FALL 2023

MOVIE AND MUSIC DATABASE

TECHNICAL REPORT

CPS510
PROFESSOR: DR. A. ABHARI
TA: JORGE LOPEZ (7)
GROUP 12

LAVANYA CHHILWAR
MAHIR SOUFIAN
MARIA LABEEBA

Table of Contents

1. Introduction

Topic: Movie and Music Database System

1.1 Description & Results Expected

Our movie and music database system is a collection of information and user feedback used by different audiences to find movies and music that appeal to them. Our system aims to collect summaries, ratings, and reviews for various films and music which will lead to more efficient and organized browsing for users. Regarding updating and inputting information, the admin will be responsible for ensuring all information is up-to-date and reliable.

1.2 Relationship between Entities

Entity	***Relationships***
Movie	● Movie has a rating (1 : N) ● Movie has movie People (N : M) ● Movie is part of a genre (N : M)
Music	● Music is part of an album (N :1) ● Music has a genre (N : M) ● Music has music people (N : M)
User	● User Searches for a Movie (N : M) ● User Searches for Music (N : M)
Admin	● Admin can edit Music (N : M) ● Admin can edit Movie (N : M)
Music People	● Music People make an album (N : 1)

1.3 Entities, Attributes & Data Type

1.3.1 Admin Entity

Attributes	***Data Type***
<u>Admin ID</u>	Integer
Username	String
Password	String

1.3.2 User Entity

Attributes	***Data Type***
User ID	Integer
Username	String
Password	String

1.3.3 Movie Entity

Attributes	***Data type***
Movie ID	Integer
Title	String
Release Date	Date
Average Rating	Float
Netflix	String
Amazon Prime	String
Apple TV	String

1.3.4 Movie People Entity

Attributes	***Data Type***
Movie People ID	Integer
Director Name	String
Actor Name	String

1.3.5 Genre Entity

Attributes	*Data Type*
Genre ID	Integer
Name	String

1.3.6 Album Entity

Attributes	*Data Type*

Album ID	Integer
Album Name	String
Number of Songs	Integer
Release Date	Integer

1.3.7 Music Entity

Attributes	***Data Type***
Music ID	Integer
Title	String
Release Date	Integer
Average Rating	Float
Spotify	String
Apple Music	String

1.3.8 Music People Entity

Attributes	***Data Type***
Music People ID	Integer
Artist Name	String
Producer Name	String

1.3.7 Rating Entity

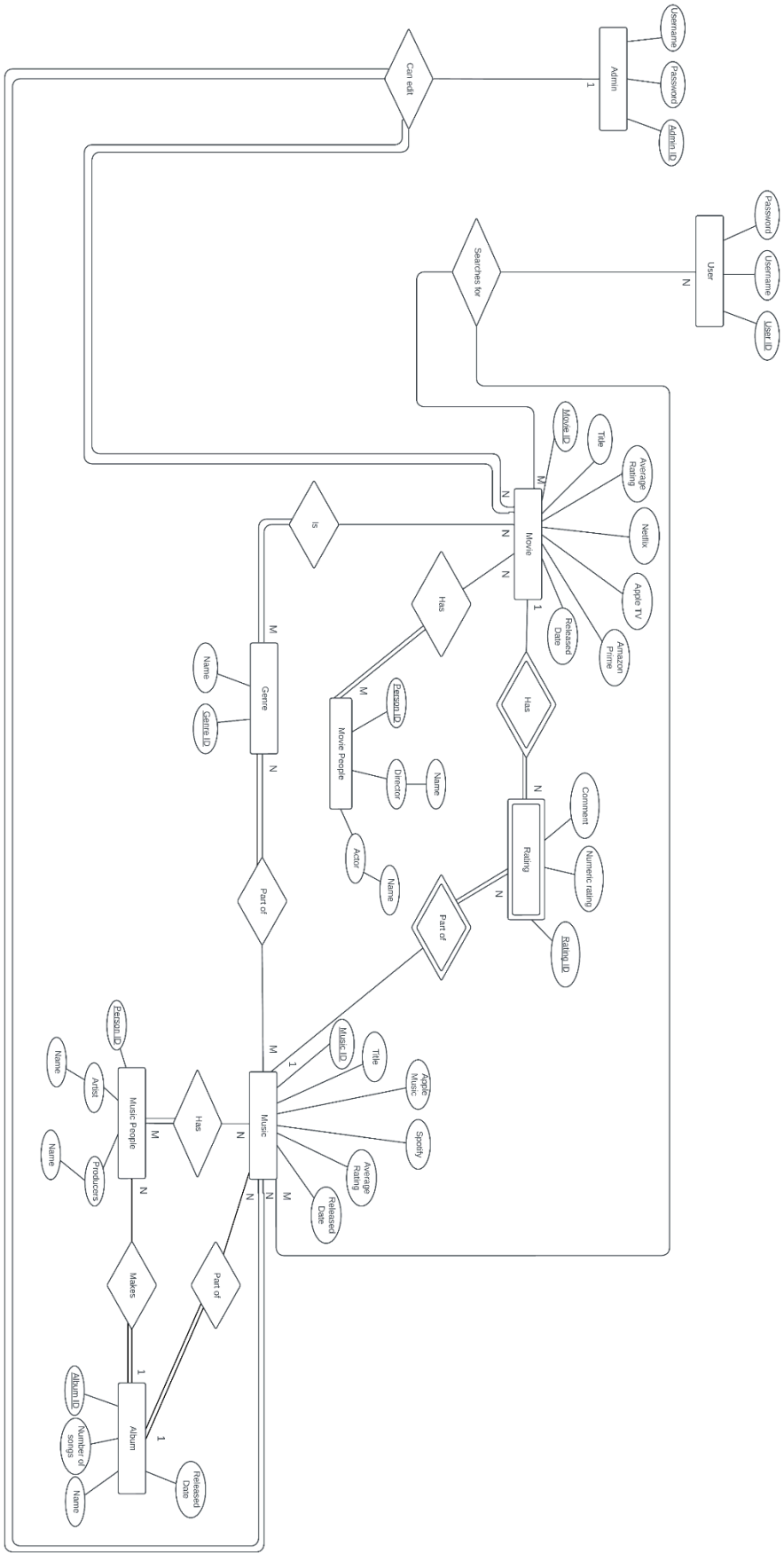
Attributes	***Data Type***
Rating ID	Integer
Numeric Rating	Float
Comments	String

1.4 Potential Functions

Function	*Description*

SearchMovie	User and Admin can browse the movie category
SearchMusic	User and Admin can browse the music category
ViewGuestInfo	Admin is able to view guest info
ViewMovieInfo	Allows the user to view the movie information
ViewMusicInfo	Allows the user to view the music information
ManageMusic	Admin is able to add necessary music info

2. ER Diagram


Admin
Username
Password
Admin ID
Can edit
1
User
Password
Username
User ID
Searches for
N
Movie
Movie ID
Title
Average Rating
Netflix
Apple TV
Amazon Prime
Released Date
M
N
Is
Has
Has
Genre
Name
Genre ID
Movie People
Person ID
Director
Name
Actor
Name
Rating
Comment
Numeric rating
Rating ID
Part of
Part of
Music
Music ID
Title
Apple Music
Spotify
Average Rating
Released Date
Has
Music People
Person ID
Name
Artist
Name
Producers
Makes
Part of
Album
Album ID
Number of songs
Name
Released Date

3. Schema Design and Database Construction

3.1 Color Key

Primary Key	Foreign Key	Attribute

3.2 Schema Design/Table Design

Admin [entity]		
admin id	username	password

User [entity]		
user id	username	password

Movie [entity]							
movie id	title	release date	average rating	netflix	amazon prime	apple tv	admin id

Movie People [entity]		
movie people id	director name	actor name

Genre [entity]	
genre id	name

Album [entity]			
album id	name	number of songs	release date

Music [entity]							
music id	title	release date	average rating	spotify	apple music	admin id	album id

Music People [entity]			
music people id	artist name	producer name	album id

Rating [weak entity]				
rating id	rate	comment	movie id	music id

Searches For Movie [N:M]	
user id	movie id

Searches For Music [N:M]	
user id	music id

Movie Has [N:M]	
movie id	movie people id

Movie Is [N:M]	
genre id	movie id

Genre Part Of [N:M]	
genre id	music id

Music Has [N:M]	
music id	music people id

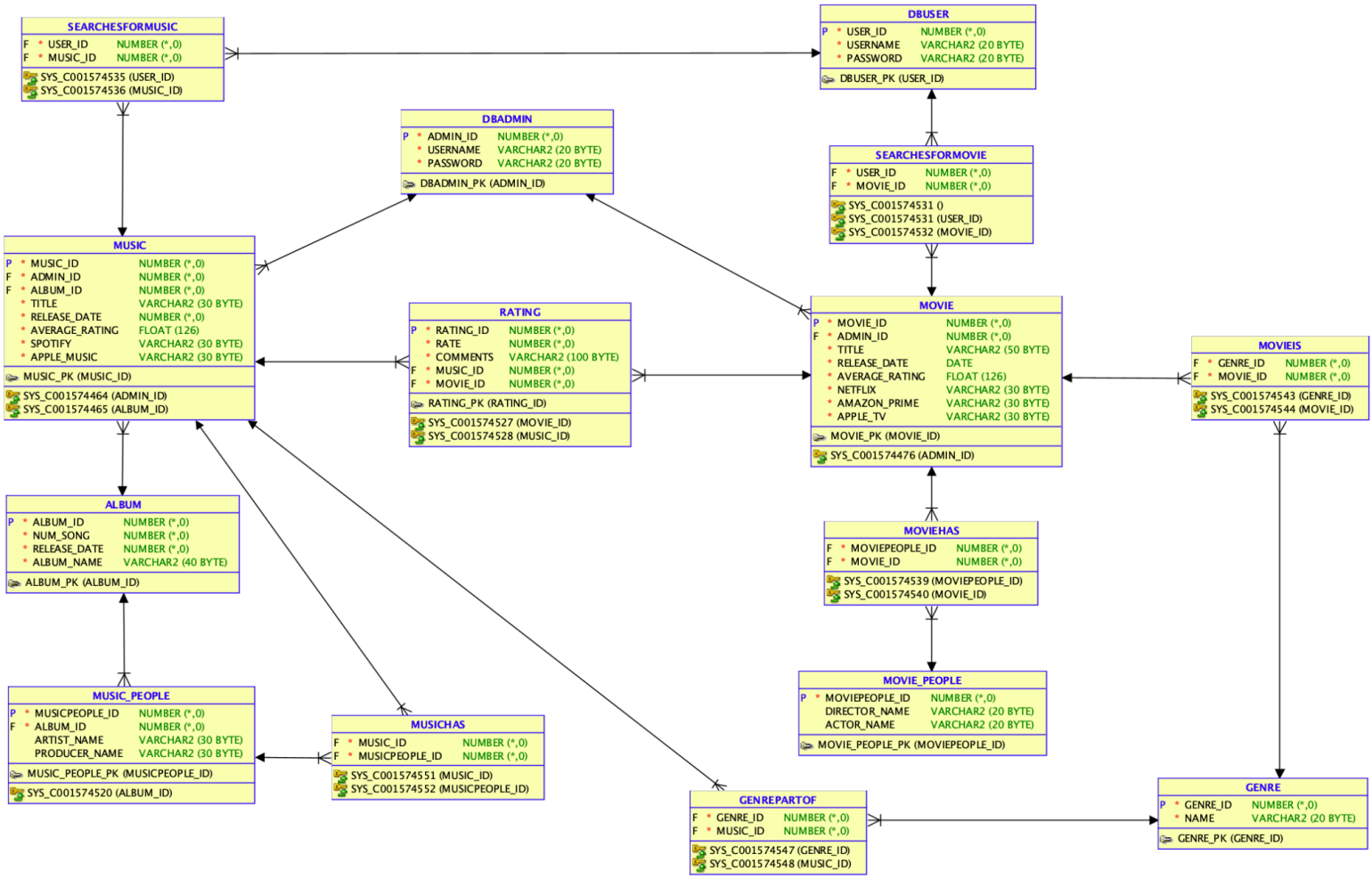
3.3 Database Construction

```
create table DBAdmin
(
   admin_id int not null,
   username varchar(50) not null,
   password varchar(50) not null,

   PRIMARY KEY(admin_id)
);

create table DBUser
(
   user_id int not null,
   username varchar(50) not null ,
   password varchar(50) not null,

   PRIMARY KEY (user_id)
);

create table Movie
(
   movie_id int not null,
   admin_id int not null,
   title varchar(200) not null,
   release_date date not null,
   average_rating float not null,
   netflix varchar(30) not null,
   amazon_prime varchar(30) not null,
   apple_tv varchar(30) not null,

   PRIMARY KEY (movie_id),
   FOREIGN KEY (admin_id) REFERENCES DBAdmin(admin_id)
);

create table Movie_People
(
   moviepeople_id int not null,
   director_name varchar(200),
   actor_name varchar(200),

   PRIMARY KEY (moviepeople_id)
);

create table Genre
(
   genre_id int not null,
   name varchar(50) not null,

   PRIMARY KEY (genre_id)
);
create table Album
(
```

```
    album_id int not null,
album_name varchar(200) not null,
    num_song int not null,
    release_date int not null,

    PRIMARY KEY (album_id)
);

create table Music
(
    music_id int not null,
    admin_id int not null,
    album_id int not null,
    title varchar(200) not null,
    release_date int not null,
    average_rating float not null,
    spotify varchar(30) not null,
    apple_music varchar(30) not null,

    PRIMARY KEY (music_id),
    FOREIGN KEY (admin_id) REFERENCES DBAdmin(admin_id),
    FOREIGN KEY (album_id) REFERENCES Album(album_id)
);

create table Music_People
(
    musicpeople_id int not null,
    album_id int not null,
    artist_name varchar(200),
    producer_name varchar(200),

    PRIMARY KEY (musicpeople_id),
    FOREIGN KEY (album_id) REFERENCES Album(album_id)
);

create table Rating -- weak entity
(
    rating_id int not null,
    rate float not null,
    comments varchar(700) not null,
    movie_id int not null,
    album_id int not null,

    PRIMARY KEY (rating_id),
    FOREIGN KEY (movie_id) REFERENCES Movie(movie_id),
    FOREIGN KEY (album_id) REFERENCES Album(album_id)
);

create table SearchesForMovie
(
    user_id int not null,
    movie_id int not null,
```

```
    PRIMARY KEY (user_id, movie_id),
    FOREIGN KEY (user_id) REFERENCES DBUser(user_id),
    FOREIGN KEY (movie_id) REFERENCES Movie(movie_id)
);
CREATE TABLE SearchesForMusic
(
    user_id int not null,
    music_id int not null,

    PRIMARY KEY (user_id, music_id),
    FOREIGN KEY (user_id) REFERENCES DBUser(user_id),
    FOREIGN KEY (music_id) REFERENCES Music(music_id)
);

create table MovieHas
(
    moviepeople_id int not null,
    movie_id int not null,

    PRIMARY KEY (moviepeople_id, movie_id),
    FOREIGN KEY (moviepeople_id) REFERENCES Movie_People(moviepeople_id),
    FOREIGN KEY (movie_id) REFERENCES Movie(movie_id)
);

create table MovieIs
(
    genre_id int not null,
    movie_id int not null,

    PRIMARY KEY (genre_id, movie_id),
    FOREIGN KEY (genre_id) REFERENCES Genre(genre_id),
    FOREIGN KEY (movie_id) REFERENCES Movie(movie_id)
);
create table GenrePartOf
(
    genre_id int not null,
    album_id int not null,

    PRIMARY KEY (genre_id, album_id),
    FOREIGN KEY (genre_id) REFERENCES Genre(genre_id),
    FOREIGN KEY (album_id) REFERENCES Album(album_id)
);

create table MusicHas
(
    music_id int not null,
    musicpeople_id int not null,

    PRIMARY KEY (music_id, musicpeople_id),
    FOREIGN KEY (music_id) REFERENCES Music(music_id),
    FOREIGN KEY (musicpeople_id) REFERENCES Music_People(musicpeople_id)
);
```

3.4 Schema Diagram



3.5 Populate Tables (Insert Statements)

You can access our insert statements through the link below-
https://docs.google.com/document/d/1rDJWDzInA5zo1nKNJCr0Fsa3bKcOm10kJMXEHRSXDLk/edit?usp=sharing

Furthermore our database can be viewed through this Excel sheet-
https://docs.google.com/spreadsheets/d/1W7OixWY_u2Cw4mw_2rF7WCImSQ3GVULQoJ86EZ9naJQ/edit?usp=sharing

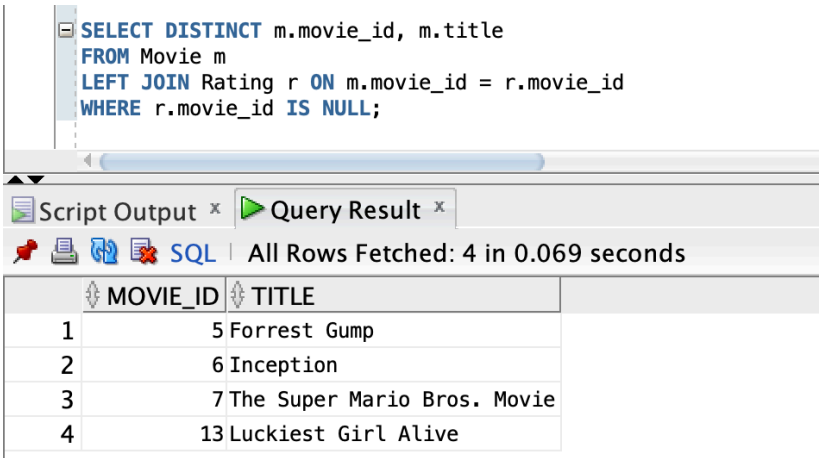
4. Designing Views/ Simple Quarries

4.1 Simple Queries

4.1.1 Query 1:

Output the list of movie IDs and movie titles that have no ratings (this searches the Movie and Ratings tables to see if movie_id is not present in the Rating table).

```
SELECT DISTINCT m.movie_id, m.title
FROM Movie m
LEFT JOIN Rating r ON m.movie_id = r.movie_id
WHERE r.movie_id IS NULL;
```



Script Output | Query Result

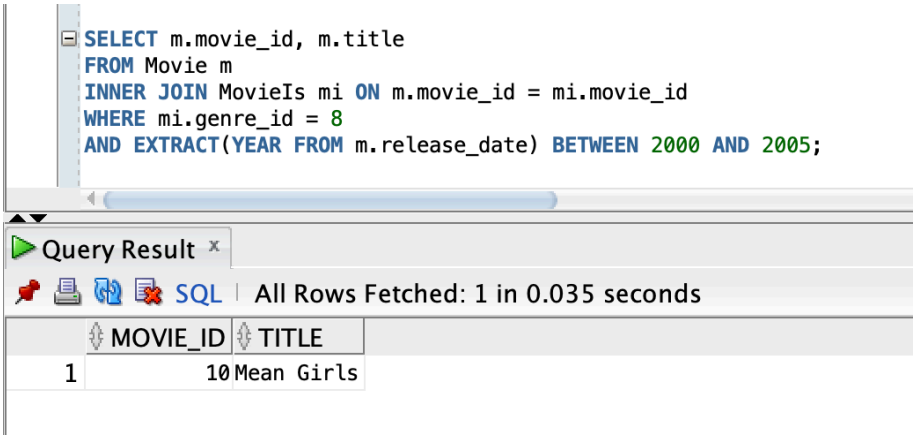
SQL | All Rows Fetched: 4 in 0.069 seconds

	MOVIE_ID	TITLE
1	5	Forrest Gump
2	6	Inception
3	7	The Super Mario Bros. Movie
4	13	Luckiest Girl Alive

4.1.2 Query 2:

Outputting a list of movie IDs and titles that have the genre, Comedy (Genre ID: 8) and were released between 2000-2005. (This searches the Movie, Genre and MovieIs table to see which Movie is of the genre Comedy).

```
SELECT m.movie_id, m.title
FROM Movie m
INNER JOIN MovieIs mi ON m.movie_id = mi.movie_id
WHERE mi.genre_id = 8
AND EXTRACT(YEAR FROM m.release_date) BETWEEN 2000 AND 2005;
```



Query Result

SQL | All Rows Fetched: 1 in 0.035 seconds

	MOVIE_ID	TITLE
1	10	Mean Girls

4.1.3 Query 3:

Outputting a list of movies that are not available on Apple TV, but are available on Netflix and Amazon Prime (This checks the movie table to see which movies are available on which platform).

SELECT movie_id, title
FROM Movie
WHERE apple_tv = 'No' AND amazon_prime = 'Yes' AND netflix = 'Yes';

```
SELECT movie_id, title
FROM Movie
WHERE apple_tv = 'No' AND amazon_prime = 'Yes' AND netflix = 'Yes';
```

Query Result

SQL | All Rows Fetched: 1 in 0.031 seconds

	MOVIE_ID	TITLE
1	5	Forrest Gump

4.1.4 Query 4:

Song titles which were released before 2010 and have a title longer than 10 characters and are available on spotify. (Checks Music columns release date, title, and spotify availability).

SELECT title
FROM music
WHERE release_date <= 2010
AND LENGTH(title) >10
AND spotify = 'No';

```
SELECT title
FROM music
WHERE release_date <= 2010
    AND LENGTH(title) > 10
    AND spotify = 'No';
```

Query Result

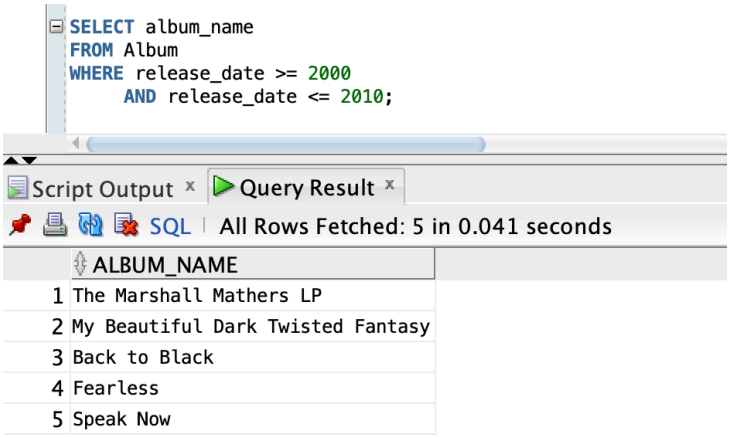
SQL | All Rows Fetched: 7 in 0.038 seconds

	TITLE
1	The Girl Is Mine
2	P.Y.T. (Pretty Young Thing)
3	Baby Be Mine
4	Human Nature
5	I Was Made to Love Her
6	A Thing or Two
7	Aren't You Glad

4.1.5 Query 5:

Albums that were released between 2000 and 2010. (This checks the Album table to see the release years).

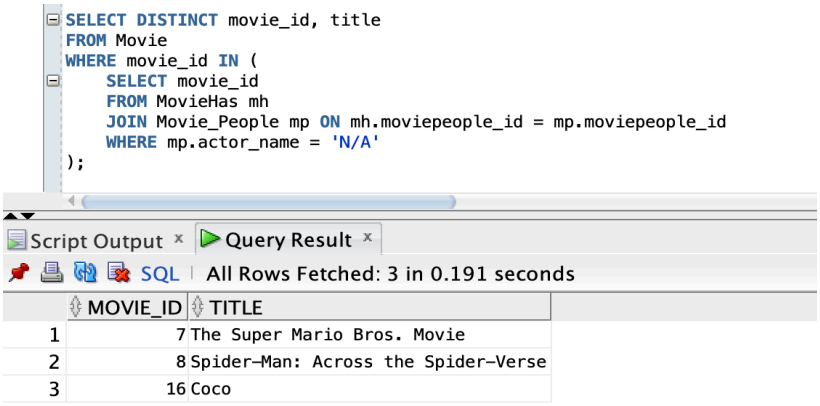
```
SELECT album_name
FROM Album
WHERE release_date >= 2000
    AND release_date <= 2010;
```



4.1.6 Query 6:

Give a list of animated movies. (This will search the MovieHas and MoviePeople tables for actor_name = N/A, as if there is no actor, the movie is animated in the database).

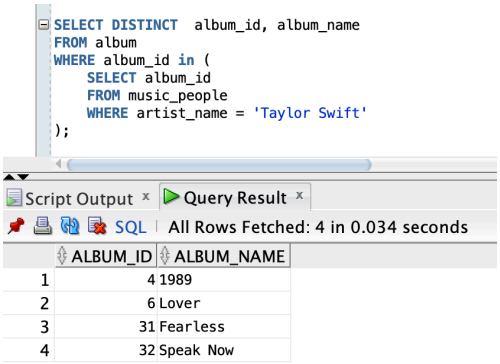
```
SELECT DISTINCT movie_id, title
FROM Movie
WHERE movie_id IN (
   SELECT movie_id
   FROM MovieHas mh
   JOIN Movie_People mp ON mh.moviepeople_id = mp.moviepeople_id
   WHERE mp.actor_name = 'N/A'
);
```



4.1.7 Query 7:

Lists the name of the albums by Taylor Swift. (This looks through the Album and Music_People tables to find which album is by Taylor Swift).

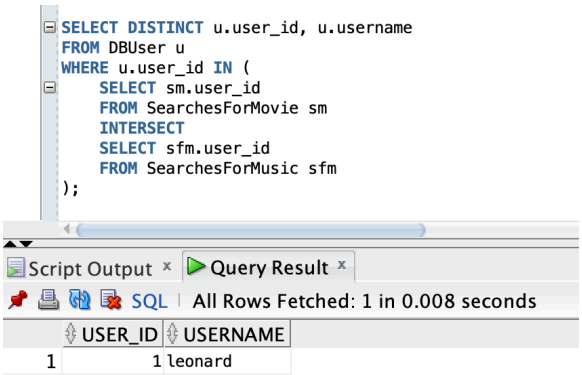
```
SELECT DISTINCT  album_id, album_name
FROM album
WHERE album_id in (
   SELECT album_id
   FROM music_people
   WHERE artist_name = 'Taylor Swift'
);
```



4.1.8 Query 8:

Who are the users that have searched for both movies and music. (This looks in the DBUser, SearchesForMovie and SearchesForMusic tables to see which user_id is present in both).

```
SELECT DISTINCT u.user_id, u.username
FROM DBUser u
WHERE u.user_id IN (
   SELECT sm.user_id
   FROM SearchesForMovie sm
   INTERSECT
   SELECT sfm.user_id
   FROM SearchesForMusic sfm
);
```



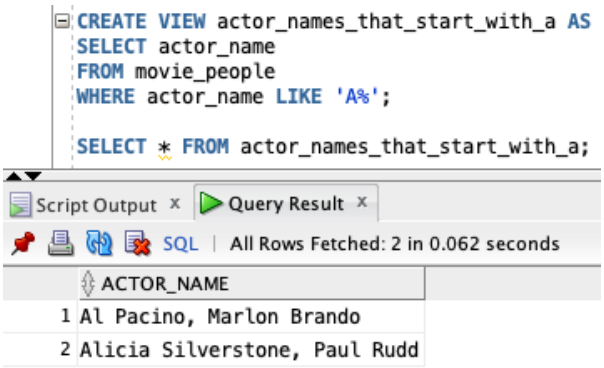
4.2 View Designs

4.2.1 View 1:

Movie actor names that start with the letter A

```
CREATE VIEW actor_names_that_start_with_a AS
SELECT actor_name
FROM movie_people
WHERE actor_name LIKE 'A%';
SELECT * FROM actor_names_that_start_with_a;
```

SQL Output:

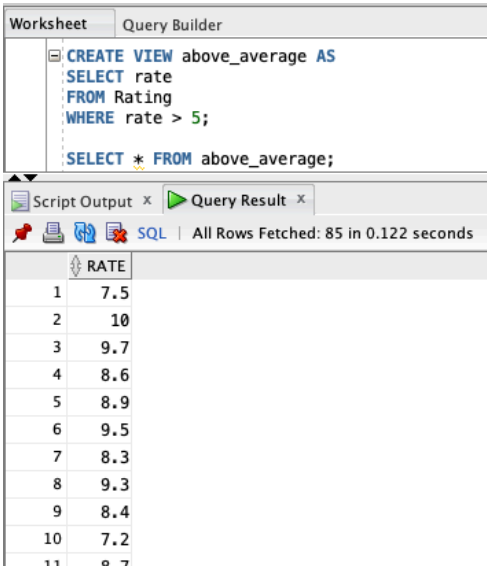


4.2.2 View 2:

View ratings that are above 5/10

```
CREATE VIEW above_average AS
SELECT rate
FROM Rating
WHERE rate > 5;
SELECT * FROM above_average;
```

SQL Output:

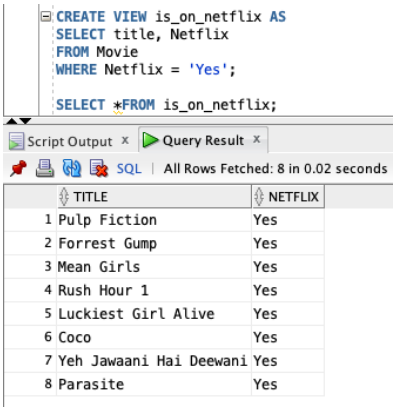


4.2.3 View 3:

View movies that are on netflix

```
CREATE VIEW is_on_netflix AS
SELECT title, Netflix
FROM Movie
WHERE Netflix = 'Yes';

SELECT *FROM is_on_netflix;
```

SQL Output:



```
CREATE VIEW is_on_netflix AS
SELECT title, Netflix
FROM Movie
WHERE Netflix = 'Yes';

SELECT *FROM is_on_netflix;
```

Script Output | Query Result

SQL | All Rows Fetched: 8 in 0.02 seconds

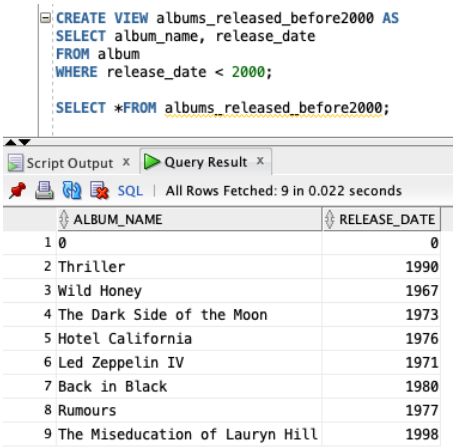
	TITLE	NETFLIX
1	Pulp Fiction	Yes
2	Forrest Gump	Yes
3	Mean Girls	Yes
4	Rush Hour 1	Yes
5	Luckiest Girl Alive	Yes
6	Coco	Yes
7	Yeh Jawaani Hai Deewani	Yes
8	Parasite	Yes

4.2.4 View 4:

View albums released before the year 2000

```
CREATE VIEW albums_released_before2000 AS
SELECT album_name, release_date
FROM album
WHERE release_date < 2000;

SELECT *FROM albums_released_before2000;
```

SQL Output:



```
CREATE VIEW albums_released_before2000 AS
SELECT album_name, release_date
FROM album
WHERE release_date < 2000;

SELECT *FROM albums_released_before2000;
```

Script Output | Query Result

SQL | All Rows Fetched: 9 in 0.022 seconds

	ALBUM_NAME	RELEASE_DATE
1	0	0
2	Thriller	1990
3	Wild Honey	1967
4	The Dark Side of the Moon	1973
5	Hotel California	1976
6	Led Zeppelin IV	1971
7	Back in Black	1980
8	Rumours	1977
9	The Miseducation of Lauryn Hill	1998

5. Advanced Queries by Unix shell Implementation

5.1 Advanced Queries

5.1.1 Query 1:

Minimum and Maximum rating for movies.

SELECT MIN(rate), MAX(rate)
FROM rating
WHERE movie_id <> 0;

Unix Shell Script Code:

```
# Query 10 (Advanced)
SET FEEDBACK OFF
COLUMN AVGSONGSPERALBUM FORMAT 999
SELECT AVG(num_song) AS avgsongsperalbum
FROM album;
```

Unix Shell Script Output:

```
MIN(RATE) MAX(RATE)
--------- ---------
        0        10
```

5.1.2 Query 2:

Average number of songs per album

SELECT AVG(num_song) AS avgsongsperalbum
FROM album;

Unix Shell Script Code:

```
# Query 10 (Advanced)
SET FEEDBACK OFF
COLUMN AVGSONGSPERALBUM FORMAT 999
SELECT AVG(num_song) AS avgsongsperalbum
FROM album;
```

Unix Shell Script Output:

```
AVGSONGSPERALBUM
----------------
              12
```

5.1.3 Query 3:

Songs and movies which have the same rating and are between 0 and 7

```
SELECT DISTINCT movie.average_rating, music.title, movie.title
FROM movie
INNER JOIN music ON movie.average_rating = music.average_rating
AND movie.average_rating <= 7
AND movie.average_rating > 0;
```

Unix Shell Script Code:

```
# Query 11 (Advanced)
SET FEEDBACK OFF
COLUMN AVERAGE_RATING FORMAT 999
COLUMN TITLE FORMAT A30
COLUMN TITLE_1 FORMAT A30
SELECT DISTINCT movie.average_rating, music.title, movie.title
FROM movie
INNER JOIN music ON movie.average_rating = music.average_rating
AND movie.average_rating <= 7
AND movie.average_rating > 0;
```

Unix Shell Script Output:

```
AVERAGE_RATING TITLE                          TITLE
-------------- ------------------------------ ---------------
             7 Awake                          Clueless
             7 Love Yourself                  Clueless
             7 Jopping                        Clueless
             7 Blood Sweat and Tears          Clueless
```

5.2 Interesting Queries with keywords

5.2.1 Query 1:

Using EXISTS to find movies which have a rating

```
SELECT movie_id, title
FROM Movie m
WHERE EXISTS (
   SELECT 1
   FROM Rating r
   WHERE r.movie_id = m.movie_id
);
```

Unix Shell Script Code:

```
# Query 12 (Interesting)
SET FEEDBACK OFF
SET PAGESIZE 1000
COLUMN MOVIE_ID FORMAT 999
COLUMN TITLE FORMAT A50
SELECT movie_id, title
FROM Movie m
WHERE EXISTS (
    SELECT 1
    FROM Rating r
    WHERE r.movie_id = m.movie_id
);
```

Unix Shell Script Output:

```
MOVIE_ID TITLE
-------- --------------------------------------------------
       1 Barbie
       2 The Shawshank Redemption
       3 Pulp Fiction
       4 The Godfather
       8 Spider-Man: Across the Spider-Verse
       9 Talk to Me
      10 Mean Girls
      11 Clueless
      12 Rush Hour 1
      14 La La Land
      15 The Greatest Showman
      16 Coco
      17 Midsommar
      18 Red, White and Royal Blue
      19 Yeh Jawaani Hai Deewani
      20 Parasite
       0 NA
```

5.2.2 *Query 2:*

Using UNION keyword we combine the movie and music titles and find titles which have “z” in them

```
SELECT name
FROM
(
   SELECT title AS name FROM Movie
   UNION
   SELECT title AS name FROM Music
)
WHERE name LIKE '%z%';
```

Unix Shell Script Code:

```
# Query 13 (Interesting)
SET FEEDBACK OFF
COLUMN NAME FORMAT A30
SELECT name
FROM
(
    SELECT title AS name FROM Movie
    UNION
    SELECT title AS name FROM Music
)
WHERE name LIKE '%z%';
```

Unix Shell Script Output:

```
NAME
------------------------------
Crazy Over You
Fertilizer
Institutionalized
```

5.2.3 Query 3:

Using GROUP BY to find the the average rating of movies which are on Amazon Prime, Apple TV, and Netflix

SELECT netflix, amazon_prime, apple_tv, AVG(average_rating) AS average_rating
FROM Movie
GROUP BY netflix, amazon_prime, apple_tv;

Unix Shell Script Code:

```
# Query 14 (Interesting)
SET FEEDBACK OFF
SET PAGESIZE 1000

COLUMN NETFLIX FORMAT A12
COLUMN AMAZON_PRIME FORMAT A12
COLUMN APPLE_TV FORMAT A12
COLUMN AVERAGE_RATING FORMAT 999
SELECT netflix, amazon_prime, apple_tv, AVG(average_rating) AS average_rating
FROM Movie
GROUP BY netflix, amazon_prime, apple_tv;
```

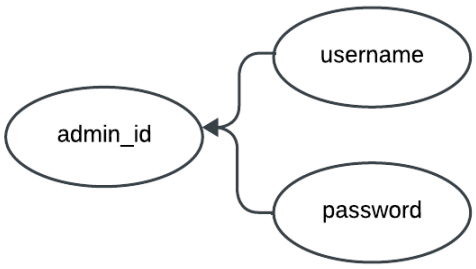
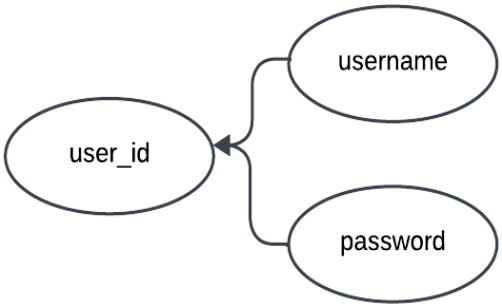
Unix Shell Script Output:

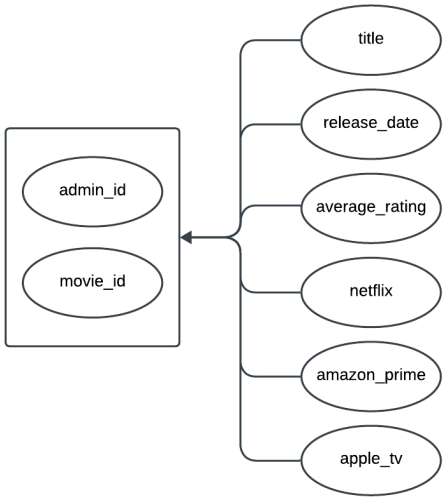
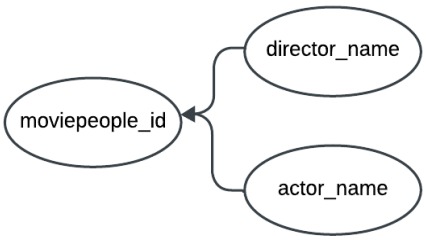
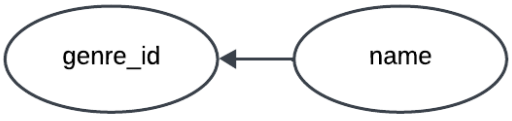
```
NETFLIX      AMAZON_PRIME APPLE_TV     AVERAGE_RATING
------------ ------------ ------------ --------------
No           Yes          No                        4
Yes          Yes          Yes                       8
Yes          No           No                        4
NA           NA           NA                        0
No           Yes          Yes                       8
Yes          Yes          No                        0
```

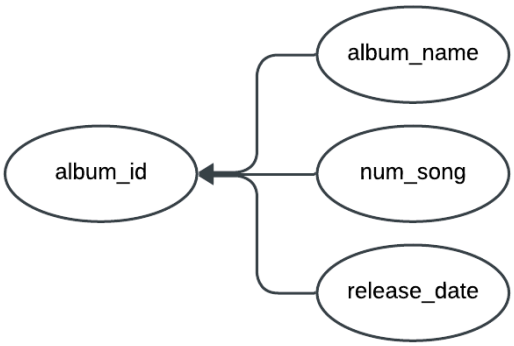
5.3 Unix Shell Code

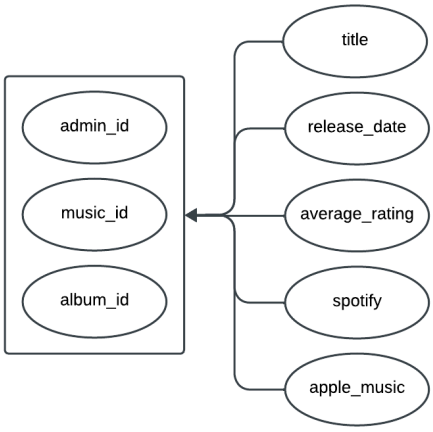
You can access the unix shell scripts through the link below - https://docs.google.com/document/d/10lGsI5fk-D0AjOCJp25nxlM_bNO-561fWslM63AovEw/edit#heading=h.uptv8xhoi0gg

6. Normalization and Functional Dependencies

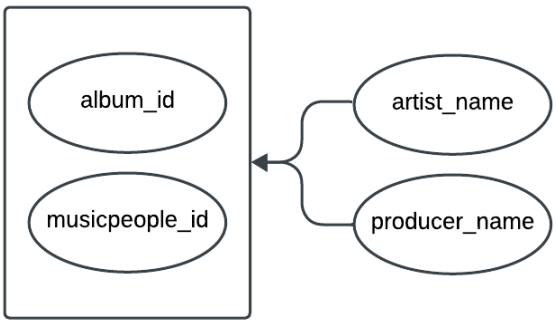
Relational Table	Normalization
create table DBAdmin (admin_id int not null, username varchar(50) not null, password varchar(50) not null, PRIMARY KEY(admin_id));  	**Functional Dependency** admin_id → username, password 1NF - There are no repeating groups and all values are atomic. 2NF - Non prime attributes are dependent on the primary key (admin_id). 3NF - No transitive dependencies; all attributes directly relate to the primary key. BCNF - DBAdmin is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (admin_id).
create table DBUser (user_id int not null, username varchar(50) not null , password varchar(50) not null, PRIMARY KEY (user_id));  	**Functional Dependency** user_id → username, password 1NF - There are no repeating groups and all values are atomic. 2NF - Non prime attributes are dependent on the primary key (user_id). 3NF - No transitive dependencies; all attributes directly relate to the primary key. BCNF - DBUser is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (user_id).
create table Movie (movie_id int not null, admin_id int not null, title varchar(200) not null, release_date date not null, average_rating float not null, netflix varchar(30) not null, amazon_prime varchar(30) not null, apple_tv varchar(30) not null,	**Functional Dependency** movie_id → title, release_date, average_rating, netflix, amazon_prime, apple_tv, admin_id 1NF - There are no repeating groups and all values are atomic. 2NF - Non prime attributes are dependent on the primary key (movie_id).

PRIMARY KEY (movie_id), FOREIGN KEY (admin_id) REFERENCES DBAdmin(admin_id));  	3NF - No transitive dependencies; all attributes directly relate to the primary key. BCNF - Movie is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (movie_id)
create table Movie_People (moviepeople_id int not null, director_name varchar(200), actor_name varchar(200), PRIMARY KEY (moviepeople_id));  	**Functional Dependency** moviepeople_id → director_name, actor_name 1NF - There are no repeating groups and all values are atomic. 2NF - Non prime attributes are dependent on the primary key (moviepeople_id). 3NF - No transitive dependencies; all attributes directly relate to the primary key. BCNF - Movie_people is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (moviepeople_id)
create table Genre (genre_id int not null, name varchar(50) not null, PRIMARY KEY (genre_id));  	**Functional Dependency** genre_id → name 1NF - There are no repeating groups and all values are atomic. 2NF - Non prime attributes are dependent on the primary key (genre_id). 3NF - No transitive dependencies; all attributes directly relate to the primary key.

	BCNF - Genre is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (user_id, movie_id).
create table Album (album_id int not null, album_name varchar(200) not null, num_song int not null, release_date int not null, PRIMARY KEY (album_id));  	**Functional Dependency** album_id → album, actor_name, num_song, release_date 1NF - There are no repeating groups and all values are atomic. 2NF - Non prime attributes are dependent on the primary key (album_id). 3NF - No transitive dependencies; all attributes directly relate to the primary key. BCNF - Album is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (album_id)
create table Music (music_id int not null, admin_id int not null, album_id int not null, title varchar(200) not null, release_date int not null, average_rating float not null, spotify varchar(30) not null, apple_music varchar(30) not null, PRIMARY KEY (music_id), FOREIGN KEY (admin_id) REFERENCES DBAdmin(admin_id), FOREIGN KEY (album_id) REFERENCES Album(album_id));	**Functional Dependency** music_id → title, release_date, average_rating, spotify, apple_music, admin_id, album_id 1NF - There are no repeating groups and all values are atomic. 2NF - Non prime attributes are dependent on the primary key (music_id). 3NF - No transitive dependencies; all attributes directly relate to the primary key. BCNF - Music is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (music_id)




```
create table Music_People
(
   musicpeople_id int not null,
   album_id int not null,
   artist_name varchar(200),
   producer_name varchar(200),

   PRIMARY KEY (musicpeople_id),
   FOREIGN KEY (album_id) REFERENCES
Album(album_id) );
```



Functional Dependency
musicpeople_id → artist_name, producers_name

1NF - Contains atomic values: moviepeople_id, director_name, actor_name.

2NF - No partial dependencies; all attributes depend on the primary key (moviepeople_id).

3NF - No transitive dependencies; all attributes directly relate to the primary key.

BCNF - Music_people is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (musicpeople_id)

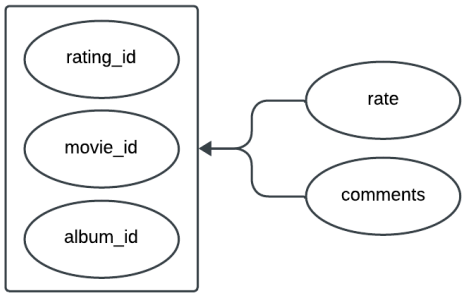
```
create table Rating -- weak entity
(
   rating_id int not null,
   rate float not null,
   comments varchar(700) not null,
   movie_id int not null,
   album_id int not null,

   PRIMARY KEY (rating_id),
   FOREIGN KEY (movie_id) REFERENCES
Movie(movie_id),
   FOREIGN KEY (album_id) REFERENCES
Album(album_id) );
```

Functional Dependency
rating_id → rate, comments, movie_id, album_id

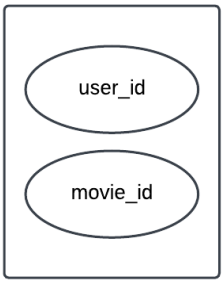
1NF: Contains atomic values: rating_id, rate, comments, movie_id, album_id.

2NF: No partial dependencies; all attributes depend on the primary key (rating_id).

3NF: No transitive dependencies; all attributes directly relate to the primary key.

<table>
<tr>
<td> 
</td>
<td>BCNF - Rating is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (rating_id)</td>
</tr>
<tr>
<td>create table SearchesForMovie
(
user_id int not null,
movie_id int not null,

PRIMARY KEY (user_id, movie_id),
FOREIGN KEY (user_id) REFERENCES DBUser(user_id),
FOREIGN KEY (movie_id) REFERENCES Movie(movie_id));

 
</td>
<td>Functional Dependency
user_id → movie_id
movie_id → user_id

1NF - There are no repeating groups and all values are atomic.

2NF - Non prime attributes are dependent on the composite primary key.

3NF - No transitive dependencies; all attributes directly relate to the primary key within their own table. (user → DBUser, movie_id → Movie)

BCNF - SearchesForMovie is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (user_id, movie_id).</td>
</tr>
<tr>
<td>CREATE TABLE SearchesForMusic
(
user_id int not null,
music_id int not null,

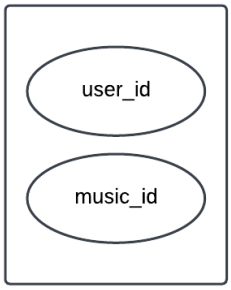
PRIMARY KEY (user_id, music_id),
FOREIGN KEY (user_id) REFERENCES DBUser(user_id),
FOREIGN KEY (music_id) REFERENCES Music(music_id));</td>
<td>Functional Dependency
user_id → music_id
music_id → user_id

1NF - There are no repeating groups and all values are atomic.

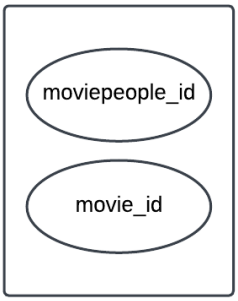
2NF - Non prime attributes are dependent on the composite primary key.

3NF - No transitive dependencies; all attributes directly relate to the primary key within their own table. (user → DBUser, music_id → Music)

BCNF - SearchesForMusic is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (user_id, music_id).</td>
</tr>
</table>




```
create table MovieHas
(
   moviepeople_id int not null,
   movie_id int not null,

   PRIMARY KEY (moviepeople_id, movie_id),
   FOREIGN KEY (moviepeople_id) REFERENCES
Movie_People(moviepeople_id),
   FOREIGN KEY (movie_id) REFERENCES
Movie(movie_id) );
```



Functional Dependency
moviepeople_id → movie_id
movie_id → moviepeople_id

1NF - There are no repeating groups and all values are atomic.

2NF - Non prime attributes are dependent on the composite primary key.

3NF - No transitive dependencies; all attributes directly relate to the primary key within their own table. (moviepeople_id → Movie_People, movie_id → Movie)

BCNF- MovieHas is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (moviepeople_id, movie_id).

```
create table MovieIs
(
   genre_id int not null,
   movie_id int not null,

   PRIMARY KEY (genre_id, movie_id),
   FOREIGN KEY (genre_id) REFERENCES
Genre(genre_id),
   FOREIGN KEY (movie_id) REFERENCES
Movie(movie_id) );
```

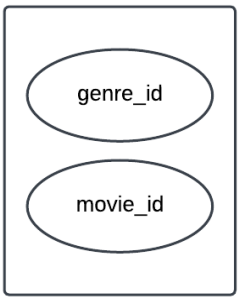
Functional Dependency
genre_id → movie_id
movie_id → genre_id

1NF - There are no repeating groups and all values are atomic.

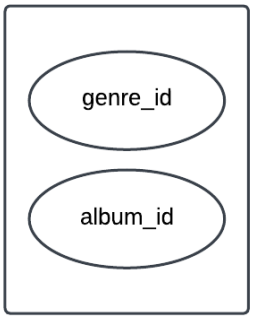
2NF - Non prime attributes are dependent on the composite primary key.

3NF - No transitive dependencies; all attributes directly relate to the primary key within their own table. (movie_id → Movie, genre_id → Genre)

BCNF- MovieIs is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (genre_id, movie_id).

<table>
<tr>
<td> 
</td>
<td></td>
</tr>
<tr>
<td>create table GenrePartOf
(
genre_id int not null,
album_id int not null,

PRIMARY KEY (genre_id, album_id),
FOREIGN KEY (genre_id) REFERENCES Genre(genre_id),
FOREIGN KEY (album_id) REFERENCES Album(album_id));

 
</td>
<td>Functional Dependency
genre_id → album_id
album_id → genre_id

1NF - There are no repeating groups and all values are atomic.

2NF - Non prime attributes are dependent on the composite primary key.

3NF - No transitive dependencies; all attributes directly relate to the primary key within their own table. (genre_id → Genre, album_id → Album)

BCNF - GenrePartOf is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (genre_id, album_id).</td>
</tr>
<tr>
<td>create table MusicHas
(
music_id int not null,
musicpeople_id int not null,

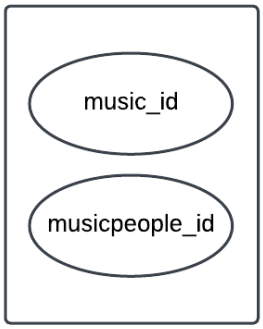
PRIMARY KEY (music_id, musicpeople_id),
FOREIGN KEY (music_id) REFERENCES Music(music_id),
FOREIGN KEY (musicpeople_id) REFERENCES Music_People(musicpeople_id));</td>
<td>Functional Dependency
music_id → musicpeople_id
musicpeople_id → music_id

1NF - There are no repeating groups and all values are atomic.

2NF - Non prime attributes are dependent on the composite primary key.

3NF - No transitive dependencies; all attributes directly relate to the primary key within their own table. (music_id → Music, musicpeople_id →)

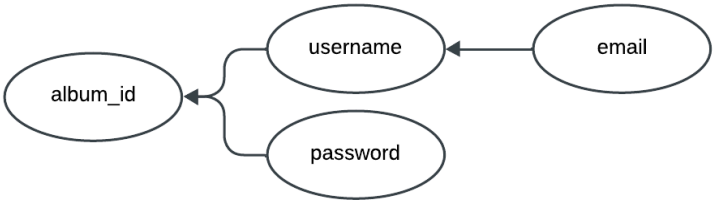
BCNF - MusicHas is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (music_id, musicpeople_id).</td>
</tr>
</table>



6.1 Synthesis Algorithm for 3NF

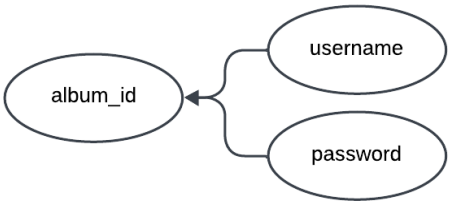
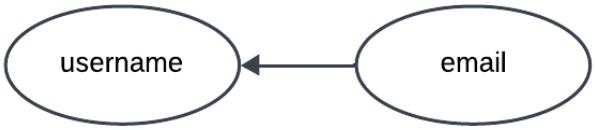
Using synthesis algorithm, the following table was changed to NOT be in 3NF:

```
CREATE TABLE DBAdmin (
   admin_id int not null,
   username varchar(50) not
null,
   password varchar(50) not
null,
   email varchar(100) not null,
   PRIMARY KEY(admin_id)
);
```



The table is not in 3NF because email is functionally dependent on “username” and not the primary key
In order to make this into a 3NF table, we will use the synthesis algorithm.

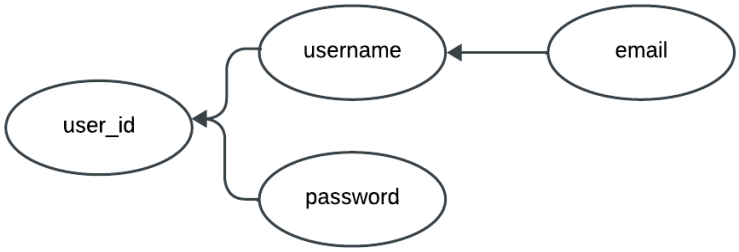
Decompose the table into 2 tables to eliminate functional dependencies:

DBAdmin1 Table	DBAdmin2 Table
Functional Dependency admin_id → username, password CREATE TABLE DBAdmin1 (admin_id int not null, username varchar(50) not null, password varchar(50) not null, PRIMARY KEY(admin_id)); 	**Functional Dependency** username → email CREATE TABLE DBAdmin2 (username varchar(50) not null, email varchar(100) not null, PRIMARY KEY(username)); 

6.2 Decomposition Algorithm for BCNF

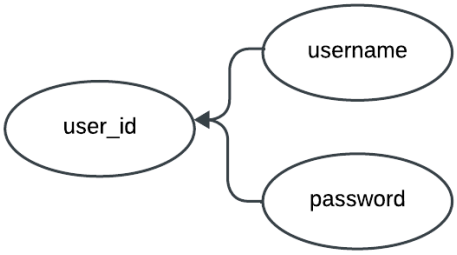
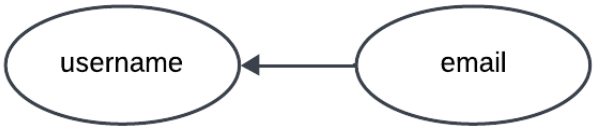
Using decomposition algorithm, the following table was changed to NOT be in BCNF:

```
CREATE TABLE DBUser
(
   user_id int not null,
   username varchar(50) not
null ,
   password varchar(50) not
null,

   PRIMARY KEY (user_id) );
```



The table is not in BCNF because email is functionally dependent on “username” and not the primary key In order to make this into a BCNF table, we will use the Decomposition algorithm.

Decompose the table into 2 tables to eliminate functional dependencies:

DBUser1 Table	DBUser2 Table
Functional Dependency user_id → username, password CREATE TABLE DBUser1 (user_id int not null, username varchar(50) not null, password varchar(50) not null, PRIMARY KEY(admin_id));  	**Functional Dependency** username → email CREATE TABLE DBUser2 (username varchar(50) not null, email varchar(100) not null, PRIMARY KEY(username)); 

7. Demo

7.1 Accessing the Interface

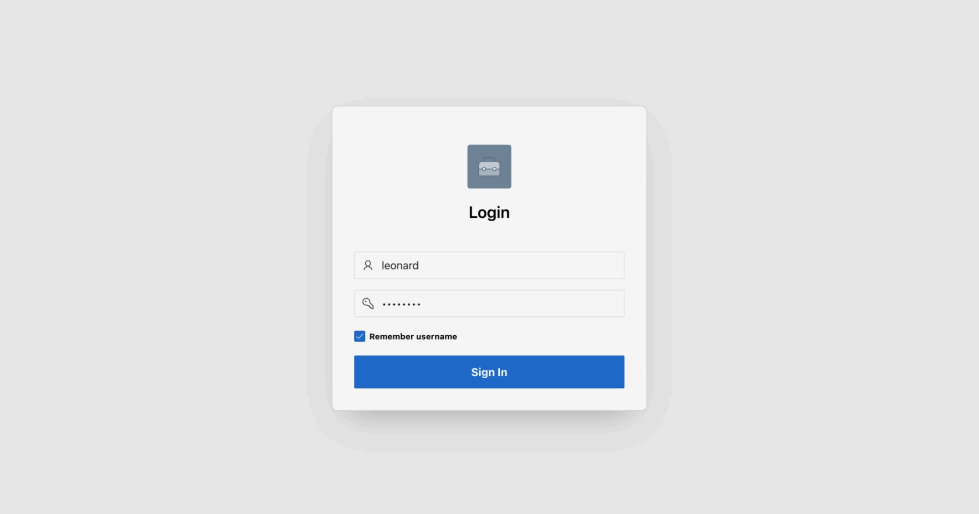
The movie and music database was made using Oracle Apex. The following is a link to the application:
https://apex.oracle.com/pls/apex/r/cps510/test10878/login?session=7543515860665

Admin Login	User Login
Username: admin Password: admin1	Username: leonard Password: password

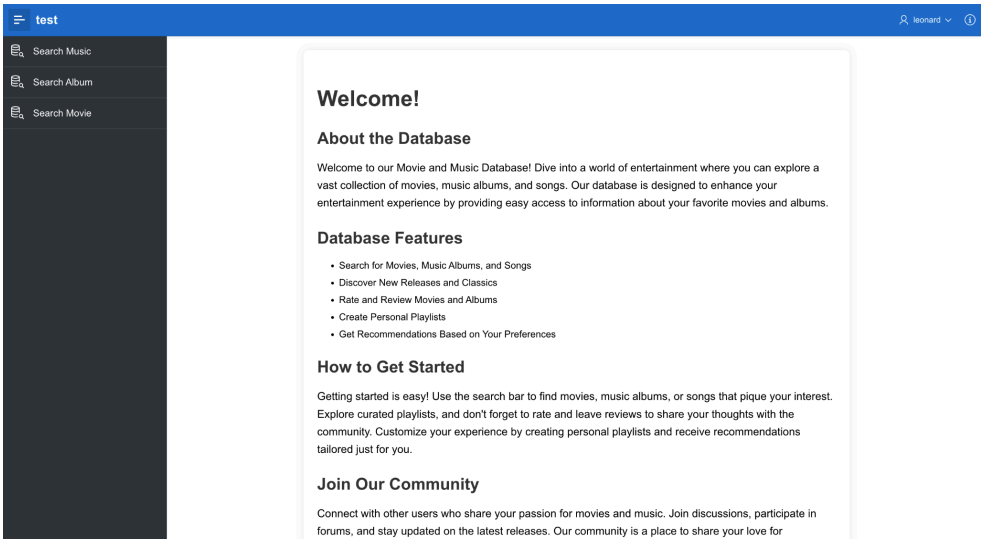
Below is a link to access the demo of our application
https://drive.google.com/file/d/1d_AZe1yfUjPzhe7v0O1qlwhPig6i2OkQ/view?usp=drive_link

7.2 Interface

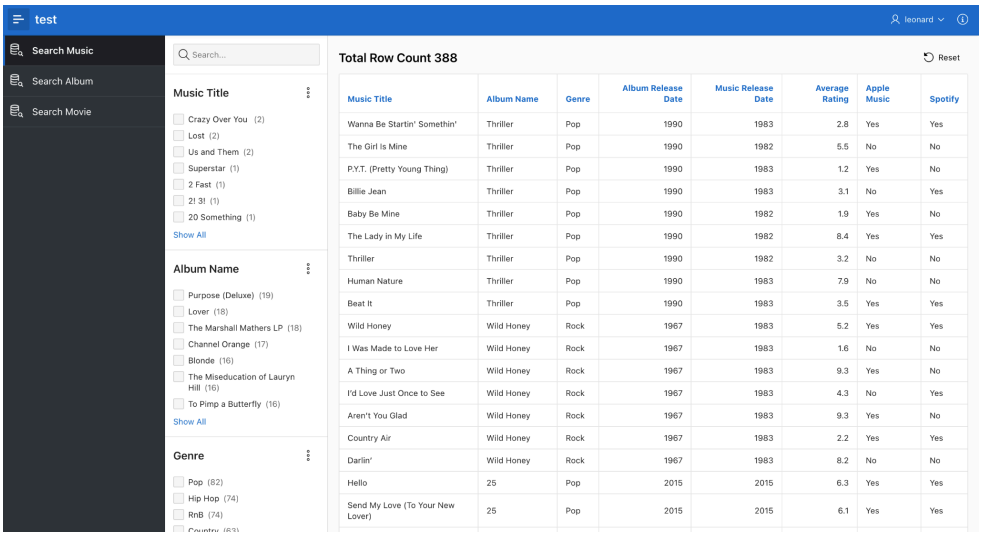
User-Side Interface

Page name	Interface
1. Login	

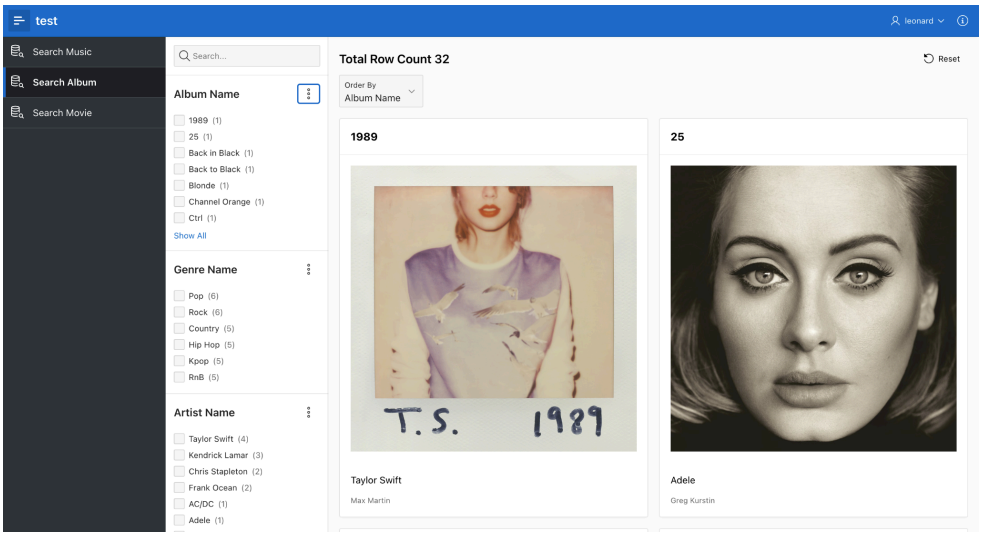
2. Homepage



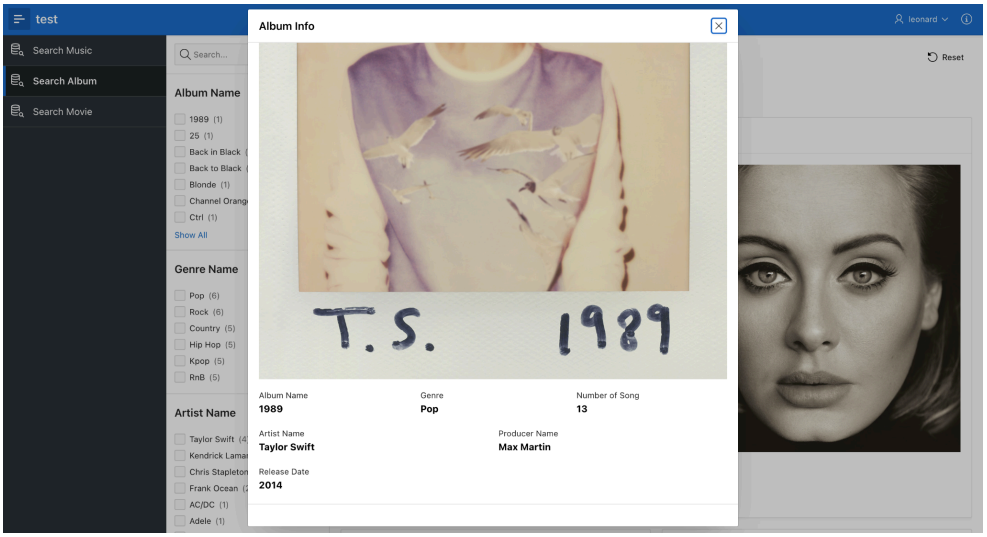
3. Search Music

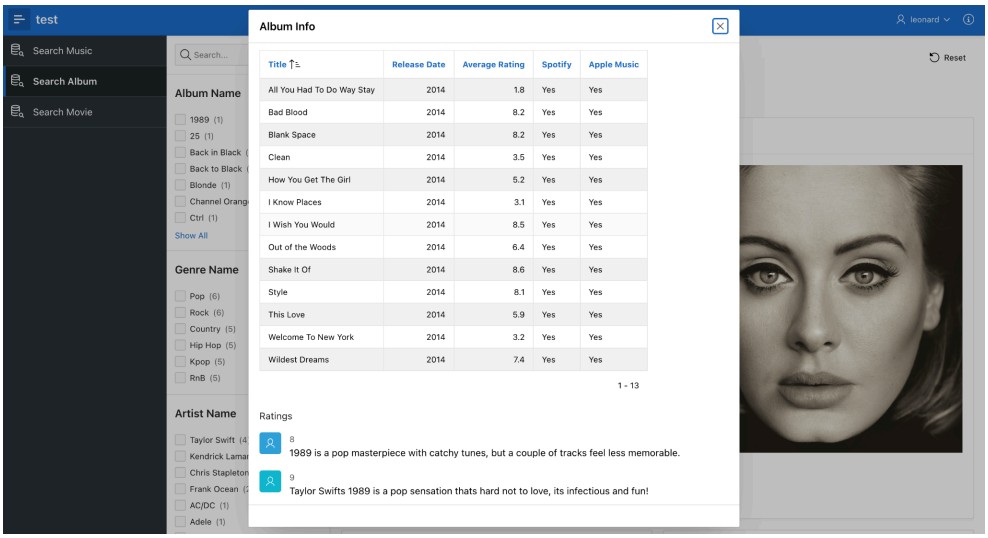


4. Search Album

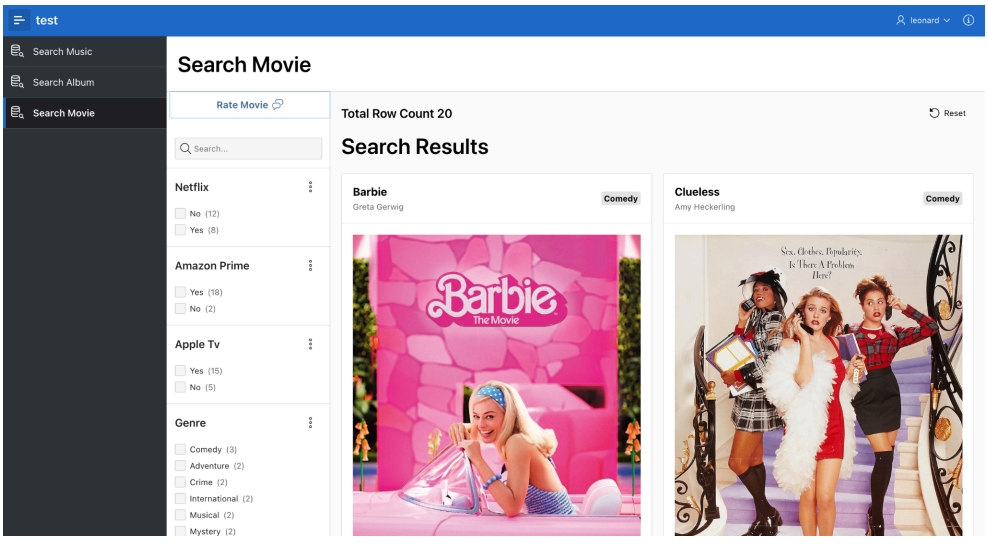


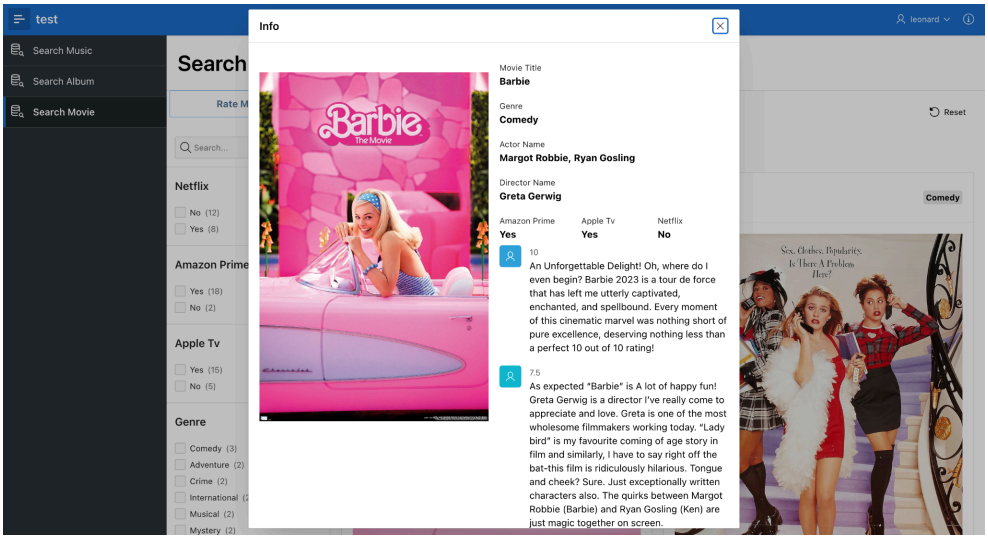
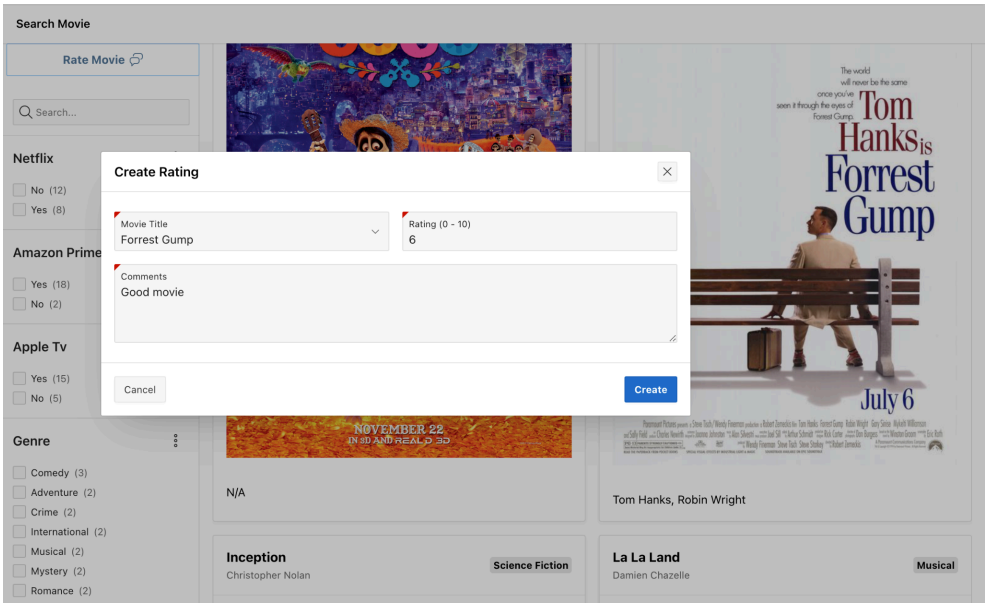
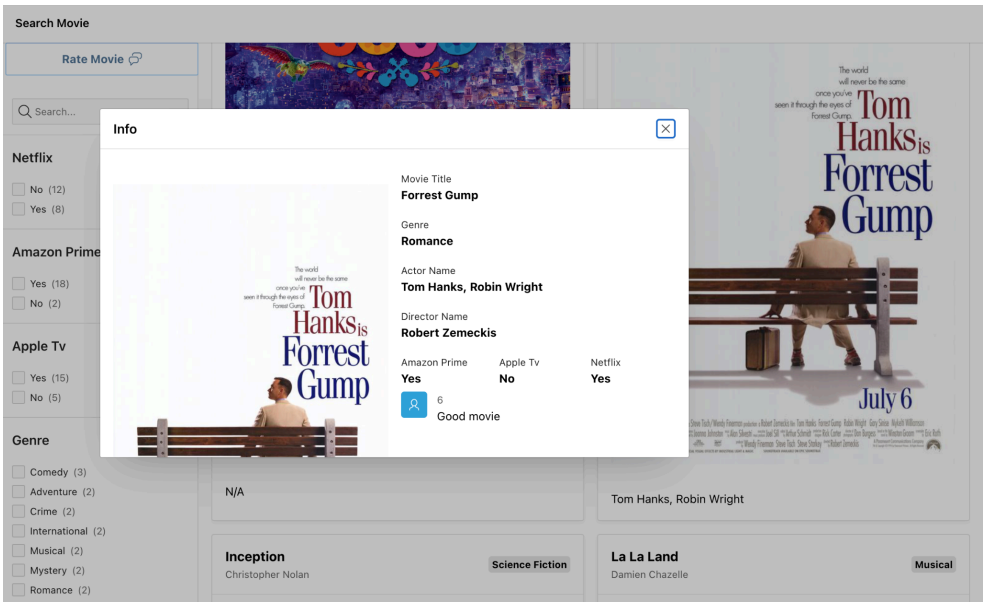
5. Album Information



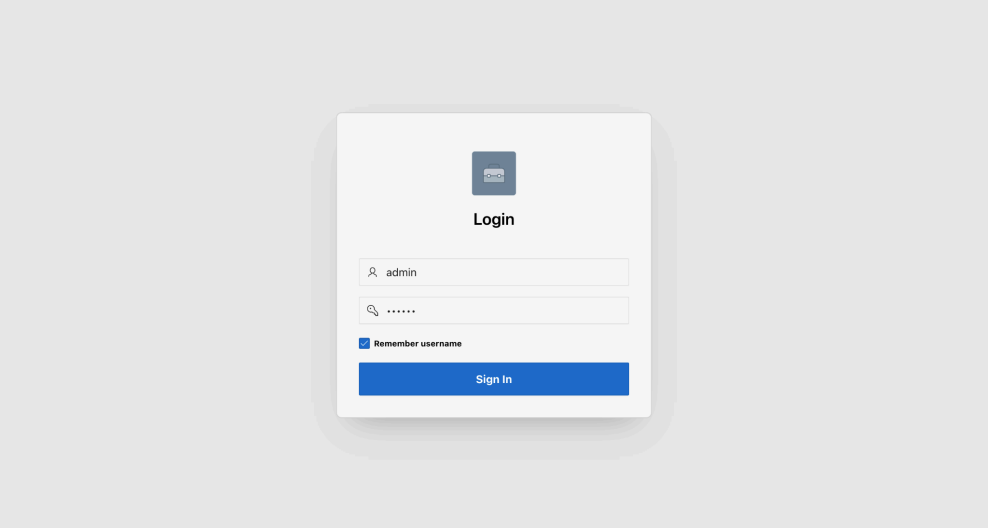
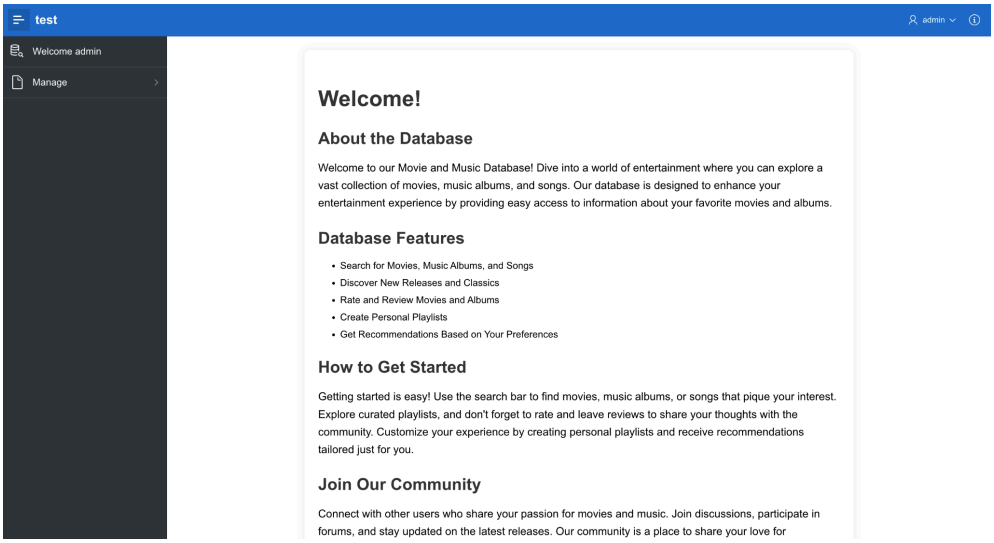
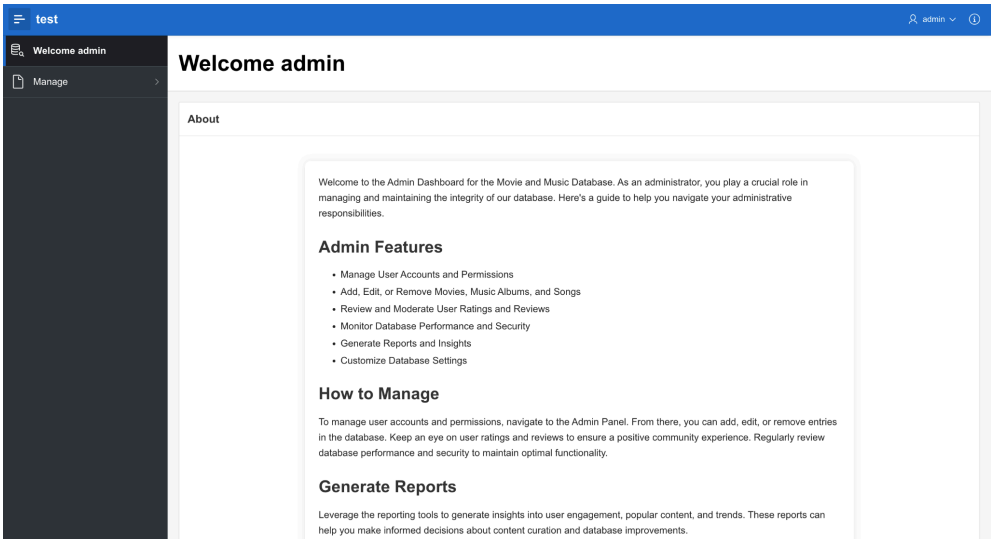


6. Search Movie

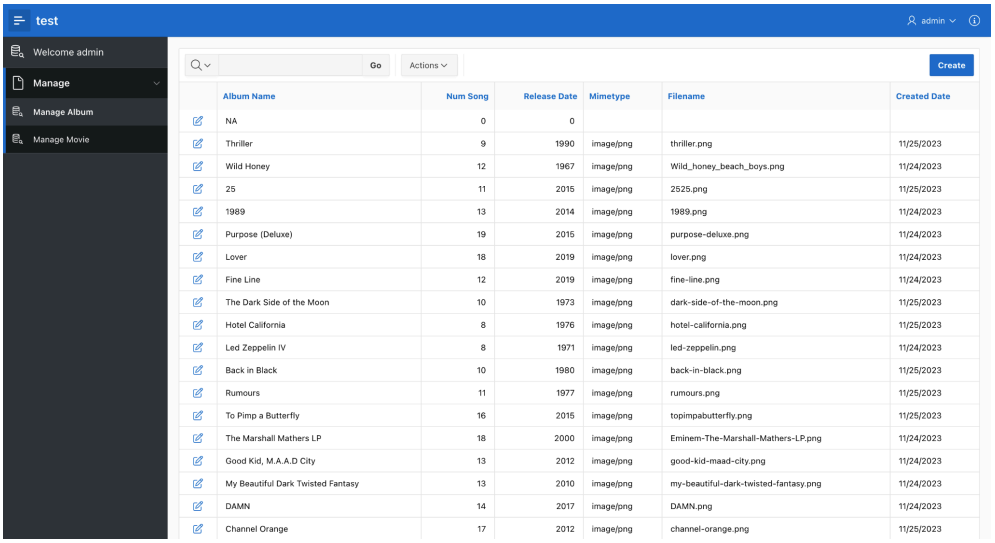


7. Movie Information	
8. Add Rating	 

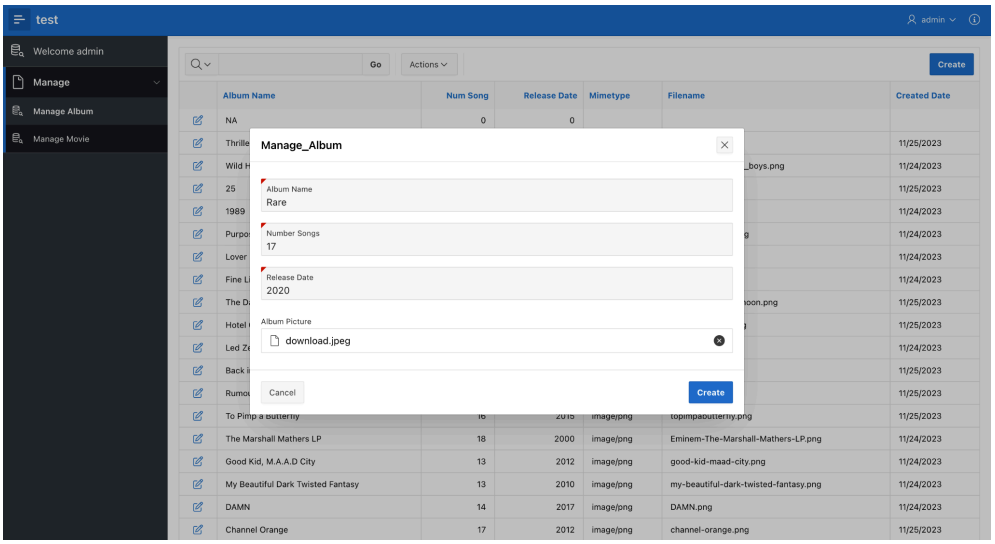
Admin-Side Interface

Page name	Interface
1. Login	
2. Homepage	
3. Admin Instructions	

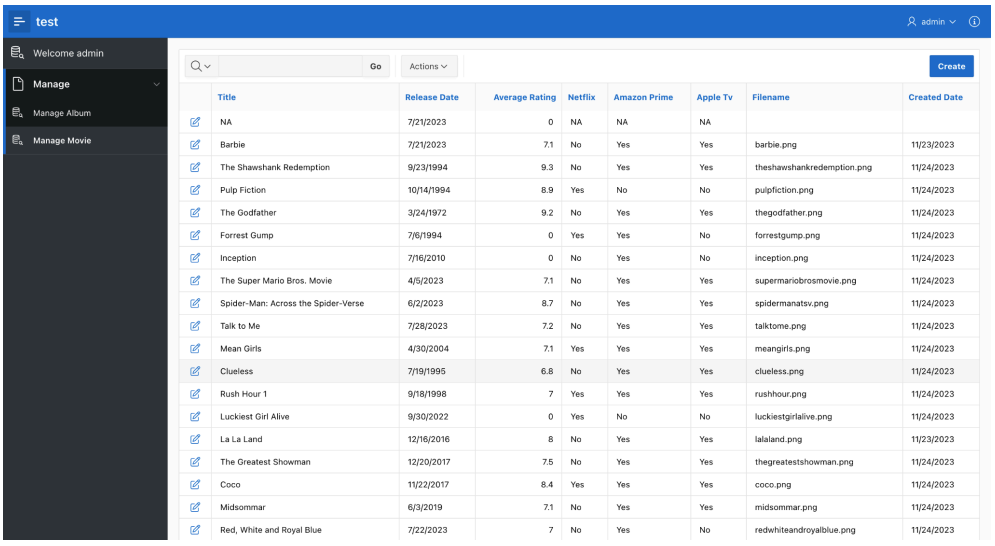
4. Manage Albums

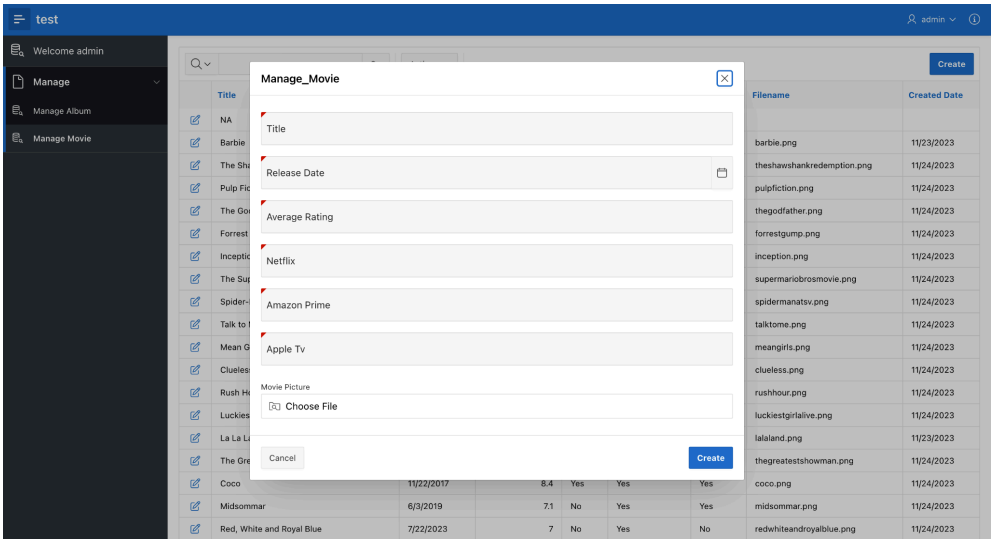


5. Add Albums



6. Manage Movie



7. Add Movie	

8. Relationship Algebra

Query 1:

Output the list of movie IDs and movie titles that have no ratings (this searches the Movie and Ratings tables to see if movie_id is not present in the Rating table).

```
SELECT DISTINCT m.movie_id, m.title
FROM Movie m
LEFT JOIN Rating r ON m.movie_id = r.movie_id
WHERE r.movie_id IS NULL;
```

Relationship Algebra:

$$\pi_{movie\_id,\, title}(\sigma_{movie.movie\_id=rating.movie\_id}(Movie \bowtie_{movie.movie\_id = rating.movie\_id} Rating))$$

Query 2:

Outputting a list of movie IDs and titles that have the genre, Comedy (Genre ID: 8) and were released between 2000-2005. (This searches the Movie, Genre and MovieIs table to see which Movie is of the genre Comedy).

```
SELECT m.movie_id, m.title
FROM Movie m
INNER JOIN MovieIs mi ON m.movie_id = mi.movie_id
WHERE mi.genre_id = 8
AND EXTRACT(YEAR FROM m.release_date) BETWEEN 2000 AND 2005;
```

Relationship Algebra:

$$\pi_{movie.movie\_id,\, movie.title}(\sigma_{movieIs.genre\_id = 8\ AND\ EXTRACT(YEAR\ FROM\ movie.release\_date)\ BETWEEN\ 2000\ AND\ 2005}(Movie \bowtie_{movie.movie\_id = movieIs.movie\_id} MovieIs))$$

Query 3:

Outputting a list of movies that are not available on Apple TV, but are available on Netflix and Amazon Prime (This checks the movie table to see which movies are available on which platform).

```
SELECT movie_id, title
FROM Movie
WHERE apple_tv = 'No' AND amazon_prime = 'Yes' AND netflix = 'Yes';
```

Relationship Algebra:

$$\pi_{movie.movie\_id,\, movie.title}(\sigma_{movie.apple\_tv = 'No'\ AND\ movie.amazon\_prime = 'Yes'\ AND\ movie.netflix = 'Yes'}(Movie))$$

Query 4:

Song titles which were released before 2010 and have a title longer than 10 characters and are available on spotify. (Checks Music columns release date, title, and spotify availability).

```
SELECT title
FROM music
WHERE release_date <= 2010
AND LENGTH(title) >10
AND spotify = 'No';
```

Relationship Algebra:

$$\pi_{music.title}(\sigma_{music.release\_date \leq 2010\ AND\ LENGTH(music.title) > 10\ \ AND\ \ music.spotify = 'No'}(Music))$$

Query 5:

Albums that were released between 2000 and 2010. (This checks the Album table to see the release years).

```
SELECT album_name
FROM Album
WHERE release_date >= 2000
   AND release_date <= 2010;
```

Relationship Algebra:

$$\pi_{Album.album\_name}(\sigma_{Album.release\_date \geq 2000\ AND\ \ Album.release\_date \leq 2010}(Album))$$

Query 6:

Give a list of animated movies. (This will search the MovieHas and MoviePeople tables for actor_name = N/A, as if there is no actor, the movie is animated in the database).

```
SELECT DISTINCT movie_id, title
FROM Movie
WHERE movie_id IN (
  SELECT movie_id
  FROM MovieHas mh
  JOIN Movie_People mp ON mh.moviepeople_id = mp.moviepeople_id
  WHERE mp.actor_name = 'N/A'
```

```
);
```

Relationship Algebra:

$$\pi_{movie\_id,\, title}(\sigma_{movie\_id \in \pi_{movie\_id}(\sigma_{actor\_name='N/A'}(MoviePeople \bowtie_{moviepeople\_id=moviepeople\_id} MovieHas))}(Movie))$$

Query 7:

Lists the name of the albums by Taylor Swift. (This looks through the Album and Music_People tables to find which album is by Taylor Swift).

```
SELECT DISTINCT  album_id, album_name
FROM album
WHERE album_id in (
   SELECT album_id
   FROM music_people
   WHERE artist_name = 'Taylor Swift'
);
```

Relationship Algebra:

$$\pi_{album\_id,\, album\_name}(\sigma_{album\_id \in \pi_{album\_id}(\sigma_{artist\_name='TaylorSwift'}(MusicPeople))}(Album))$$

Query 8:

Who are the users that have searched for both movies and music. (This looks in the DBUser, SearchesForMovie and SearchesForMusic tables to see which user_id is present in both).

```
SELECT DISTINCT u.user_id, u.username
FROM DBUser u
WHERE u.user_id IN (
   SELECT sm.user_id
   FROM SearchesForMovie sm
   INTERSECT
   SELECT sfm.user_id
   FROM SearchesForMusic sfm
);
```

Relationship Algebra:

$$\pi_{user\_id,\, username}(\sigma_{user\_id \in (\pi_{user\_id}(SearchesForMovie) \cap \pi_{user\_id}(SearchesForMusic))}(DBUser))$$