratings" section which can take a specific movie or show and return all the ratings from all different viewers that match this title.

    However, there were a few ideas that could improve the application. In terms of the mesh capability, for the output, the application could have shown the users' names instead of their userIDs and a capability to show recommendations of genres for both users to look into as well as show/movies to watch would add a creative capability to the application. Also, utilization of the location data provided to show heat maps of what is available in their countries could provide solid values to our users. In terms of database design, to optimize and reduce the lines of code used for ratings, the MovieRatings and ShowRatings tables could have been combined with an extra column to identify the type of media it was. This

would reduce the need for checks of “if MovieRating do this else if ShowRating do this…”.

- Discussion on changes to the schema or source of the data for the application:

    There were a few changes made to the scheme from stage 1. Most of these changes were small in terms of adding or removing an attribute from a table. Some of the major changes include adding CastGroup and CustomerGroups tables to better improve some of the functionalities in the overall product. We also split our main “media” table into two separate tables for MovieRatings and ShowRatings.

    The source of the data stayed the same throughout the project. All the data was taken from kaggle datasets about a variety of streaming platforms and information about the shows and movies featured on each. These were all combined together to provide a centralized table for Movies and Shows where it allowed the team better control of the columns to query.

- Discussion on changes to the ER diagram and/or table implementations - noting differences between the original design and the final design and reasoning for how this is a more suitable design.

    While working on stage 3 there were a few aspects of the table designs and ER diagrams that had to be changed to better suit the implementation in the SQL database. Some of the differences included changing the Platform table to add a country field and changing the CustomerGroups table to add a groupName value. These changes were made to better optimize the database and they were useful in implementing queries for different functions. The updates that were made to the tables and connections between tables was necessary to accurately implement the tables in a way that would make the product useful.

- Discussion of functionalities added or removed:

    A functionality that was added is a sign in feature which takes in a user’s first name and last name, and signs them into their respective account. Once a user is signed in, then they are allowed to view their ratings, add/update/delete their ratings, find specific information based on two advanced queries, and find information using a stored procedure. This improved the overall product since only existing users should be able to view the information provided and make changes to the database that this product is hosted on. Another functionality that was added is the ability for

a user to view their show and movie ratings. This allows users to always check what movies or shows they have reviewed, and also what rating they gave to each title.

The functionality that was removed from stage 1 was the ability to allow users to check all ratings given a specific movie or show title. This was removed due to not fitting the requirements of this project and therefore was deemed an extra feature to implement if time allows.

- Advanced Database Programs and how it complements the application:

  The advanced queries implemented (Stored Procedure and Trigger) and their functionality to the application are explained below:

  1. Stored Procedure: The stored procedure that was implemented into our application was called get_ratings which took in two integers which represented the userIDs. This would then go in and get the movie and show ratings for both users and run a comparison between all of them. For the comparison, the stored procedure would compare the rating of user 1 to the rating of user 2 and output their results noting if one user had a higher rating then the other or if they rated the movie/show the same. Then, it would output both users' ratings. This would be used in the front end to complete the Mesh capability of the application after the consumer provided two user IDs to compare.

     Advanced queries used: Join, Group By, and Subquery

     SQL Query:

```
CREATE PROCEDURE get_ratings(IN userId1 INT, IN userId2 INT)
BEGIN
  DECLARE movie_name VARCHAR(255);
  DECLARE movie_rating1 INT;
  DECLARE movie_rating2 INT;
  DECLARE show_name VARCHAR(255);
  DECLARE show_rating1 INT;
  DECLARE show_rating2 INT;
  DECLARE exit_loop BOOLEAN DEFAULT FALSE;

  -- USING JOIN (ADVANCED QUERY 1) to get Movies that both users have reviewed from MovieRating
  DECLARE movie_cursor CURSOR FOR
```

```
    SELECT MR.name, MR.value as rating1, MR2.value as rating2
    FROM MovieRating MR
    INNER JOIN MovieRating MR2 ON MR.name = MR2.name
    WHERE MR.id = userId1 AND MR2.id = userId2;


  -- USING SUBQUERY (ADVANCED QUERY 2) to get Shows that both
users have reviewed from ShowRating
  DECLARE show_cursor CURSOR FOR
    SELECT SR.name,
      (
        SELECT value FROM ShowRating
        WHERE id = userId1 AND name = SR.name
      ) as rating1,
      (
        SELECT value FROM ShowRating
        WHERE id = userId2 AND name = SR.name
      ) as rating2
    FROM ShowRating SR
    WHERE SR.id IN (userId1, userId2)
    GROUP BY SR.name -- Group by ShowRating.Name (ADVANCED
QUERY)
    HAVING COUNT(DISTINCT id) = 2;


  DECLARE CONTINUE HANDLER FOR NOT FOUND SET exit_loop =
TRUE;


  SET movie_rating1 = 0;
  SET movie_rating2 = 0;
  SET show_rating1 = 0;
  SET show_rating2 = 0;


  -- LOOP THROUGH MOVIE RATINGS --
  OPEN movie_cursor; -- USING CURSOR
  movie_loop: LOOP -- USING LOOPING STRUCTURE
    FETCH movie_cursor INTO movie_name, movie_rating1,
movie_rating2;
```

```
    IF exit_loop THEN
      LEAVE movie_loop;
    END IF;

    -- Processing Movie Ratings USING CONTROL STRUCTURE
(IF-Statement)
    IF movie_rating1 > movie_rating2 THEN
            SELECT CONCAT('User ', userId1, ' rated the Movie "',
movie_name, '" higher than User ', userId2, ' (', movie_rating1, ' vs ',
movie_rating2, ')') AS comparison;
            ELSEIF movie_rating1 < movie_rating2 THEN
            SELECT CONCAT('User ', userId2, ' rated the Movie "',
movie_name, '" higher than User ', userId1, ' (', movie_rating2, ' vs ',
movie_rating1, ')') AS comparison;
            ELSE
            SELECT CONCAT('Both users rated "', movie_name, '" equally
(', movie_rating1, ')') AS comparison;
            END IF;

  END LOOP;

  CLOSE movie_cursor;

  -- LOOP THROUGH SHOW RATINGS --
  SET exit_loop = FALSE;
  OPEN show_cursor; -- USING CURSOR
  show_loop: LOOP -- USING LOOPING STRUCTURE
    FETCH show_cursor INTO show_name, show_rating1, show_rating2;

    IF exit_loop THEN
      LEAVE show_loop;
    END IF;

    -- Processing Show Ratings USING CONTROL STRUCTURE
(IF-Statement)

    IF show_rating1 > show_rating2 THEN
```

```
            SELECT CONCAT('User ', userId1, ' rated the Show "', show_name, '" higher than User ', userId2, ' (', show_rating1, ' vs ', show_rating2, ')') AS comparison;
            ELSEIF show_rating1 < show_rating2 THEN
             SELECT CONCAT('User ', userId2, ' rated the Show "', show_name, '" higher than User ', userId1, ' (', show_rating2, ' vs ', show_rating1, ')') AS comparison;
            ELSE
             SELECT CONCAT('Both users rated "', show_name, '" equally (', show_rating1, ')') AS comparison;
            END IF;

  END LOOP;

  CLOSE show_cursor;

 END;
```

2. Trigger: There were two triggers created (boundMovieRating and boundShowRating that was used on the database that checks before an update to a rating using a condition to set bounds on how high or low a user can update a rating. If a user updated a value greater than 10, limit it to 10. If a user updated a value to below 0, then set it to 0. The value this brings to the application is that it would limit the users ability to give ratings that are greater than 10 or below 0 to keep the standard the same across the application for processing.

   SQL Queries:

   - boundMovieRating

```
CREATE TRIGGER boundMovieRating BEFORE UPDATE ON MovieRating FOR EACH ROW
BEGIN
  IF NEW.value > 10 THEN
    SET NEW.value = 10;
  ELSEIF NEW.value < 0 THEN
    SET NEW.value = 0;
  END IF;
END;
```

- boundShowRating Trigger:

```
CREATE TRIGGER boundShowRating BEFORE UPDATE ON
ShowRating FOR EACH ROW BEGIN
  IF NEW.value > 10 THEN
    SET NEW.value = 10;
  ELSEIF NEW.value < 0 THEN
    SET NEW.value = 0;
  END IF;
END;
```

- Team Members Technical Challenges:

Anthony Doan:

There was the challenge of being able to set up the initial SQL and VM services within GCP as this was a new Cloud platform to all the members. The lecture videos provided some guidance, however, there were still aspects (front end development knowledge and interacting with the SQL database) that were newer to the team.

Divya Manirajan:

There was a challenge with implementing proper advanced SQL queries in both the backend and front end. The backend was difficult because it relied on the tables and structure of the database created in stage 1, which we had to change to improve the overall application efficiency. Since changes were made to the database, queries that we came up with in earlier stages had to be modified to work with the updated database. Integrating these queries with the front-end was also a challenge since the format and wording of Node.js required specific syntax to integrate properly.

Mohammad Zayyad:

There was the challenge of different SQL versions within the version provided in GCP. Additionally, there were challenges with connecting to the database and altering code and maintaining version control with no conflict between team members.

Rohan Sreenivasan:

Had challenges adding data into GCP from our excel file. I ended up using a jupyter notebook to connect to the DB using python and running insert queries on each table. The process was taking a long time and we eventually found out that we can run these queries in bulk to save time. Additionally, when implementing the trigger I kept getting errors with each line and I later found out that GCP was interpreting each semicolon as the end of the sentence. We collaborated and realized that the use of a delimiter can signify the end of a code block in our terminal. This resolved the issue.

- Other changes from Final Project to Original Proposal

  In the original proposal, we had expected to add a groupUsers aspect for users to find shows/movies that fit multiple user's preferences. Although we did build out the GroupCustomers table for this aspect, we were not able to connect this to the Users table or implement this table with our project. However, the stored procedure we implemented provided a similar functionality by comparing 2 users and outputting shows/movies that both users have placed reviews for.

- Future Application Improvements:

  A future application improvement would be to update the stored procedure since the current one did not output the genre and recommended Movies and Shows for users to see. With the integration, it would help complete the creative aspect of the application by providing a recommendation engine that allows for two users who entered the mesh to understand what their ratings were, but also seeing similar genres, shows or movies that both users can explore together.

  Another improvement that could be made was to better utilize the advanced queries that would just display all the countries and user reviews of movies and shows in that country. This would be done in the front end where, as mentioned earlier, the application could display a heat map of popular showings in different countries as this would provide users better functionality of the advanced queries. The update would be done in the SQL query to provide average ratings of all reviews of a given within a given country, which could display top movies/shows in a country and the platform whenever users hover over that country on the map.

- Division of Labor:

  Anthony Doan: Google Cloud Platform integration, backend migrations, front end development

  Divya Manirajan: SQL query development and integration of backend database, front end development

  Mohammad Zayyad: SQL query development and integration of backend database, front end development

  Rohan Sreenivasan: Backend migrations of data tables into MySQL, SQL query development, backend migrations, front end development

  Note: All users contributed to all aspects of the delivery of the application through the consistent biweekly knowledge sharing amongst the team. Also, whenever assistance was needed, team members would be all hands on deck to help triage or debug any errors in the code or application.

  Paired programming sessions were held amongst the whole team to help develop the following:

  - Node.js front end views and page
  - SQL Advanced Queries
  - SQL Stored Procedure
  - SQL Trigger