📖🎟️🍿Entertainment Engine 🚂🚆

Group Members

Name	***Email***	***GitHub Username***
Eduardo Gonzalez	edgoze@seas	Edgoze
Leah Levin	leahl01@sas	leahl123
Sherie Pan	sheriep@sas	sheriep
Jeffrey Xiao	jxiao23@seas	jxiao

Credentials:

```
db_config = {
    "username" : "admin",
    "host": "https://database-1.cvx6rgg3nqsn.us-east-1.rds.amazonaws.com/",
    "port": "3306",
    "password": "cis4500!"
}
```

Complex SQL Queries:

Query 1:

The **search** page allows users to search for books/movies **by** book/movie **title**, **author**, **director** or **actor**. There will be one search bar where the user can input any of the above search items. The search result will be a view that will later be reused for filtering. Because it is a view, it will be recomputed on every search. Search results can later be filtered by rating, genre, media type (book or movie), and movie adaptation of book.

```
VIEW SearchResults:

WITH Matched_books AS (
SELECT ISBN, Title, 'book' AS Type, rating
FROM Books
WHERE Title LIKE '%{input}%'
),
Matched_movies AS (
SELECT Title, Movie_id, 'movie' AS Type
FROM Movies
WHERE Title LIKE '%{input}%'
),
Matched_authors AS (
SELECT BookISBN, AuthorName
FROM Writes
WHERE AuthorName LIKE '%{input}%'
),
Matched_directors AS (
SELECT Name, DirectorId, Movie_id
FROM Directs JOIN Directors ON Directs.DirectorId = Directors.Id
WHERE Name LIKE '%{input}%'
),
Matched_actors AS (
SELECT Name, ActorId, Movie_id
FROM Plays JOIN Actors on Plays.ActorId = Actors.Id
WHERE Name LIKE '%{input}%'
),
Movie_ratings AS (
SELECT MovieId, AVG(rating) as AverageRating, COUNT(DISTINCT UserId) as NumRaters
FROM Ratings
GROUP BY MovieId
),
Book_genres AS (
SELECT BookISBN, GROUP_CONCAT(GenreName ORDER BY GenreName) AS GenreList
FROM GenreOfBook
),
Movie_genres AS (
SELECT Movie_id, GROUP_CONCAT(GenreName ORDER BY GenreName) AS GenreList
FROM GenreOfMovie
),
Final_books AS (
SELECT B.ISBN as Id, B.Title, B.Type, B.Rating, G.GenreList
FROM Matched_books B
JOIN Matched_authors A ON B.ISBN = A.BookISBN
JOIN Book_genres G ON B.ISBN = G.BookISBN
),
Final_movies AS (
SELECT M.Movie_id as Id, M.Title, M.Type, M.AverageRating AS Rating, G.GenreList
FROM Matched_movies M
JOIN Matched_directors D ON M.Movie_id = D.Movie_id
JOIN Matched_actors A ON M.Movie_id = A.Movie_id
JOIN Movie_genres G ON M.Movie_id = G.Movie_id
)
(SELECT Id, Title, Type, Rating, GenreList
FROM Final_books)
UNION
(SELECT Id, Title, Type, Rating, GenreList
FROM Final_movies);
```

Query 2:

The recommendation page will ask users to fill out a form to generate 10 recommended books and/or movies. The form contains the following fields:

- Media (users check one or more of book and movie)
- Genre (users check up to 10 genres they are interested in)
- Avg rating (users choose a minimum avg rating on a slider)
- Num raters (users select a minimum number of ratings out of a set of options we provide) - using aggregate operator

Books/movies that match the greatest number of the selected genres are returned. If the user selects both book and movie, we recommend 5 movies and 5 books, using the following SQL query:

```
WITH books_genres AS (
SELECT BookISBN, COUNT(*) AS GenresMatched
FROM GenreOfBook
WHERE GenreName IN ('Genre1', 'Genre2', 'Genre3', 'etc')
GROUP BY BookISBN
ORDER BY GenresMatched DESC
),
movies_genres AS (
SELECT Movie_id, COUNT(*) AS GenresMatched
FROM GenreOfMovie
WHERE GenreName IN ('Genre1', 'Genre2', 'Genre3', 'etc')
GROUP BY Movie_id
ORDER BY GenresMatched DESC
),
Movie_ratings AS (
SELECT MovieId, AVG(rating) as AverageRating, COUNT(DISTINCT UserId) as
NumRaters
FROM Ratings
GROUP BY MovieId
HAVING AverageRating >= '{inputRating}' AND NumRaters >= '{inputNumRaters}'
),
Five_books AS (
SELECT Title, ISBN AS Id, 'book' as Type
FROM Books A
JOIN (SELECT BookISBN FROM books_genres) B ON A.ISBN = B.BookISBN
WHERE Rating >= '{inputRating}'
LIMIT 5
),
Five_movies AS (
SELECT Title, A.Movie_id AS Id, 'movie' as Type
FROM Movies A
JOIN (SELECT MovieId FROM Movie_ratings) R ON A.Movie_id = R.MovieId
JOIN movies_genres G ON A.Movie_id = G.Movie_id
LIMIT 5
)
(SELECT Title, Id, Type
FROM Five_books)
UNION
(SELECT Title, Id, Type
FROM Five_movies)
```

Query 3:

In the detailed view page for each movie, we display other books/movies that are similar. This includes:

- Movie by same director
- Movie with same actor

```
WITH Director_movies AS (
SELECT Movie_id, COUNT(*) AS numSimilarDirectors
FROM Directs
WHERE DirectorId IN ('Director1', 'Director2', 'etc.')
GROUP BY DirectorId
ORDER BY numSimilarDirectors DESC
),
Actor_movies AS (
SELECT Movie_id, COUNT(*) AS numSimilarActors
FROM Plays
WHERE ActorId IN ('Actor1', 'Actor2', 'etc.')
GROUP BY ActorId
ORDER BY numSimilarActors DESC
)
SELECT A.Movie_id, A.numSimilarActors + B.numSimilarDirectors AS numSimilar
FROM Actor_movies A JOIN Director_movies B ON A.Movie_id = B.Movie_id
ORDER BY numSimilar DESC
LIMIT 10;
```

Query 4:
Page showing the director(s) with the highest average rated movies, only considering movies with at least 2 raters, and directors that have at least 2 movies that have at least 2 raters.

```
WITH MovieRatings AS (
SELECT MovieId, AVG(rating) as AverageRating, COUNT (DISTINCT UserId) as NumRaters
FROM Ratings
GROUP BY MovieId
HAVING NumRaters >= 2
),
DirectorStats AS (
SELECT DirectorId, AVG(AverageRating) AS DirectorAvgRating
FROM Directs D
JOIN MovieRatings M ON D.Movie_id = M.MovieId
GROUP BY DirectorId
HAVING COUNT(*) >= 2
),
HighestDirectors AS (
SELECT DirectorId
FROM DirectorStats
WHERE DirectorAvgRating >= ALL (
   SELECT DirectorAvgRating
  FROM DirectorStats
   )
),
DirectorBestRating AS (
SELECT H.DirectorId AS DirectorId, MAX(AverageRating) as max_rating
FROM HighestDirectors H
JOIN Directs D ON H.DirectorId = D.DirectorId
JOIN MovieRatings M ON D.movie_id = M.MovieId
GROUP BY DirectorId
),
BestMovies AS (
SELECT DirectorId, Movie_id
FROM DirectorBestRating
JOIN Directs D ON DirectorBestRating.DirectorId = D.DirectorId
JOIN MovieRatings M ON D.movie_id = M.MovieId
WHERE M.AverageRating = DirectorBestRating.max_rating
),
OneBestMoviePerDirector AS (
SELECT DirectorId, Movie_id
FROM (SELECT * FROM BestMovies ORDER BY RAND())
GROUP BY DirectorId
)
SELECT D.name, M.title
```

```
FROM OneBestMoviePerDirector O
JOIN Directors D ON D.Id = O.DirectorId
JOIN Movies M ON M.movie_id = O.movie_id
```

Schema: DirectorName, MovieId, Title

Simple SQL Queries:

Query 1: Get the row from movie dataset based on key

```
SELECT Title, Overview
FROM Movies
WHERE Movie_id = {id};
```

Query 2: Get genres of a movie.

```
SELECT GenreName
FROM GenreOfMovie
WHERE Movie_id = {id};
```

Query 3: Get the name and gender of the directors of a movie.

```
SELECT Name, Gender
FROM Directs
JOIN Directors ON Directs.DirectorId = Directors.Id
WHERE Movie_id = {id};
```

Query 4: Get the name, character, and gender of the actors of a movie.

```
SELECT Name, PlaysCharacter, Gender
FROM Plays JOIN Actors ON Plays.ActorId = Actors.Id
WHERE Movie_id = {id};
```

Query 5: Get the row from the book dataset based on key.

```
SELECT AuthorName, ImageURL, Description, Title, Rating, NumPages,
GoodreadsLink
FROM Books JOIN Writes ON Books.ISBN = Writes.BookISBN
WHERE ISBN = {isbn};
```

Query 6: Get genres of a book.

```
SELECT GenreName
FROM GenreOfBook
WHERE BookISBN = {isbn};
```

Query 7: Returns books and movies where there is a movie based on the book

```
SELECT A.Title, A.ISBN, B.Movie_id
FROM Books A JOIN Movies B ON A.Title = B.Title;
```

Schema Normalization:

- Books(ISBN, Image, Description, Title, Rating, Num_pages, Goodreads_links)
  - F_{Books}^{+} = { ISBN -> Image, Description, Title, Rating, Num_pages, Goodreads_links } (ignoring trivial dependencies)
  - Candidate Key is ISBN, so this is in BCNF and 3NF
- Author(Name)
  - $F_{Authors}^{+}$ = { } (ignoring trivial dependencies)
  - so this is in BCNF and 3NF
- Movies(MovieId, Title, Overview, Rating, NumRaters)
  - F_{Movies}^{+} = { MovieId -> Title, Overview, Rating, NumRaters } (ignoring trivial dependencies)
  - Candidate Key is MovieId, so this is in BCNF and 3NF
- Genres(name)
  - F_{Genres}^{+} = { } (ignoring trivial dependencies)
  - so this is in BCNF and 3NF
- Actors(Id, Name, Gender)
  - F_{Actors}^{+} = { Id -> Name, Gender} (ignoring trivial dependencies)
  - Candidate Key is Id, so this is in BCNF and 3NF
- Directors(Id, Name, Gender)
  - $F_{Directors}^{+}$ = { Id -> Name, Gender} (ignoring trivial dependencies)
  - Candidate Key is Id, so this is in BCNF and 3NF
- Writes(AuthorName, BookISBN)

```
        AuthorName FOREIGN KEY References Author(Name)
        BooksISBN FOREIGN KEY References Books(ISBN)
CREATE ASSERTION
In Writes
CHECK (NOT EXISTS
                (SELECT *
                FROM Books
                WHERE ISBN NOT IN
                        (SELECT BookISBN
                    FROM Writes))
```

- F_{Actors}^{+} = { BookISBN -> AuthorName} (ignoring trivial dependencies)
- Candidate Key is BookISBN, so this is in BCNF and 3NF

- GenreOfBook(BookISBN, GenreName)

  BookISBN FOREIGN KEY References Books(ISBN)
  GenreName FOREIGN KEY References Genre(name)

  - $F_{GenreOfBook}^{+}$ = { BookISBN -> GenreName} (ignoring trivial dependencies)
  - Candidate Key is BookISBN, so this is in BCNF and 3NF

- GenreOfMovie(MovieId, GenreName)

  MovieId FOREIGN KEY References Movie(MovieId)
  GenreName FOREIGN KEY References Genre(name)

  - $F_{GenreOfMovie}^{+}$ = { MovieId -> GenreName} (ignoring trivial dependencies)
  - Candidate Key is MovieId, so this is in BCNF and 3NF

- Plays(ActorId, MovieId, Character)

  ActorId FOREIGN KEY References Actors(id)
  MovieId FOREIGN KEY References Movie(MovieId)

  - F_{Plays}^{+} = { } (ignoring trivial dependencies)
  - so this is in BCNF and 3NF

- Directs(DirectorId, MovieId)

  DirectorId FOREIGN KEY References Directors(id)
  MovieId FOREIGN KEY References Movie(sMovieId)

```
CREATE ASSERTION
In Directs
CHECK (NOT EXISTS
        (SELECT *
        FROM Directors
        WHERE id NOT IN
                (SELECT DirectorId
                FROM Directs))
```

  - $F_{Directors}^{+}$ = { } (ignoring trivial dependencies)
  - so this is in BCNF and 3NF

- Ratings(MovieId, UserId, Rating)

  MovieId FOREIGN KEY References Movie(MovieId)

  - $F_{Ratings}^{+}$ = { MovieI, UserId -> Rating } (ignoring trivial dependencies)
  - so this is in BCNF and 3NF