

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

<b>1. Introduction</b>	<b>4</b>
1.1 Description & Results Expected	4
1.2 Relationship between Entities	4
1.3 Entities, Attributes & Data Type	4
1.3.1 Admin Entity	4
1.3.2 User Entity	5
1.3.3 Movie Entity	5
1.3.4 Movie People Entity	5
1.3.5 Genre Entity	5
1.3.6 Album Entity	5
1.3.7 Music Entity	6
1.3.8 Music People Entity	6
1.3.7 Rating Entity	6
1.4 Potential Functions	6
<b>2. ER Diagram</b>	<b>8</b>
<b>3. Schema Design and Database Construction</b>	<b>9</b>
3.1 Color Key	9
3.2 Schema Design/Table Design	9
3.3 Database Construction	10
3.4 Schema Diagram	13
3.5 Populate Tables (Insert Statements)	13
<b>4. Designing Views/ Simple Quarries</b>	<b>14</b>
4.1 Simple Queries	14
4.1.1 Query 1:	14
4.1.2 Query 2:	14
4.1.3 Query 3:	15
4.1.4 Query 4:	15
4.1.5 Query 5:	16
4.1.6 Query 6:	16
4.1.7 Query 7:	17
4.1.8 Query 8:	17
4.2 View Designs	18
4.2.1 View 1:	18
4.2.2 View 2:	18
4.2.3 View 3:	19
4.2.4 View 4:	19
<b>5. Advanced Queries by Unix shell Implementation</b>	<b>20</b>
5.1 Advanced Queries	20
5.1.1 Query 1:	20
5.1.2 Query 2:	20
5.1.3 Query 3:	20
5.2 Interesting Queries with keywords	21
5.2.1 Query 1:	21

5.2.2 Query 2:	22
5.2.3 Query 3:	23
5.3 Unix Shell Code	23
<b>6. Normalization and Functional Dependencies</b>	<b>24</b>
6.1 Synthesis Algorithm for 3NF	31
6.2 Decomposition Algorithm for BCNF	32
<b>7. Demo</b>	<b>33</b>
7.1 Accessing the Interface	33
7.2 Interface	33
<b>8. Relationship Algebra</b>	<b>39</b>
Query 1:	39
Query 2:	39
Query 3:	40
Query 4:	40
Query 5:	40
Query 6:	40
Query 7:	41
Query 8:	41

# 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

<i>Entity</i>	<i>Relationships</i>
Movie	<ul style="list-style-type: none"><li>• Movie has a rating (1 : N)</li><li>• Movie has movie People (N : M)</li><li>• Movie is part of a genre (N : M)</li></ul>
Music	<ul style="list-style-type: none"><li>• Music is part of an album (N : 1)</li><li>• Music has a genre (N : M)</li><li>• Music has music people (N : M)</li></ul>
User	<ul style="list-style-type: none"><li>• User Searches for a Movie (N : M)</li><li>• User Searches for Music (N : M)</li></ul>
Admin	<ul style="list-style-type: none"><li>• Admin can edit Music (N : M)</li><li>• Admin can edit Movie (N : M)</li></ul>
Music People	<ul style="list-style-type: none"><li>• Music People make an album (N : 1)</li></ul>

## 1.3 Entities, Attributes & Data Type

### 1.3.1 Admin Entity

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

### 1.3.2 User Entity

<i>Attributes</i>	<i>Data Type</i>
<u>User ID</u>	Integer
Username	String
Password	String

### 1.3.3 Movie Entity

<i>Attributes</i>	<i>Data type</i>
<u>Movie ID</u>	Integer
Title	String
Release Date	Date
Average Rating	Float
Netflix	String
Amazon Prime	String
Apple TV	String

### 1.3.4 Movie People Entity

<i>Attributes</i>	<i>Data Type</i>
<u>Movie People ID</u>	Integer
Director Name	String
Actor Name	String

### 1.3.5 Genre Entity

<i>Attributes</i>	<i>Data Type</i>
<u>Genre ID</u>	Integer
Name	String

### 1.3.6 Album Entity

<i>Attributes</i>	<i>Data Type</i>

<u>Album ID</u>	Integer
Album Name	String
Number of Songs	Integer
Release Date	Integer

#### 1.3.7 Music Entity

<i>Attributes</i>	<i>Data Type</i>
<u>Music ID</u>	Integer
Title	String
Release Date	Integer
Average Rating	Float
Spotify	String
Apple Music	String

#### 1.3.8 Music People Entity

<i>Attributes</i>	<i>Data Type</i>
<u>Music People ID</u>	Integer
Artist Name	String
Producer Name	String

#### 1.3.7 Rating Entity

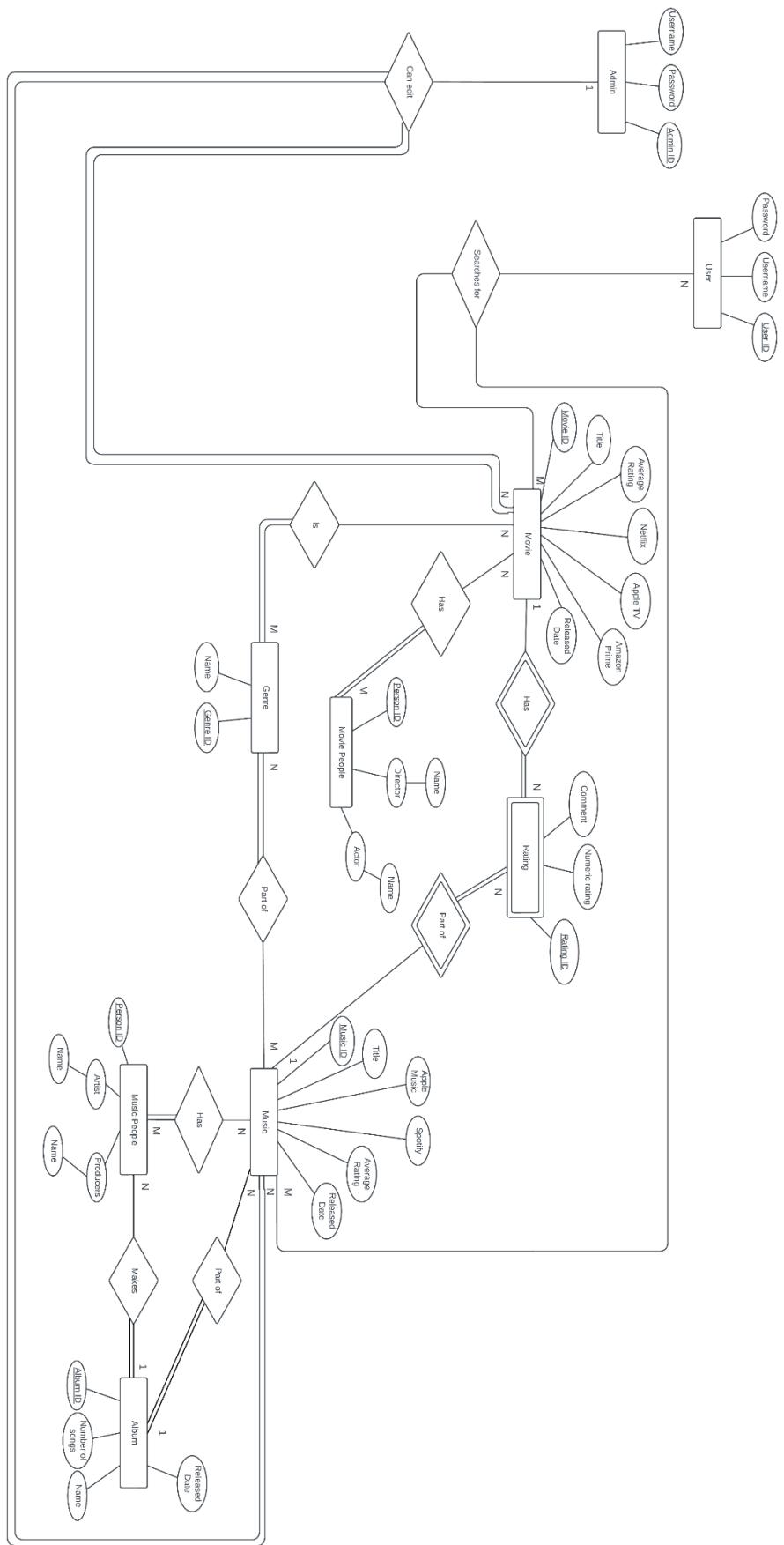
<i>Attributes</i>	<i>Data Type</i>
<u>Rating ID</u>	Integer
Numeric Rating	Float
Comments	String

## 1.4 Potential Functions

<u>Function</u>	<u>Description</u>

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



### 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,

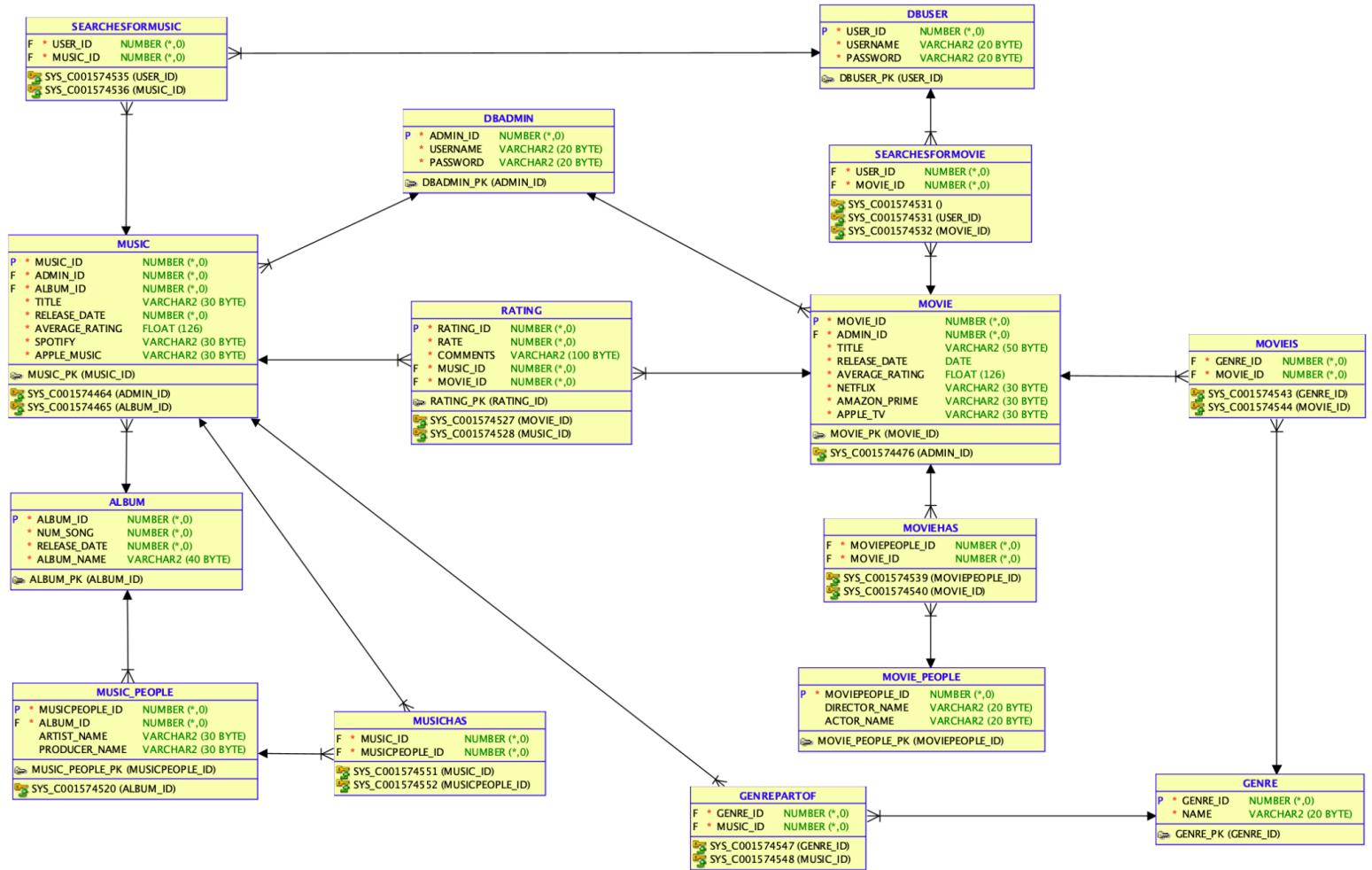
    PRIMARY KEY (genre_id, album_id),
    FOREIGN KEY (genre_id) REFERENCES Genre(genre_id),
    FOREIGN KEY (album_id) REFERENCES Album(album_id)
);

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](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;
```

The screenshot shows the MySQL Workbench interface. In the top pane, a code editor contains the SQL query for finding movies with no ratings. Below it, the 'Query Result' tab is selected, showing the output of the query. The results are presented in a table with columns 'MOVIE\_ID' and 'TITLE'. The data rows are:

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;
```

The screenshot shows the MySQL Workbench interface. In the top pane, a code editor contains the SQL query for finding comedy movies released between 2000 and 2005. Below it, the 'Query Result' tab is selected, showing the output of the query. The results are presented in a table with columns 'MOVIE\_ID' and 'TITLE'. The data row is:

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';
```

The screenshot shows the MySQL Workbench interface. In the top pane, a query is being typed into the SQL editor:

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

In the bottom pane, the "Query Result" tab is selected, showing the results of the executed query:

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';
```

The screenshot shows the MySQL Workbench interface. In the top pane, a query is being typed into the SQL editor:

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

In the bottom pane, the "Query Result" tab is selected, showing the results of the executed query:

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;
```

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

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

The results are displayed in the 'Query Result' tab, showing five rows of album names:

ALBUM_NAME
1 The Marshall Mathers LP
2 My Beautiful Dark Twisted Fantasy
3 Back to Black
4 Fearless
5 Speak Now

#### 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'
);
```

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
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'
);
```

The results are displayed in the 'Query Result' tab, showing three rows of movie IDs and titles:

MOVIE_ID	TITLE
1	7 The Super Mario Bros. Movie
2	8 Spider-Man: Across the Spider-Verse
3	16 Coco

#### 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'
);
```

The screenshot shows a database interface with a SQL editor and a results viewer. The SQL query is:

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

The results table has columns ALBUM\_ID and ALBUM\_NAME, containing the following data:

ALBUM_ID	ALBUM_NAME
1	4 1989
2	6 Lover
3	31 Fearless
4	32 Speak Now

#### 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
);
```

The screenshot shows a database interface with a SQL editor and a results viewer. The SQL query is:

```
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
);
```

The results table has columns USER\_ID and USERNAME, containing the following data:

USER_ID	USERNAME
1	leonard

## 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:

The screenshot shows the SQL Server Management Studio interface. In the top pane, there is a script editor window containing the SQL code for creating a view named 'actor\_names\_that\_start\_with\_a'. The code includes a SELECT statement to retrieve 'actor\_name' from the 'movie\_people' table where the name starts with 'A%', and a final SELECT \* statement to show all rows from the view. Below the script editor is a 'Query Result' grid. The title bar of the grid says 'SQL | All Rows Fetched: 2 in 0.062 seconds'. The grid has a single column labeled 'ACTOR\_NAME'. The data rows are: 1 Al Pacino, Marlon Brando and 2 Alicia Silverstone, Paul Rudd.

ACTOR_NAME
1 Al Pacino, Marlon Brando
2 Alicia Silverstone, Paul Rudd

### 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:

The screenshot shows the SQL Server Management Studio interface. In the top pane, there is a script editor window containing the SQL code for creating a view named 'above\_average'. The code includes a SELECT statement to retrieve 'rate' from the 'Rating' table where the rating is greater than 5, and a final SELECT \* statement to show all rows from the view. Below the script editor is a 'Query Result' grid. The title bar of the grid says 'SQL | All Rows Fetched: 85 in 0.122 seconds'. The grid has a single column labeled 'RATE'. The data rows are: 1 7.5, 2 10, 3 9.7, 4 8.6, 5 8.9, 6 9.5, 7 8.3, 8 9.3, 9 8.4, 10 7.2, and so on up to 11 8.7.

RATE
1 7.5
2 10
3 9.7
4 8.6
5 8.9
6 9.5
7 8.3
8 9.3
9 8.4
10 7.2
11 8.7

#### 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:

The screenshot shows the SQL Server Management Studio interface. At the top, there is a script pane containing the SQL code for creating a view and selecting from it. Below the script pane is a results pane titled "Query Result" which displays the output of the query. The output is a table with two columns: "TITLE" and "NETFLIX". The data in the table is as follows:

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:

The screenshot shows the SQL Server Management Studio interface. At the top, there is a script pane containing the SQL code for creating a view and selecting from it. Below the script pane is a results pane titled "Query Result" which displays the output of the query. The output is a table with two columns: "ALBUM\_NAME" and "RELEASE\_DATE". The data in the table is as follows:

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 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:*

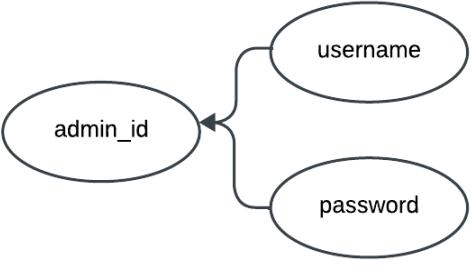
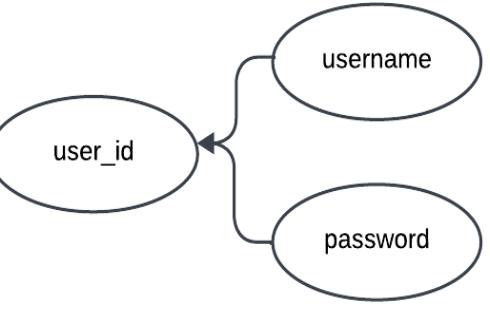
NETFLIX	AMAZON_PRIME	APPLE_TV	AVERAGE_RATING
No	Yes	No	4
Yes	Yes	Yes	8
Yes	No	No	4
NA	NA	NA	0
No	Yes	Yes	8
Yes	Yes	No	0

## 5.3 Unix Shell Code

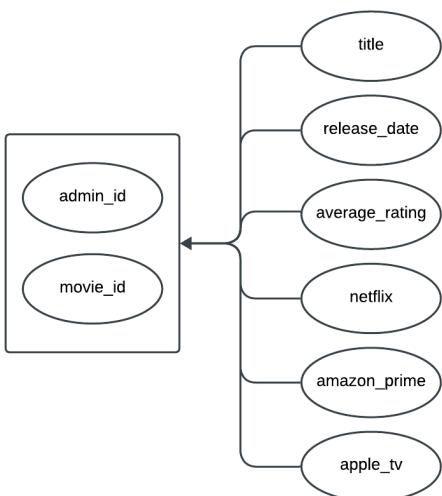
You can access the unix shell scripts through the link below -

[https://docs.google.com/document/d/10lGsl5fk-D0AjOCJp25nxIM\\_bNO-561fWslM63AovEw/edit#heading=h.uptv8xhoi0gg](https://docs.google.com/document/d/10lGsl5fk-D0AjOCJp25nxIM_bNO-561fWslM63AovEw/edit#heading=h.uptv8xhoi0gg)

## 6. Normalization and Functional Dependencies

Relational Table	Normalization
<pre>create table DBAdmin (     admin_id int not null,     username varchar(50) not null,     password varchar(50) not null,     PRIMARY KEY(admin_id) );</pre>  <pre>     graph LR         admin_id([admin_id]) --&gt; username([username])         admin_id --&gt; password([password])     </pre>	<p><b>Functional Dependency</b>  <math>\text{admin\_id} \rightarrow \text{username, password}</math></p> <p>1NF - There are no repeating groups and all values are atomic.</p> <p>2NF - Non prime attributes are dependent on the primary key (admin_id).</p> <p>3NF - No transitive dependencies; all attributes directly relate to the primary key.</p> <p>BCNF - DBAdmin is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (admin_id).</p>
<pre>create table DBUser (     user_id int not null,     username varchar(50) not null ,     password varchar(50) not null,     PRIMARY KEY (user_id) );</pre>  <pre>     graph LR         user_id([user_id]) --&gt; username([username])         user_id --&gt; password([password])     </pre>	<p><b>Functional Dependency</b>  <math>\text{user\_id} \rightarrow \text{username, password}</math></p> <p>1NF - There are no repeating groups and all values are atomic.</p> <p>2NF - Non prime attributes are dependent on the primary key (user_id).</p> <p>3NF - No transitive dependencies; all attributes directly relate to the primary key.</p> <p>BCNF - DBUser is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (user_id).</p>
<pre>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,</pre>	<p><b>Functional Dependency</b>  <math>\text{movie\_id} \rightarrow \text{title, release\_date, average\_rating, netflix, amazon\_prime, apple\_tv, admin\_id}</math></p> <p>1NF - There are no repeating groups and all values are atomic.</p> <p>2NF - Non prime attributes are dependent on the primary key (movie_id).</p>

PRIMARY KEY (movie\_id),  
 FOREIGN KEY (admin\_id) REFERENCES  
 DBAdmin(admin\_id);

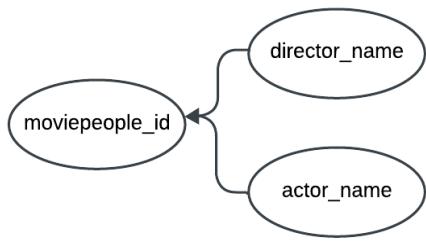


3NF - No transitive dependencies; all attributes directly relate to the primary key.

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**  
 $\text{moviepeople\_id} \rightarrow \text{director\_name, actor\_name}$

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

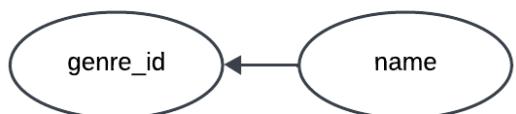
2NF - Non prime attributes are dependent on the primary key (moviepeople\_id).

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**  
 $\text{genre\_id} \rightarrow \text{name}$

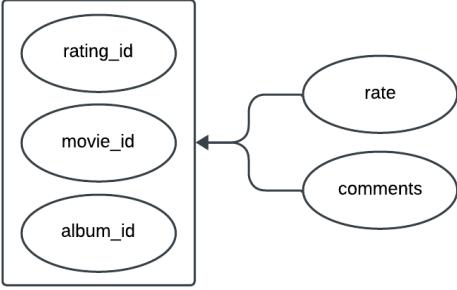
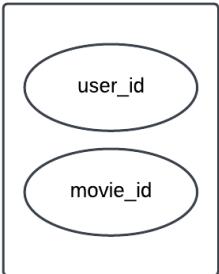
1NF - There are no repeating groups and all values are atomic.

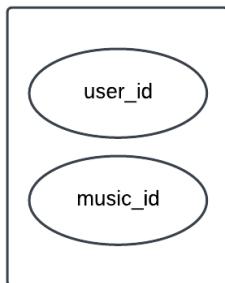
2NF - Non prime attributes are dependent on the primary key (genre\_id).

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

	<p>BCNF - Genre is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (user_id, movie_id).</p>
<pre>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 );</pre> <pre>     graph TD         album_id([album_id]) --&gt; album_name([album_name])         album_id --&gt; num_song([num_song])         album_id --&gt; release_date([release_date])     </pre>	<p><b>Functional Dependency</b>  <math>album\_id \rightarrow album, actor\_name, num\_song, release\_date</math></p> <p>1NF - There are no repeating groups and all values are atomic.</p> <p>2NF - Non prime attributes are dependent on the primary key (album_id).</p> <p>3NF - No transitive dependencies; all attributes directly relate to the primary key.</p> <p>BCNF - Album is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (album_id)</p>
<pre>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 );</pre>	<p><b>Functional Dependency</b>  <math>music\_id \rightarrow title, release\_date, average\_rating, spotify, apple\_music, admin\_id, album\_id</math></p> <p>1NF - There are no repeating groups and all values are atomic.</p> <p>2NF - Non prime attributes are dependent on the primary key (music_id).</p> <p>3NF - No transitive dependencies; all attributes directly relate to the primary key.</p> <p>BCNF - Music is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (music_id)</p>

<pre>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));</pre>	<p><b>Functional Dependency</b>  <math>\text{musicpeople\_id} \rightarrow \text{artist\_name, producers\_name}</math></p> <p>1NF - Contains atomic values: moviepeople_id, director_name, actor_name.</p> <p>2NF - No partial dependencies; all attributes depend on the primary key (moviepeople_id).</p> <p>3NF - No transitive dependencies; all attributes directly relate to the primary key.</p> <p>BCNF - Music_people is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (musicpeople_id)</p>	
<pre>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));</pre>	<p><b>Functional Dependency</b>  <math>\text{rating\_id} \rightarrow \text{rate, comments, movie\_id, album\_id}</math></p> <p>1NF: Contains atomic values: rating_id, rate, comments, movie_id, album_id.</p> <p>2NF: No partial dependencies; all attributes depend on the primary key (rating_id).</p> <p>3NF: No transitive dependencies; all attributes directly relate to the primary key.</p>	

	<p>BCNF - Rating is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (rating_id)</p>
<pre>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);</pre> 	<p><b>Functional Dependency</b>  <math>\text{user\_id} \rightarrow \text{movie\_id}</math>  <math>\text{movie\_id} \rightarrow \text{user\_id}</math></p> <p>1NF - There are no repeating groups and all values are atomic.</p> <p>2NF - Non prime attributes are dependent on the composite primary key.</p> <p>3NF - No transitive dependencies; all attributes directly relate to the primary key within their own table. (<math>\text{user} \rightarrow \text{DBUser}</math>, <math>\text{movie\_id} \rightarrow \text{Movie}</math>)</p> <p>BCNF - SearchesForMovie is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (user_id, movie_id).</p>
<pre>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);</pre>	<p><b>Functional Dependency</b>  <math>\text{user\_id} \rightarrow \text{music\_id}</math>  <math>\text{music\_id} \rightarrow \text{user\_id}</math></p> <p>1NF - There are no repeating groups and all values are atomic.</p> <p>2NF - Non prime attributes are dependent on the composite primary key.</p> <p>3NF - No transitive dependencies; all attributes directly relate to the primary key within their own table. (<math>\text{user} \rightarrow \text{DBUser}</math>, <math>\text{music\_id} \rightarrow \text{Music}</math>)</p> <p>BCNF - SearchesForMusic is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (user_id, music_id).</p>



```
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**  
 $\text{moviepeople\_id} \rightarrow \text{movie\_id}$   
 $\text{movie\_id} \rightarrow \text{moviepeople\_id}$

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. ( $\text{moviepeople\_id} \rightarrow \text{Movie\_People}$ ,  $\text{movie\_id} \rightarrow \text{Movie}$ )

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);
```

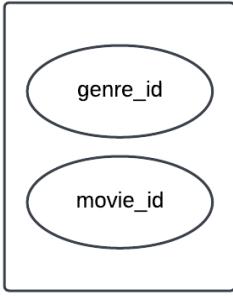
**Functional Dependency**  
 $\text{genre\_id} \rightarrow \text{movie\_id}$   
 $\text{movie\_id} \rightarrow \text{genre\_id}$

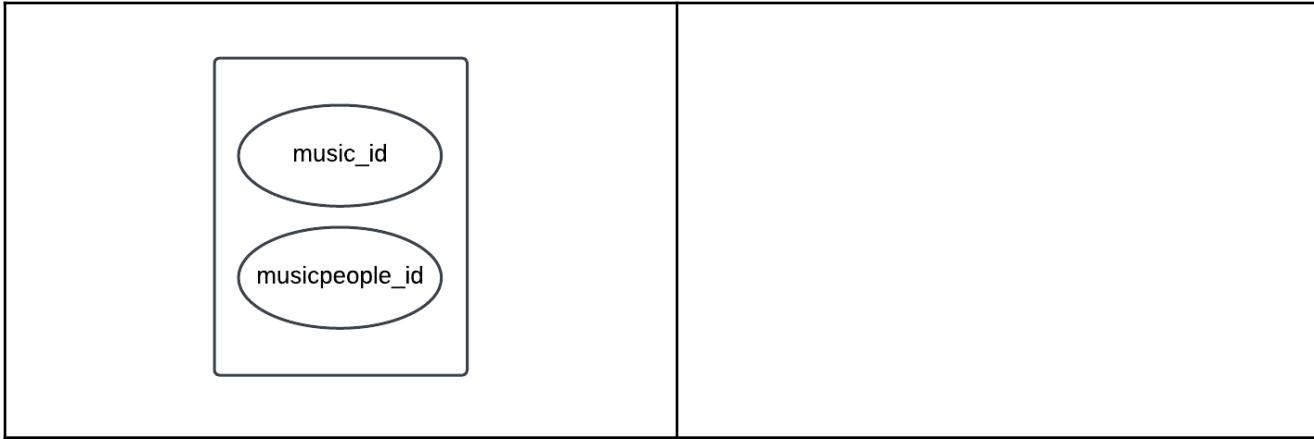
1NF - There are no repeating groups and all values are atomic.

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. ( $\text{movie\_id} \rightarrow \text{Movie}$ ,  $\text{genre\_id} \rightarrow \text{Genre}$ )

BCNF- MovieIs is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (genre\_id, movie\_id).

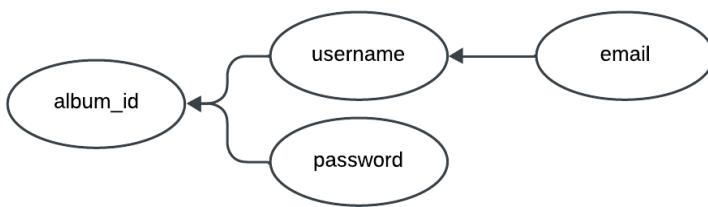
		
<pre>create table GenrePartOf (     genre_id int not null,     album_id int not null,     PRIMARY KEY (genre_id, album_id),     FOREIGN KEY (genre_id) REFERENCES     Genre(genre_id),     FOREIGN KEY (album_id) REFERENCES     Album(album_id);</pre>	<p><b>Functional Dependency</b>  <math>\text{genre\_id} \rightarrow \text{album\_id}</math>  <math>\text{album\_id} \rightarrow \text{genre\_id}</math></p> <p>1NF - There are no repeating groups and all values are atomic.</p> <p>2NF - Non prime attributes are dependent on the composite primary key.</p> <p>3NF - No transitive dependencies; all attributes directly relate to the primary key within their own table. (<math>\text{genre\_id} \rightarrow \text{Genre}</math>, <math>\text{album\_id} \rightarrow \text{Album}</math>)</p> <p>BCNF - <code>GenrePartOf</code> is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (<math>\text{genre\_id}, \text{album\_id}</math>).</p>	
<pre>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);</pre>	<p><b>Functional Dependency</b>  <math>\text{music\_id} \rightarrow \text{musicpeople\_id}</math>  <math>\text{musicpeople\_id} \rightarrow \text{music\_id}</math></p> <p>1NF - There are no repeating groups and all values are atomic.</p> <p>2NF - Non prime attributes are dependent on the composite primary key.</p> <p>3NF - No transitive dependencies; all attributes directly relate to the primary key within their own table. (<math>\text{music\_id} \rightarrow \text{Music}</math>, <math>\text{musicpeople\_id} \rightarrow \text{Music\_People}</math>)</p> <p>BCNF - <code>MusicHas</code> is in BCNF because all functional dependencies within the table are satisfied based on the composite primary key (<math>\text{music\_id}, \text{musicpeople\_id}</math>).</p>	



## 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.

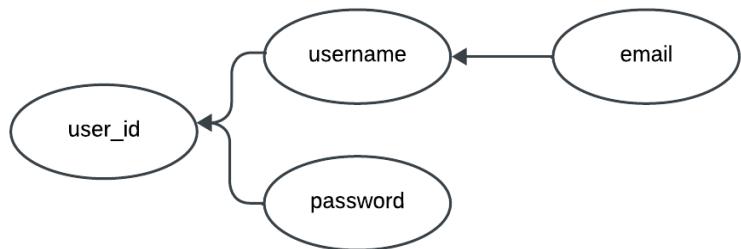
Decompose the table into 2 tables to eliminate functional dependencies:

DBAdmin1 Table	DBAdmin2 Table
<p><b>Functional Dependency</b>  <math>\text{admin\_id} \rightarrow \text{username, password}</math></p> <pre>CREATE TABLE DBAdmin1 (     admin_id int not null,     username varchar(50) not null,     password varchar(50) not null,     PRIMARY KEY(admin_id) );</pre>	<p><b>Functional Dependency</b>  <math>\text{username} \rightarrow \text{email}</math></p> <pre>CREATE TABLE DBAdmin2 (     username varchar(50) not null,     email varchar(100) not null,     PRIMARY KEY(username) );</pre>

## 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
<p><b>Functional Dependency</b>  <math>\text{user\_id} \rightarrow \text{username, password}</math></p> <pre>CREATE TABLE DBUser1 (     user_id int not null,     username varchar(50) not null,     password varchar(50) not null,     PRIMARY KEY(admin_id) );</pre> <pre> graph LR     user_id1((user_id)) --&gt; username1((username))     user_id1 --&gt; password1((password))   </pre>	<p><b>Functional Dependency</b>  <math>\text{username} \rightarrow \text{email}</math></p> <pre>CREATE TABLE DBUser2 (     username varchar(50) not null,     email varchar(100) not null,     PRIMARY KEY(username) );</pre> <pre> graph LR     email2((email)) --&gt; username2((username))     email2 --&gt; user_id2((user_id))   </pre>

## 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>

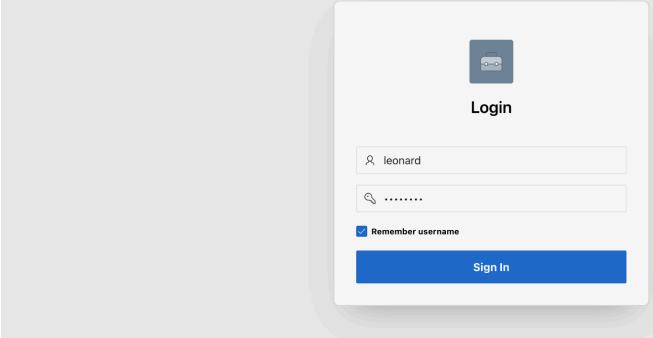
Admin Login	User Login
Username: admin Password: admin1	Username: leonard Password: password

Below is a link to access the demo of our application

[https://drive.google.com/file/d/1d\\_AZe1yfUjPzhe7v0O1qlwhPig6i2OkQ/view?usp=drive\\_link](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

**Welcome!**

**About the Database**

Welcome to our Movie and Music Database! Dive into a world of entertainment where you can explore a vast collection of movies, music albums, and songs. Our database is designed to enhance your entertainment experience by providing easy access to information about your favorite movies and albums.

**Database Features**

- Search for Movies, Music Albums, and Songs
- Discover New Releases and Classics
- Rate and Review Movies and Albums
- Create Personal Playlists
- Get Recommendations Based on Your Preferences

**How to Get Started**

Getting started is easy! Use the search bar to find movies, music albums, or songs that pique your interest. Explore curated playlists, and don't forget to rate and leave reviews to share your thoughts with the community. Customize your experience by creating personal playlists and receive recommendations tailored just for you.

**Join Our Community**

Connect with other users who share your passion for movies and music. Join discussions, participate in forums, and stay updated on the latest releases. Our community is a place to share your love for

## 3. Search Music

**Total Row Count 388**

Music Title	Album Name	Genre	Album Release Date	Music Release Date	Average Rating	Apple Music	Spotify
Wanna Be Startin' Somethin'	Thriller	Pop	1990	1983	2.8	Yes	Yes
The Girl Is Mine	Thriller	Pop	1990	1982	5.5	No	No
P.Y.T. (Pretty Young Thing)	Thriller	Pop	1990	1983	1.2	Yes	No
Billie Jean	Thriller	Pop	1990	1983	3.1	No	Yes
Baby Be Mine	Thriller	Pop	1990	1982	1.9	Yes	No
The Lady In My Life	Thriller	Pop	1990	1982	8.4	Yes	Yes
Human Nature	Thriller	Pop	1990	1983	7.9	No	No
Beat It	Thriller	Pop	1990	1983	3.5	Yes	Yes
Wild Honey	Wild Honey	Rock	1967	1983	5.2	Yes	Yes
I Was Made to Love Her	Wild Honey	Rock	1967	1983	1.6	No	No
A Thing or Two	Wild Honey	Rock	1967	1983	9.3	Yes	No
I'd Love Just Once to See	Wild Honey	Rock	1967	1983	4.3	No	Yes
Aren't You Glad	Wild Honey	Rock	1967	1983	9.3	Yes	No
Country Air	Wild Honey	Rock	1967	1983	2.2	Yes	Yes
Darlin'	Wild Honey	Rock	1967	1983	8.2	No	No
Hello	25	Pop	2015	2015	6.3	Yes	Yes
Send My Love (To Your New Lover)	25	Pop	2015	2015	6.1	Yes	Yes

## 4. Search Album

**Total Row Count 32**

Album Name
1989 (1)
25 (1)
Back in Black (1)
Back to Black (1)
Blonde (1)
Channel Orange (1)
Ctrl (1)

**Artist Name**

Artist Name
Taylor Swift (4)
Kendrick Lamar (3)
Chris Stapleton (2)
Frank Ocean (2)
AC/DC (1)
Adele (1)



1989

T.S. 1989

Taylor Swift  
Max Martin



25

Adele  
Greg Kurstin

## 5. Album Information

**Album Info**



**1989**

**Taylor Swift**

**2014**

Album Name	1989	Genre	Pop	Number of Song	13
Artist Name	Taylor Swift	Producer Name	Max Martin		
Release Date	2014				

**Album Info**

Title	Release Date	Average Rating	Spotify	Apple Music
All You Had To Do Was Stay	2014	1.8	Yes	Yes
Bad Blood	2014	8.2	Yes	Yes
Blank Space	2014	8.2	Yes	Yes
Clean	2014	3.5	Yes	Yes
How You Get The Girl	2014	5.2	Yes	Yes
I Know Places	2014	3.1	Yes	Yes
I Wish You Would	2014	8.5	Yes	Yes
Out of the Woods	2014	6.4	Yes	Yes
Shake It Off	2014	8.6	Yes	Yes
Style	2014	8.1	Yes	Yes
This Love	2014	5.9	Yes	Yes
Welcome To New York	2014	3.2	Yes	Yes
Wildest Dreams	2014	7.4	Yes	Yes

1 - 13

**Ratings**

8 1989 is a pop masterpiece with catchy tunes, but a couple of tracks feel less memorable.

9 Taylor Swift's 1989 is a pop sensation that's hard not to love, it's infectious and fun!

## 6. Search Movie

**Search Movie**

**Rate Movie**

**Total Row Count 20**

**Search Results**

<b>Barbie</b> Greta Gerwig	<b>Comedy</b>
	
<b>Clueless</b> Amy Heckerling	<b>Comedy</b>
	

## 7. Movie Information

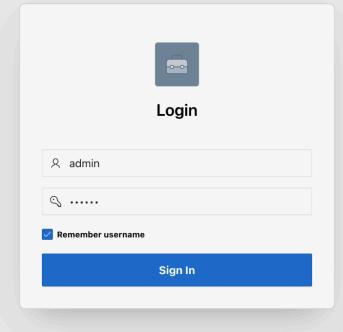
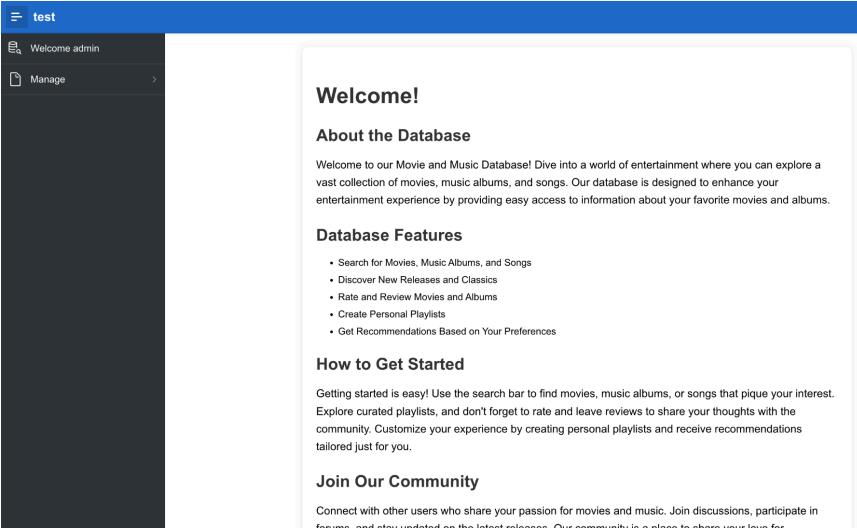
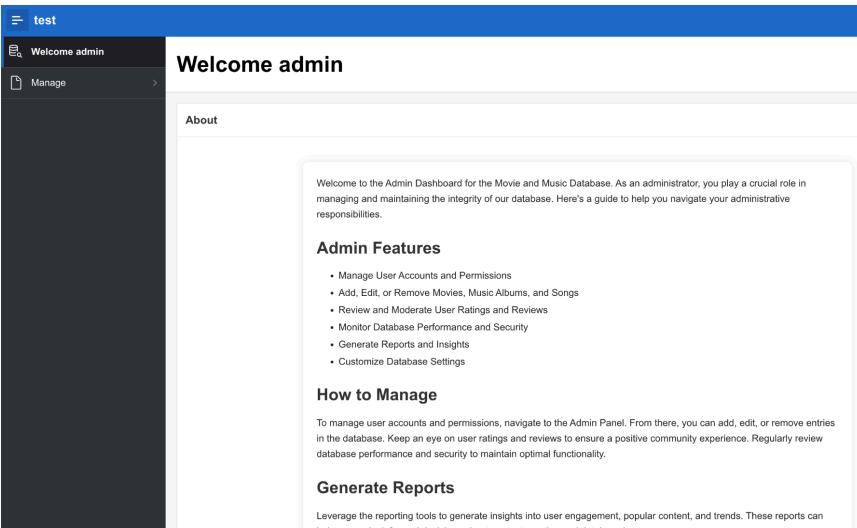
The screenshot shows a movie information page for 'Barbie'. At the top, there's a search bar with the query 'leonard'. Below it, a sidebar lists filters for 'Search Music', 'Search Album', and 'Search Movie'. Under 'Search Movie', there are filters for 'Rate Movie' (Yes/No), 'Netflix' (Yes/No), 'Amazon Prime' (Yes/No), 'Apple TV' (Yes/No), and 'Genre' (Comedy, Adventure, Crime, International, Musical, Mystery). The main content area displays the movie 'Barbie' with its title, genre (Comedy), actors (Margot Robbie, Ryan Gosling), director (Greta Gerwig), and availability on Amazon Prime (Yes), Apple TV (Yes), and Netflix (No). A large image of Margot Robbie as Barbie is shown. Below the main info card, there's a quote from a user rating: 'As expected "Barbie" is A lot of happy fun! Greta Gerwig is a director I've really come to appreciate and love. Greta is one of the most wholesome filmmakers working today. "Lady bird" is my favourite coming of age story in film and similarly, I have to say right off the bat-this film is ridiculously hilarious. Tongue and cheek? Sure. Just exceptionally written characters also. The quirks between Margot Robbie (Barbie) and Ryan Gosling (Ken) are just magic together on screen.' To the right, there's a smaller image showing three women in costumes.

## 8. Add Rating

The screenshot shows a 'Create Rating' dialog box for the movie 'Forrest Gump'. It has fields for 'Movie Title' (Forrest Gump) and 'Rating (0 - 10)' (6). Below the dialog, there's a 'Comments' field containing 'Good movie'. In the background, there's a 'Search Movie' sidebar with filters for 'Rate Movie', 'Genre' (Comedy, Adventure, Crime, International, Musical, Mystery, Romance), and 'Platform' (Netflix, Amazon Prime, Apple TV). The main content area shows movie cards for 'Inception' (Science Fiction, Christopher Nolan) and 'La La Land' (Musical, Damien Chazelle). A large poster for 'Forrest Gump' is visible on the right.

The screenshot shows a movie information page for 'Forrest Gump'. At the top, there's a search bar with the query 'leonard'. Below it, a sidebar lists filters for 'Search Music', 'Search Album', and 'Search Movie'. Under 'Search Movie', there are filters for 'Rate Movie' (Yes/No), 'Netflix' (Yes/No), 'Amazon Prime' (Yes/No), 'Apple TV' (Yes/No), and 'Genre' (Comedy, Adventure, Crime, International, Musical, Mystery, Romance). The main content area displays the movie 'Forrest Gump' with its title, genre (Romance), actors (Tom Hanks, Robin Wright), director (Robert Zemeckis), and availability on Amazon Prime (Yes), Apple TV (No), and Netflix (Yes). A large image of Tom Hanks as Forrest Gump is shown. Below the main info card, there's a quote from a user rating: 'Good movie'. To the right, there's a smaller image showing Tom Hanks sitting on a bench. A large poster for 'Forrest Gump' is visible on the right.

## Admin-Side Interface

Page name	Interface
1. Login	
2. Homepage	 <p>Welcome!</p> <p><b>About the Database</b></p> <p>Welcome to our Movie and Music Database! Dive into a world of entertainment where you can explore a vast collection of movies, music albums, and songs. Our database is designed to enhance your entertainment experience by providing easy access to information about your favorite movies and albums.</p> <p><b>Database Features</b></p> <ul style="list-style-type: none"> <li>• Search for Movies, Music Albums, and Songs</li> <li>• Discover New Releases and Classics</li> <li>• Rate and Review Movies and Albums</li> <li>• Create Personal Playlists</li> <li>• Get Recommendations Based on Your Preferences</li> </ul> <p><b>How to Get Started</b></p> <p>Getting started is easy! Use the search bar to find movies, music albums, or songs that pique your interest. Explore curated playlists, and don't forget to rate and leave reviews to share your thoughts with the community. Customize your experience by creating personal playlists and receive recommendations tailored just for you.</p> <p><b>Join Our Community</b></p> <p>Connect with other users who share your passion for movies and music. Join discussions, participate in forums, and stay updated on the latest releases. Our community is a place to share your love for</p>
3. Admin Instructions	 <p>Welcome admin</p> <p><b>About</b></p> <p>Welcome to the Admin Dashboard for the Movie and Music Database. As an administrator, you play a crucial role in managing and maintaining the integrity of our database. Here's a guide to help you navigate your administrative responsibilities.</p> <p><b>Admin Features</b></p> <ul style="list-style-type: none"> <li>• Manage User Accounts and Permissions</li> <li>• Add, Edit, or Remove Movies, Music Albums, and Songs</li> <li>• Review and Moderate User Ratings and Reviews</li> <li>• Monitor Database Performance and Security</li> <li>• Generate Reports and Insights</li> <li>• Customize Database Settings</li> </ul> <p><b>How to Manage</b></p> <p>To manage user accounts and permissions, navigate to the Admin Panel. From there, you can add, edit, or remove entries in the database. Keep an eye on user ratings and reviews to ensure a positive community experience. Regularly review database performance and security to maintain optimal functionality.</p> <p><b>Generate Reports</b></p> <p>Leverage the reporting tools to generate insights into user engagement, popular content, and trends. These reports can help you make informed decisions about content curation and database improvements.</p>

#### 4. Manage Albums

Album Name	Num Song	Release Date	Mimetype	Filename	Created Date
NA	0	0			
Thriller	9	1990	image/png	thriller.png	11/25/2023
Wild Honey	12	1967	image/png	Wild_honey_beach_boys.png	11/24/2023
25	11	2015	image/png	2525.png	11/25/2023
1989	13	2014	image/png	1989.png	11/24/2023
Purpose (Deluxe)	19	2015	image/png	purpose-deluxe.png	11/24/2023
Lover	18	2019	image/png	lover.png	11/24/2023
Fine Line	12	2019	image/png	fine-line.png	11/24/2023
The Dark Side of the Moon	10	1973	image/png	dark-side-of-the-moon.png	11/25/2023
Hotel California	8	1976	image/png	hotel-california.png	11/25/2023
Led Zeppelin IV	8	1971	image/png	led-zeppelin.png	11/24/2023
Back in Black	10	1980	image/png	back-in-black.png	11/25/2023
Rumours	11	1977	image/png	rumours.png	11/25/2023
To Pimp a Butterfly	16	2015	image/png	topimpabutterfly.png	11/25/2023
The Marshall Mathers LP	18	2000	image/png	Eminem-The-Marshall-Mathers-LP.png	11/24/2023
Good Kid, M.A.A.D City	13	2012	image/png	good-kid-maad-city.png	11/24/2023
My Beautiful Dark Twisted Fantasy	13	2010	image/png	my-beautiful-dark-twisted-fantasy.png	11/24/2023
DAMN	14	2017	image/png	DAMN.png	11/24/2023
Channel Orange	17	2012	image/png	channel-orange.png	11/25/2023

#### 5. Add Albums

Manage\_Album

Album Name	Rare
Number Songs	17
Release Date	2020
Album Picture	<input type="file" value="download.jpeg"/>
<input type="button" value="Create"/>	

#### 6. Manage Movie

Title	Release Date	Average Rating	Netflix	Amazon Prime	Apple Tv	Filename	Created Date
NA	7/21/2023	0	NA	NA	NA		
Barbie	7/21/2023	7.1	No	Yes	Yes	barbie.png	11/23/2023
The Shawshank Redemption	9/23/1994	9.3	No	Yes	Yes	thesshawshankredemption.png	11/24/2023
Pulp Fiction	10/14/1994	8.9	Yes	No	No	pulpfiction.png	11/24/2023
The Godfather	3/24/1972	9.2	No	Yes	Yes	thegodfather.png	11/24/2023
Forrest Gump	7/6/1994	0	Yes	Yes	No	forrestgump.png	11/24/2023
Inception	7/16/2010	0	No	Yes	No	inception.png	11/24/2023
The Super Mario Bros. Movie	4/5/2023	7.1	No	Yes	Yes	supermariobrosmovie.png	11/24/2023
Spider-Man: Across the Spider-Verse	6/2/2023	8.7	No	Yes	Yes	spidermanatsv.png	11/24/2023
Talk to Me	7/28/2023	7.2	No	Yes	Yes	talktome.png	11/24/2023
Mean Girls	4/30/2004	7.1	Yes	Yes	Yes	meangirls.png	11/24/2023
Clueless	7/19/1995	6.8	No	Yes	Yes	clueless.png	11/24/2023
Rush Hour 1	9/18/1998	7	Yes	Yes	Yes	rushhour1.png	11/24/2023
Luckiest Girl Alive	9/30/2022	0	Yes	No	No	luckiestgirlalive.png	11/24/2023
La La Land	12/16/2016	8	No	Yes	Yes	lalaland.png	11/23/2023
The Greatest Showman	12/20/2017	7.5	No	Yes	Yes	thegreatestshowman.png	11/24/2023
Coco	11/22/2017	8.4	Yes	Yes	Yes	coco.png	11/24/2023
Midsommar	6/3/2019	7.1	No	Yes	Yes	midsommar.png	11/24/2023
Red, White and Royal Blue	7/22/2023	7	No	Yes	No	redwhiteandroyalblue.png	11/24/2023

## 7. Add Movie

The screenshot shows a user interface for managing movies. On the left, there's a sidebar with 'Manage' and 'Manage Movie' sections, listing various movie titles like 'NA', 'Barbie', 'The Shawshank Redemption', etc. The main area has a 'Manage\_Movie' form with fields for Title, Release Date, Average Rating, and a dropdown for Netflix, Amazon Prime, or Apple TV. Below this is a 'Movie Picture' section with a 'Choose File' button. To the right is a table of uploaded files with columns 'Filename' and 'Created Date'. A 'Create' button is visible at the top right of the form.

Filename	Created Date
barbie.png	11/23/2023
theshawshankredemption.png	11/24/2023
pulpfiction.png	11/24/2023
thegodfather.png	11/24/2023
forrestgump.png	11/24/2023
inception.png	11/24/2023
supermariobrosmovie.png	11/24/2023
spidermanatvs.png	11/24/2023
talktome.png	11/24/2023
meangirls.png	11/24/2023
clueless.png	11/24/2023
rushhour.png	11/24/2023
luckiestgirlalive.png	11/24/2023
lalaland.png	11/23/2023
thegreatestshowman.png	11/24/2023
coco.png	11/24/2023
midsommar.png	11/24/2023
redwhiteandroyalblue.png	11/24/2023

## 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:*

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

*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:*

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

*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:*

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

*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:*

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

*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:*

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

*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:*

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