CSCE 5350.001

Fundamentals of Database Systems

Project Part 5

Naga Vara Pradeep Yendluri

11646461

nagavarapradeepyendluri@my.unt.edu

Project Description:

The Movie Producer Management System is an application that is being developed for a movie production company like Universal Studios. The system is designed to store and manage information about the company's movies, artists, songs, employees, and various other aspects of the movie production process. The system will store information about the producing site locations, movie-script-inventory, sponsoring companies, employee data, and payroll. It will also store information about the artists and the movies they have worked on, as well as the various aspects of the movie production process, such as soundtracks, awards, and more.

We have Identified the following entities and relations for the movie producer management system. We have identified the Functional Dependencies to make the database normalized to BCNF or 3NF and identified the dependency preserving and lossless join between the tables also updated the tables accordingly.

1. **Movies:** The entity 'Movies' provides information about the different movies produced by the company. It has 6 attributes, including the movie id, movie title, release date, duration, in production and director. This entity is important for keeping track of the different movies produced by the company and the information related to each movie.
2. **Artists:** The entity 'Artists' provides information about the actors involved in the movies. It has 5 attributes, including the actors id, actors name, actors date of birth, address and age. This entity is important for maintaining the information about the actors, their age and date of birth, which is required for casting actors for various roles.
3. **Genre:** The entity 'Genre' provides information about the genre to which the movie belongs. It has 2 attributes, genre id, and genre name. This entity is important for categorizing the movies into different genres, which helps in better management and analysis of the movies.
4. **Sponsoring Companies:** The entity 'Sponsoring Companies' provides information about the companies that sponsor the movies. It has 4 attributes, including sponsor id, sponsor name, movie id and movie sponsored. This entity is important for tracking the sponsorship deals and the companies that sponsor the movies.

5. **Site Locations:** The entity 'Site Locations' provides information about the different producing sites, including their addresses and buildings. It has 4 attributes, including location id, name, address, and building names. This entity is important for tracking the different producing sites and their details, which is essential for managing the movie production process.
6. **Buildings:** The entity 'Buildings' provides information about the buildings in each producing site. It has 3 attributes, including Building id, name, and type of building. This entity is important for tracking the different types of buildings present in the producing sites, which is essential for managing the resources and maintenance of the buildings.
7. **Movie Script Inventory:** The entity 'Movie Script Inventory' provides information about the movie scripts. It has 5 attributes, including script id, name, movie id, author, and publication date. This entity is important for tracking the different movie scripts and the information related to each script.
8. **Employee:** The entity 'Employee' provides information about the employees of the company. It has 5 attributes, including employee id, name, job title, hourly pay and phone. This entity is important for maintaining the information about the employees, their job title and contact information, which is required for managing the human resources of the company.
9. **Payroll:** The entity 'Payroll' provides information about employee payroll data. It has 5 attributes, including employee id, hours worked, joining date, work date, etc. This entity is important for tracking the payroll information of the employees, which is essential for managing the finances of the company.
10. **Songs:** The entity 'Songs' provides information about different soundtracks used in the movies. It has 4 attributes, including song id, track title, movie id, and singer name. This entity is important for tracking the different soundtracks used in the movies and the information related to each soundtrack.

Latest Schema:

Movie (movie_id, title, avg_rating, date_of_release, duration, script_inventory_id, director, in_production)

Songs (song_id, song_name, singer_name, movie_id)

Genre (genre_id, genre_name)

TaggedWith (movie_id, genre_id)

SiteLocation (location_id, location_name, address)

ShotAt (location_id, movie_id)

Building (building_id, building_name, purpose, location_id)

PostProductionDoneIn (movie_id, building_id)

Employees (employee_id, employee_name, designation, phone_number, hourly_pay, employee_level)

Manages (employee_id, location_id)

Payroll (payroll_id, employee_id, hours_worked, date)

SponsoringCompany (company_id, company_name, num_movies_produced)

getsPaidBy (artist_id, company_id)

Produces (company_id, movie_id)

Artist (artist_id, artist_name, date_of_birth, gender, address)

MovieScriptInventory (script_inventory_id, script_inventory_name)

ActsIn(movie_id, artist_id)

Rating(rating_id, movie_id, rating)

Additional Assumptions:

- From the latest schema we got 18 functional dependencies.
- Normalized the tables into 2NF,3NF and BCNF.

Functional dependencies for the database:

Movie: movie_id → title, date_of_release, duration, script_inventory_id,

director, in_production

MovieRating: movie_id → avg_rating

Songs: song_id → song_name, singer_name, movie_id

Genre: genre_id → genre_name

MovieGenre: (movie_id, genre_id) → {movie_id, genre_id}

SiteLocation: location_id → location_name, address

MovieLocation: (location_id, movie_id) → {location_id, movie_id}

Building: building_id → building_name, purpose, location_id

MoviePostProduction: (movie_id, building_id) → {movie_id, building_id}

Employees: employee_id → employee_name, designation, phone_number, employee_level

EmployeeSalary: employee_id → hourly_pay

Manages: (employee_id, location_id) → {employee_id, location_id}

Payroll: payroll_id → employee_id, hours_worked, date, hourly_pay

SponsoringCompany: company_id → company_name, num_movies_produced

Artist: artist_id → artist_name, date_of_birth, gender, address

ArtistCompany: (artist_id, company_id) → {artist_id, company_id}

MovieScriptInventory: script_inventory_id → script_inventory_name

ActsIn: (movie_id, artist_id) → {movie_id, artist_id}

From the above functional dependencies, we are taking 12 among them.

Normalization:

- **1NF:**

The tables are in 1NF since they do not have repeating groups or multivalued attributes.

- **2NF:**

Movie:

Movie: movie_id → title, date_of_release, duration, script_inventory_id, director, in_production

Candidate key, prime attributes = { movie_id }

Non – prime attributes = { in_production }

Since, both the location_name and address are fully dependent on location_id. There are no partial dependencies, hence it is in 2NF.

Songs:

song_id → song_name, singer_name, movie_id

Candidate key, prime attributes = { song_id }

Non – prime attributes = { song_name, singer_name, movie_id }

All the non – prime attributes are fully dependent on candidate key (song_id). So, it is in 2NF

Genre:

genre_id → genre_name

Candidate key, prime attributes = { genre_id }

Non – prime attributes = { genre_name }

Since, the FD consists of only one attribute on both sides, so it is trivially in 2NF.

Site location:

location_id → location_name, address

Candidate key, prime attributes = { location_id }

Non – prime attributes = { location_name, address }

All the non–prime attributes are fully dependent on the candidate key this is in 2NF.

Payroll:

payroll_id → employee_id, hours_worked, date, hourly_pay

Candidate key, prime attributes = { payroll_id }

Non – prime attributes = { employee_id, hours_worked, date, hourly_pay }

All the non-prime attributes are fully dependent on primary key it is in 2NF.

Artist:

artist_id → artist_name, date_of_birth, gender, address

Candidate key, prime attributes = { artist_id }

Non – prime attributes = { artist_name, date_of_birth, gender, address }

All the non–prime attributes are fully dependent on the candidate key. So, it is in 2NF.

ActsIn:

movie_id, artist_id → movie_id, artist_id

Candidate key, prime attributes = { artist_id, movie_id }

Non – prime attributes = None

Hence there are no partial dependencies and non-prime attributes it is in 2NF.

MoviePostProduction:

movie_id, building_id → movie_id, building_id

Candidate key, prime attributes = { movie_id, building_id }

Non – prime attributes =None

Hence there are no partial dependencies and non-prime attributes it is in 2NF.

Employees:

employee_id → employee_name, designation, phone_number, employee_level

Candidate key, prime attributes = { employee_id }

Non – prime attributes ={ employee_name, designation, phone_number, employee_level }

As there is only one candidate key and no partial dependencies on any part of the candidate key, this relation is already in 2NF.

MovieScriptInventory:

script_inventory_id → script_inventory_name

Candidate key, prime attributes ={ script_inventory_id }

Non – prime attributes =None

Since the given FD has only one attribute on the right-hand side, it is already in 2NF and no further normalization is needed.

- **3NF:**

Movie:

Movie: movie_id → title, date_of_release, duration, script_inventory_id, director, in_production

Candidate key, prime attributes = { movie_id }

Non – prime attributes = { in_production }

Movie_id is the **super key**. There are no transitive dependencies in the relation. Hence, the given relation is in 3NF.

Songs:

song_id → song_name, singer_name, movie_id

Candidate key, prime attributes = { song_id }

Non – prime attributes = { song_name, singer_name, movie_id }

Any combination of attributes that includes **song_id** would be a **super key.**

Genre:

genre_id → genre_name

Candidate key, prime attributes = { genre_id }

Non – prime attributes = { genre_name }

Genre_id is the **super key**. There are no transitive dependencies in the relation. Hence, the given relation is in 3NF.

Site location:

location_id → location_name, address

Candidate key, prime attributes = { location_id }

Non – prime attributes = { location_name, address }

Location_id is the **super key.** There are no transitive dependencies in the relation. Hence, the given relation is in 3NF.

Payroll:

payroll_id → employee_id, hours_worked, date, hourly_pay

Candidate key, prime attributes = { payroll_id }

Non – prime attributes = { employee_id, hours_worked, date, hourly_pay }

Any combination of attributes that includes **payroll_id** would be a **super key**.Since there are no transitive dependencies. Hence, the given relation is in 3NF.

Artist:

artist_id → artist_name, date_of_birth, gender, address

Candidate key, prime attributes = { artist_id }

Non – prime attributes = { artist_name, date_of_birth, gender, address }

Any combination of attributes that includes **artist_id** would be a **super key**. There are no transitive dependencies in the relation. Hence, the given relation is in 3NF.

ActsIn:

movie_id, artist_id → movie_id, artist_id

Candidate key, prime attributes = { artist_id, movie_id }

Non – prime attributes = None

Any combination of attributes that includes both **artist_id** and **movie_id** would be a **superkey** for this relation. There are no transitive dependencies in the relation. Hence, the given relation is in 3NF.

MoviePostProduction:

movie_id, building_id → movie_id, building_id

Candidate key, prime attributes = { movie_id, building_id }

Non – prime attributes =None

The super key is { movie_id, building_id }.

There are no transitive dependencies in the relation. Hence, the given relation is in 3NF.

Employees:

employee_id → employee_name, designation, phone_number, employee_level

Candidate key, prime attributes = { employee_id }

Non – prime attributes ={ employee_name, designation, phone_number, employee_level }

This functional dependency is already in 3NF as there are no transitive dependencies.

MovieScriptInventory:

script_inventory_id → script_inventory_name

Candidate key, prime attributes ={ script_inventory_id }

Non – prime attributes ={}

Since there is only one FD it will satisfy the 3NF condition and it is in 3NF.

- **BCNF:**

Movie:

Movie: movie_id → title, date_of_release, duration, script_inventory_id, director, in_production

Candidate key, prime attributes = { movie_id }

Non – prime attributes = { in_production }

The only candidate key is { movie_id }, and there are no non-trivial dependencies. Therefore, the relation is in BCNF.

Songs:

song_id → song_name, singer_name, movie_id

Candidate key, prime attributes = { song_id }

Non – prime attributes = { song_name, singer_name, movie_id }

The only candidate key is { song_id }, and there are no non-trivial dependencies. Therefore, the relation is in BCNF.

Genre:

genre_id → genre_name

Candidate key, prime attributes = { genre_id }

Non – prime attributes = { genre_name }

The only candidate key is { genre_id }, and there are no non-trivial dependencies. Therefore, the relation is in BCNF.

Site location:

location_id → location_name, address

Candidate key, prime attributes = { location_id }

Non – prime attributes = { location_name, address }

The only candidate key is { location_id }, and there are no non-trivial dependencies. Therefore, the relation is in BCNF.

Payroll:

payroll_id → employee_id, hours_worked, date, hourly_pay

Candidate key, prime attributes = { payroll_id }

Non – prime attributes = { employee_id, hours_worked, date, hourly_pay }

The given relation is not in BCNF. To bring it to BCNF, we need to decompose the relation into two relations:

- R1(employee_id, hourly_pay)
- R2(payroll_id, employee_id, hours_worked, date)

Each relation has a single determinant for each of its attributes, and the relation satisfies BCNF.

Artist:

artist_id → artist_name, date_of_birth, gender, address

Candidate key, prime attributes = { artist_id }

Non – prime attributes = { artist_name, date_of_birth, gender, address }

Since there is only one candidate key and no functional dependencies other than the trivial ones, the relation is automatically in BCNF.

ActsIn:

movie_id, artist_id → movie_id, artist_id

Candidate key, prime attributes = { artist_id, movie_id }

Non – prime attributes = None

Since there are no non-prime attributes and every non-trivial functional dependency in the relation has a candidate key as the determinant, the given relation is in BCNF.

MoviePostProduction:

movie_id, building_id → movie_id, building_id

Candidate key, prime attributes = { movie_id, building_id }

Non – prime attributes =None

Since there are no prime attributes, it is in BCNF.

Employees:

employee_id → employee_name, designation, phone_number, employee_level

Candidate key, prime attributes = { employee_id }

Non – prime attributes ={ employee_name, designation, phone_number, employee_level }

It is also in BCNF since the left-hand side (LHS) contains the candidate key.

MovieScriptInventory:

script_inventory_id → script_inventory_name

Candidate key, prime attributes ={ script_inventory_id }

Non – prime attributes ={}

As the given FD has only one candidate key it is the super key, so it is in BCNF.

Therefore, all the Functional dependencies are in BCNF.

Dependency Preserving and Lossless Join:

Movie:

Movie: movie_id → title, date_of_release, duration, script_inventory_id, director, in_production

Here we have Movie table in which we have attributes of movie_script_invetory table.

Comparing both tables to find whether they have common attributes and dependencies.

Movie:

- movie_id → title
- movie_id → date_of_release
- movie_id → duration
- movie_id → script_inventory_id
- movie_id → director

- movie_id → in_production

Script_Inventory:

- script_inventory_id → {}

The common attribute between the two tables is "script_inventory_id". However, since the "script_inventory_id" attribute is a foreign key in the "Movie" table.

Therefore, based on the information given, we can conclude that the join between the "Movie" and "Script_Inventory" tables is both dependency preserving and lossless.

Songs:

song_id → song_name, singer_name, movie_id

If the "movie_id" attribute in the "song" table refers to a separate table containing information about the movies that the songs belong to, we have the following dependencies in each table:

Song:

- song_id → song_name
- song_id → singer_name
- song_id → movie_id

Movie:

- movie_id → {}

The common attribute between the two tables is "movie_id".

For example, if a movie has two songs with different song_ids but the same singer and song name, then the join would produce an additional tuple that combines the information for those two songs.

Therefore, we can conclude that the join between the "Song" and "Movie" tables is dependency preserving and lossless join.

Genre:

- genre_id → genre_name

There are no other tables to join with, so there are no common attributes to consider.

Based on the above scenario this is dependency preserving,

Since the dependency is a simple one-to-one mapping between genre_id and genre_name, there are no redundant tuples in the table that can be eliminated.

Therefore, we can conclude that the table is lossless.

Site location:

location_id → location_name, address

Location:

- location_id → location_name
- location_id → address

There are no other tables to join with, so there are no common attributes to consider.Based on these dependencies, we can conclude that the table is dependency preserving, as the dependencies are trivially preserved by any join operation.

To determine if the table is lossless, we need to check if the table if there are any redundant tuples, as there are no redundant tuples the join is lossless.

Payroll:

payroll_id → employee_id, hours_worked, date, hourly_pay

These are the individual dependencies.

Payroll:

- payroll_id → employee_id
- payroll_id → hours_worked
- payroll_id → date
- payroll_id → hourly_pay

From the above dependencies we conclude that it is dependency preserving.

Since payroll_id is the primary key of the table, there are no redundant tuples in the table that can be eliminated without losing any information. Therefore, we can conclude that the table is lossless.

The "Payroll" table is both dependency preserving and lossless.

Artist:

artist_id → artist_name, date_of_birth, gender, address

The only table we have is the one containing the information about artists. In this case, we can identify the following dependency in the table:

Artists:

- artist_id → artist_name
- artist_id → date_of_birth

- artist_id → gender
- artist_id → address

Based on these dependencies, we can conclude that the table is dependency preserving, as all of the dependencies are preserved by any join operation.

The "Artists" table is lossless, since there are no redundant tuples that can be eliminated without losing any information.

ActsIn:

movie_id, artist_id → movie_id, artist_id

The combination of values in movie_id and artist_id uniquely determines the values of movie_id and artist_id. This dependency is trivially true, as the right-hand side of the dependency is simply equal to the left-hand side.

The join of the two tables on the common attribute movie_id and artist_id will result in a table with the same attributes as the original tables.

Therefore, the join between these two tables is both dependency preserving and lossless.

MoviePostProduction:

movie_id, building_id → movie_id, building_id

The movie_id and building_id uniquely determine the values of movie_id and building_id. This dependency is trivially true.

Both the primary keys movie_id and building _id, the join will result in the same table without any information loss.

Hence, the MoviePostProduction is lossless and dependency preserving.

Employees:

employee_id → employee_name, designation, phone_number, employee_level

Employees:

- employee_id → employee_name
- employee_id → designation
- employee_id → phone_number
- employee_id → employee_level

Based on these dependencies, we can conclude that the table is dependency preserving, as all of the dependencies are preserved by any join operation.

The "Employees" table is lossless, since there are no redundant tuples that can be eliminated without losing any information.

MovieScriptInventory:

script_inventory_id → script_inventory_name

Since we have no other table, the dependency is preserved.

Since the script_inventory_id is a primary key, there can be no duplicate values of script_inventory_id in the table. Therefore, there can be no redundant tuples in the table, and the table is lossless.

New tables:

Movie:

```
CREATE TABLE Movie (
movie_id INT PRIMARY KEY,
title VARCHAR2(100) NOT NULL,
date_of_release DATE NOT NULL,
duration INT NOT NULL,
script_inventory_id INT REFERENCES MovieScriptInventory(script_inventory_id) NOT NULL,
director VARCHAR2(100) NOT NULL,
in_production CHAR(1) NOT NULL
);
```

MovieRating:

```
CREATE TABLE MovieRating (
movie_id INT PRIMARY KEY REFERENCES Movie(movie_id),
avg_rating NUMBER(3, 1) NOT NULL
);
```

Songs:

```
CREATE TABLE Songs (
song_id INT PRIMARY KEY,
```

song_name VARCHAR2(100) NOT NULL,

singer_name VARCHAR2(100) NOT NULL,

movie_id INT REFERENCES Movie(movie_id) NOT NULL

);

Genre:

CREATE TABLE Genre (

genre_id INT PRIMARY KEY,

genre_name VARCHAR2(100) NOT NULL

);

MovieGenre:

CREATE TABLE MovieGenre (

movie_id INT REFERENCES Movie(movie_id),

genre_id INT REFERENCES Genre(genre_id),

PRIMARY KEY(movie_id, genre_id)

);

SiteLocation:

CREATE TABLE SiteLocation (

location_id INT PRIMARY KEY,

location_name VARCHAR2(100) NOT NULL,

address VARCHAR2(200) NOT NULL

);

MovieLocation:

CREATE TABLE MovieLocation (

location_id INT REFERENCES SiteLocation(location_id),

movie_id INT REFERENCES Movie(movie_id),

PRIMARY KEY(location_id,movie_id)

);

Building:

CREATE TABLE Building (

building_id INT PRIMARY KEY,

building_name VARCHAR2(100) NOT NULL

);

Employees:

CREATE TABLE Employees (

employee_id INT PRIMARY KEY,

hourly_pay NUMBER(6, 2) NOT NULL,

UNIQUE(phone_number)

);

EmployeeDetails:

CREATE TABLE EmployeeDetails (

employee_id INT REFERENCES Employees(employee_id),

employee_name VARCHAR2(100) NOT NULL,

designation VARCHAR2(100) NOT NULL,

employee_level VARCHAR2(100) NOT NULL,

PRIMARY KEY(employee_id)

);

Manages:

CREATE TABLE Manages (

employee_id INT REFERENCES Employees(employee_id),

location_id INT REFERENCES SiteLocation(location_id),

PRIMARY KEY(employee_id, location_id)
);

Payroll:

CREATE TABLE Payroll (
payroll_id INT PRIMARY KEY,
employee_id INT REFERENCES Employees(employee_id) NOT NULL,
hours_worked NUMBER(3, 1) NOT NULL,
date DATE NOT NULL
);

Sponsoring_Company:

CREATE TABLE SponsoringCompany (
company_id INT PRIMARY KEY,
company_name VARCHAR2(100) NOT NULL
);

CompanyMovies:

CREATE TABLE CompanyMovies (
company_id INT REFERENCES SponsoringCompany(company_id),
movie_id INT REFERENCES Movie(movie_id),
PRIMARY KEY(company_id, movie_id)
);

getsPaidBy:

CREATE TABLE getsPaidBy (
artist_id INT PRIMARY KEY REFERENCES Artist(artist_id),
company_id INT REFERENCES SponsoringCompany(company_id) NOT NULL

);

Produces:

```
CREATE TABLE Produces (
company_id INT REFERENCES SponsoringCompany(company_id),
movie_id INT REFERENCES Movie(movie_id),
PRIMARY KEY(company_id, movie_id)
);
```

ArtistDetails:

```
CREATE TABLE ArtistDetails (
artist_id INT REFERENCES Artist(artist_id),
date_of_birth DATE NOT NULL,
gender CHAR(1) NOT NULL,
address VARCHAR2(200) NOT NULL,
PRIMARY KEY(artist_id)
);
```

ActsIn:

```
CREATE TABLE ActsIn (
movie_id INT REFERENCES Movie(movie_id),
artist_id INT REFERENCES Artist(artist_id),
PRIMARY KEY(movie_id, artist_id)
);
```

Rating:

```
CREATE TABLE Rating (
rating_id INT PRIMARY KEY,
```

movie_id INT REFERENCES Movie(movie_id) NOT NULL,
rating NUMBER(1, 1) NOT NULL
);

PostProductionDoneIn:

CREATE TABLE PostProductionDoneIn (
movie_id INT REFERENCES Movie(movie_id),
building_id INT REFERENCES Building(building_id),
PRIMARY KEY(movie_id, building_id)
);

BuildingPurpose:

CREATE TABLE BuildingPurpose (
building_id INT REFERENCES Building(building_id),
purpose VARCHAR2(100) NOT NULL,
PRIMARY KEY(building_id, purpose)
);

BuildingLocation:

CREATE TABLE BuildingLocation (
building_id INT REFERENCES Building(building_id),
location_id INT REFERENCES SiteLocation(location_id),
PRIMARY KEY(building_id, location_id)
);

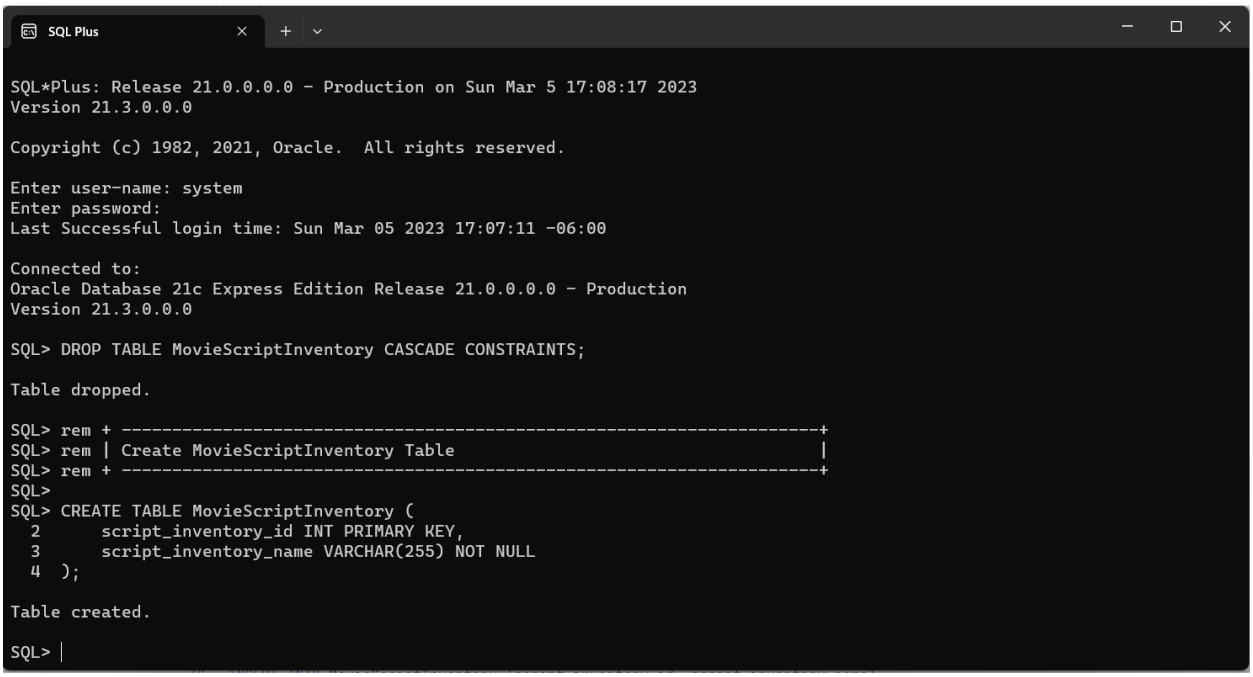
The above tables satisfy the BCNF and 3NF requirements and all functional dependencies.

In summary, we identified 21 non-trivial functional dependencies among the original tables and normalized them into 22 tables that satisfy BCNF and 3NF requirements.

The final Database:

Movie_Script_Inventory:

CREATE TABLE MovieScriptInventory (script_inventory_id INT PRIMARY KEY, script_inventory_name VARCHAR(255) NOT NULL);



Movies:

CREATE TABLE Movie (

movie_id INT PRIMARY KEY,

title VARCHAR2(100) NOT NULL,

date_of_release DATE NOT NULL,

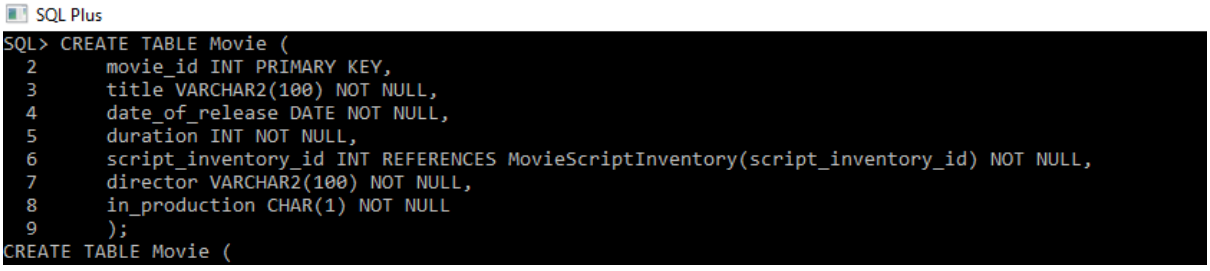
duration INT NOT NULL,

script_inventory_id INT REFERENCES MovieScriptInventory(script_inventory_id) NOT NULL,

director VARCHAR2(100) NOT NULL,

in_production CHAR(1) NOT NULL

);



Movie Rating:

CREATE TABLE MovieRating (movie_id INT PRIMARY KEY REFERENCES Movie(movie_id), avg_rating NUMBER(3, 1) NOT NULL);

```
SQL> CREATE TABLE MovieRating (
  2       movie_id INT PRIMARY KEY REFERENCES Movie(movie_id),
  3       avg_rating NUMBER(3, 1) NOT NULL
  4       );
CREATE TABLE MovieRating (
```

```
SQL> select * from MovieRating;

  MOVIE_ID AVG_RATING
---------- ----------
         1        8.2
         2        7.5
         3          9
         4        6.9
         5        8.7
         6        7.1
         7        9.5
         8        6.5
         9        8.8
        10        7.3

10 rows selected.

SQL>
```

Songs:

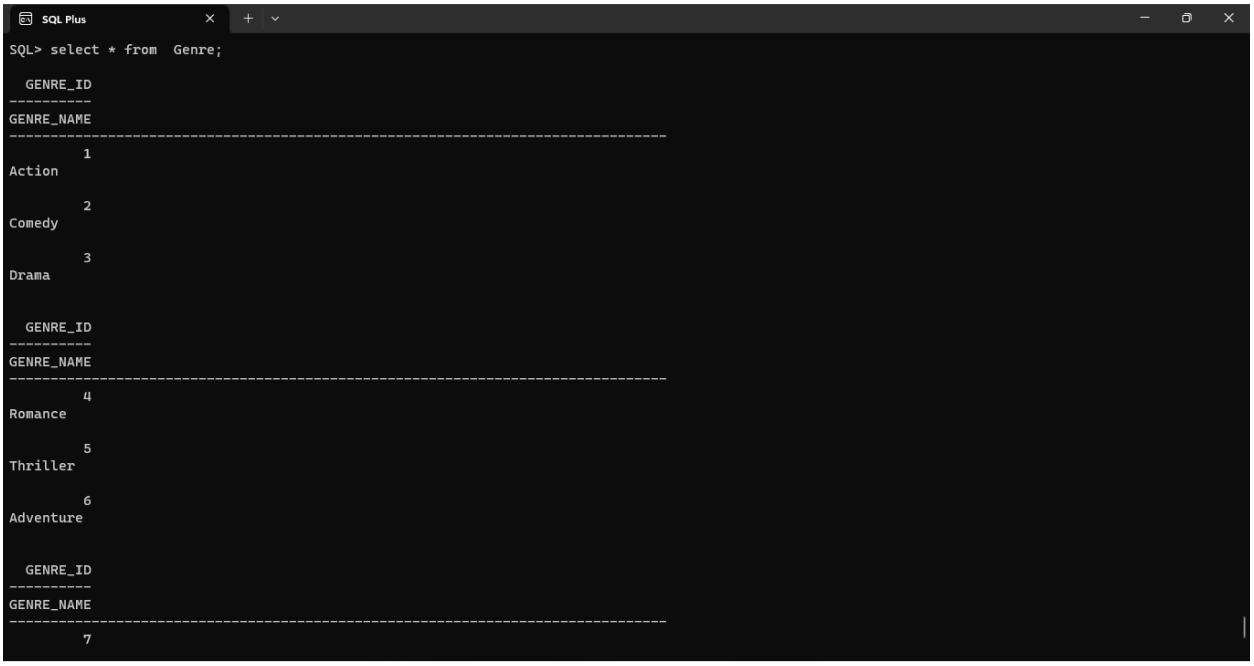
```
CREATE TABLE Songs (
song_id INT NOT NULL,
song_name VARCHAR(255) NOT NULL,
singer_name VARCHAR(255) NOT NULL,
movie_id INT NOT NULL,
PRIMARY KEY (song_id),
FOREIGN KEY (movie_id) REFERENCES Movie(movie_id)
);
```

Genre:

```
CREATE TABLE Genre (
genre_id INT NOT NULL,
genre_name VARCHAR(255) NOT NULL,
```

PRIMARY KEY (genre_id)
);

```
SQL> DROP TABLE Genre CASCADE CONSTRAINTS;

Table dropped.

SQL> rem  +--------------------------------------------------------------------+
SQL> rem  |  Create Genre Table                                                |
SQL> rem  +--------------------------------------------------------------------+
SQL>
SQL> CREATE TABLE Genre (
  2      genre_id INT NOT NULL,
  3      genre_name VARCHAR(255) NOT NULL,
  4      PRIMARY KEY (genre_id)
  5  );

Table created.

SQL>
```



Tagged with:

CREATE TABLE TaggedWith (
movie_id INT NOT NULL,
genre_id INT NOT NULL,
PRIMARY KEY (movie_id, genre_id),
FOREIGN KEY (movie_id) REFERENCES Movie(movie_id),
FOREIGN KEY (genre_id) REFERENCES Genre(genre_id)
);

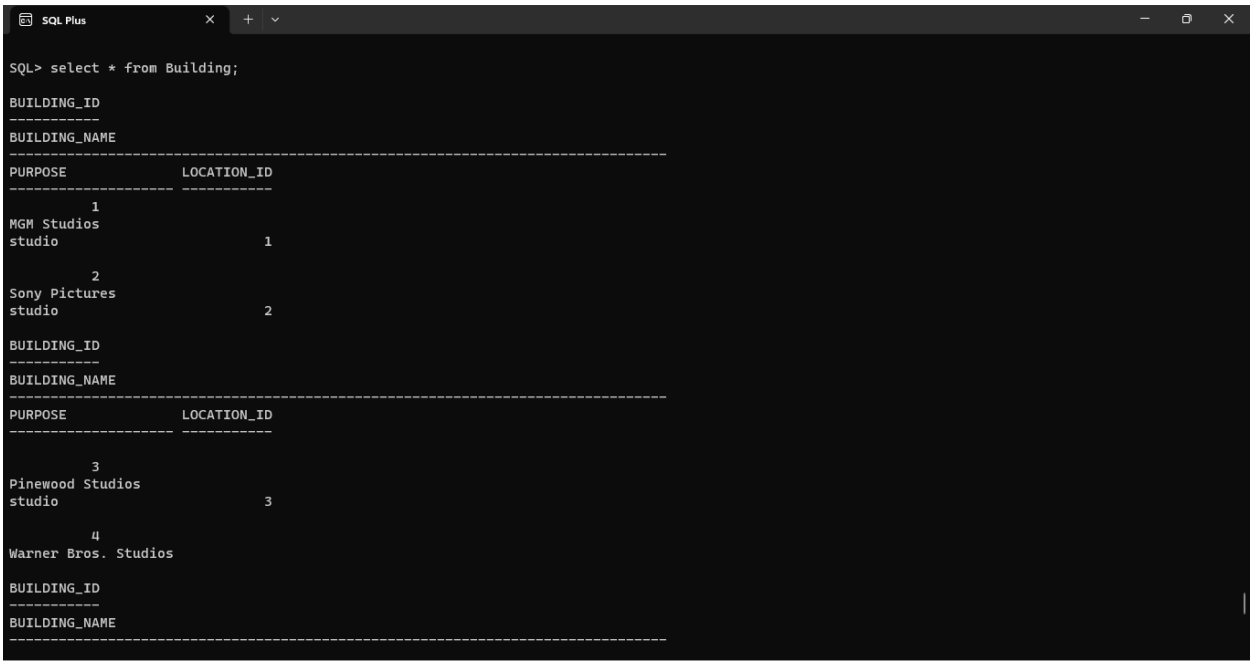
```
SQL> DROP TABLE TaggedWith CASCADE CONSTRAINTS;

Table dropped.

SQL> rem  +--------------------------------------------------------------------+
SQL> rem  |  Create TaggedWith Table                                           |
SQL> rem  +--------------------------------------------------------------------+
SQL>
SQL> CREATE TABLE TaggedWith (
  2      movie_id INT NOT NULL,
  3      genre_id INT NOT NULL,
  4      PRIMARY KEY (movie_id, genre_id),
  5      FOREIGN KEY (movie_id) REFERENCES Movie(movie_id),
  6      FOREIGN KEY (genre_id) REFERENCES Genre(genre_id)
  7  );

Table created.

SQL>
```

```
SQL> select * from TaggedWith;

  MOVIE_ID   GENRE_ID
---------- ----------
         1          4
         1          3
        20          4
         2          3
         3          4
         3          3
        12          4
         5          4
         5          3
        13          9
         6          1

  MOVIE_ID   GENRE_ID
---------- ----------
         6          5
        11          1
         7          5
         8          6
         8          7
        13          8
         9          6
         9          7
        14          8
        10          1
        10          2

22 rows selected.
```

Sitelocation:

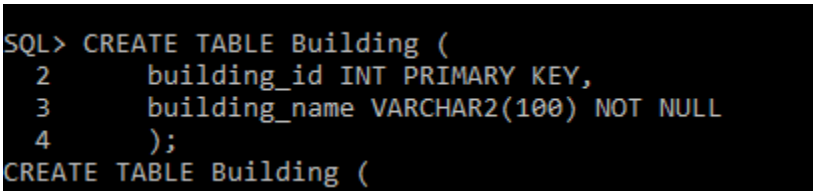
CREATE TABLE SiteLocation (
location_id INT NOT NULL,
location_name VARCHAR(255) NOT NULL,
address VARCHAR(255) NOT NULL,
PRIMARY KEY (location_id)
);

```
SQL> DROP TABLE SiteLocation CASCADE CONSTRAINTS;

Table dropped.

SQL> rem  +-----------------------------------------------------------------------+
SQL> rem  |  Create SiteLocation Table                                            |
SQL> rem  +-----------------------------------------------------------------------+
SQL>
SQL> CREATE TABLE SiteLocation (
  2      location_id INT NOT NULL,
  3      location_name VARCHAR(255) NOT NULL,
  4      address VARCHAR(255) NOT NULL,
  5      PRIMARY KEY (location_id)
  6  );

Table created.

SQL>
```



Building:

CREATE TABLE Building (building_id INT PRIMARY KEY, building_name VARCHAR2(100) NOT NULL);



```
SQL> select * from Building;

BUILDING_ID
-----------
BUILDING_NAME
--------------------------------------------------------------------------------
          1
Empire State Building

          2
Burj Khalifa

          3
Taipei 101


BUILDING_ID
-----------
BUILDING_NAME
--------------------------------------------------------------------------------
          4
Shanghai Tower

          5
The Shard

          6
CN Tower


BUILDING_ID
-----------
BUILDING_NAME
--------------------------------------------------------------------------------
          7
Petronas Towers

          8
Eiffel Tower
```

ShotAt:

CREATE TABLE ShotAt (

location_id INT NOT NULL,

movie_id INT NOT NULL,

PRIMARY KEY (location_id, movie_id),

FOREIGN KEY (location_id) REFERENCES SiteLocation(location_id),

FOREIGN KEY (movie_id) REFERENCES Movie(movie_id)

);

```
SQL> DROP TABLE ShotAt CASCADE CONSTRAINTS;

Table dropped.

SQL> rem  +----------------------------------------------------------------------+
SQL> rem  |  Create ShotAt Table                                                 |
SQL> rem  +----------------------------------------------------------------------+
SQL>
SQL> CREATE TABLE ShotAt (
  2      location_id INT NOT NULL,
  3      movie_id INT NOT NULL,
  4      PRIMARY KEY (location_id, movie_id),
  5      FOREIGN KEY (location_id) REFERENCES SiteLocation(location_id),
  6      FOREIGN KEY (movie_id) REFERENCES Movie(movie_id)
  7  );

Table created.

SQL>
```

```
SQL> select * from ShotAt;

LOCATION_ID   MOVIE_ID
----------- ----------
          1          1
          2          1
          2          2
          3          2
          4          3
          4          4
          5          5
          6          6
          7          7
          7          8
          7          9

LOCATION_ID   MOVIE_ID
----------- ----------
          8         10
          9         11
         10         12
         10         13
         10         14
         10         15
         10         16
         10         17
         10         18
         10         19

21 rows selected.

SQL>
```

PostProductionDoneIn:

CREATE TABLE PostProductionDoneIn (
movie_id INT NOT NULL,
building_id INT NOT NULL,
PRIMARY KEY (movie_id, building_id),
FOREIGN KEY (movie_id) REFERENCES Movie(movie_id),
FOREIGN KEY (building_id) REFERENCES Building(building_id)
);

```
SQL> DROP TABLE PostProductionDoneIn CASCADE CONSTRAINTS;

Table dropped.

SQL> rem + ------------------------------------------------------------------------+
SQL> rem | Create PostProductionDoneIn Table                                      |
SQL> rem + ------------------------------------------------------------------------+
SQL>
SQL> CREATE TABLE PostProductionDoneIn (
  2      movie_id INT NOT NULL,
  3      building_id INT NOT NULL,
  4      PRIMARY KEY (movie_id, building_id),
  5      FOREIGN KEY (movie_id) REFERENCES Movie(movie_id),
  6      FOREIGN KEY (building_id) REFERENCES Building(building_id)
  7  );

Table created.

SQL>
```

```
SQL> select * from PostProductionDoneIn;

  MOVIE_ID BUILDING_ID
---------- -----------
         1           1
         2           2
         3           3
         4           4
         5           5
         6           6
         7           7
         8           8
         9           9
        10          10
        11          11

  MOVIE_ID BUILDING_ID
---------- -----------
        12          12
        13          13
        14          14
        15          15
        16          16
        17          17
        18          18
        19          19
        20          20

20 rows selected.

SQL>
```

Employees:

CREATE TABLE Employees (

employee_id INT NOT NULL,

employee_name VARCHAR(255) NOT NULL,

designation VARCHAR(255) NOT NULL,

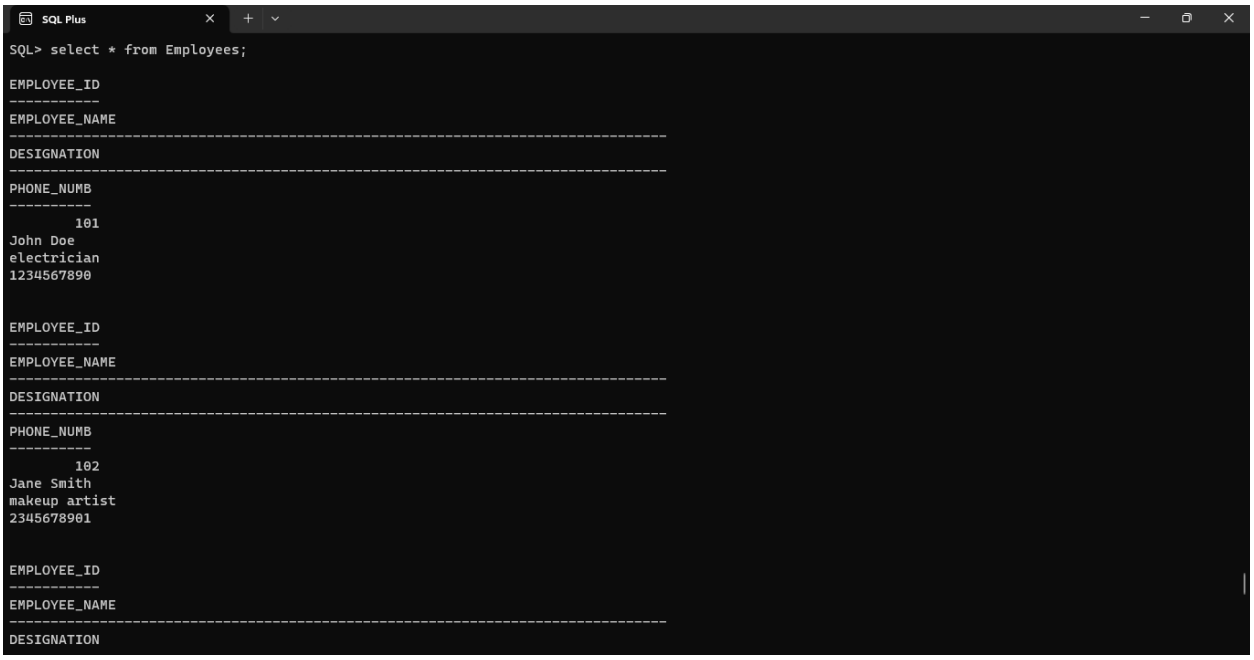
phone_number VARCHAR(10) NOT NULL UNIQUE,

PRIMARY KEY (employee_id),

CONSTRAINT designatoin_constraint_violated CHECK (

designation IN ('choreographers', 'security','sound engineer', 'makeup artist', 'electrician', 'janitor','manager')));

```
SQL> DROP TABLE Employees CASCADE CONSTRAINTS;

Table dropped.

SQL> rem +----------------------------------------------------------------------+
SQL> rem | Create Employees Table                                               |
SQL> rem +----------------------------------------------------------------------+
SQL>
SQL> CREATE TABLE Employees (
  2      employee_id INT NOT NULL,
  3      employee_name VARCHAR(255) NOT NULL,
  4      designation VARCHAR(255) NOT NULL,
  5      phone_number VARCHAR(10) NOT NULL UNIQUE,
  6      PRIMARY KEY (employee_id),
  7      CONSTRAINT designatoin_constraint_violated CHECK (
  8          designation IN (
  9              'choreographer',
 10              'security',
 11              'sound engineer',
 12              'makeup artist',
 13              'electrician',
 14              'janitor',
 15              'manager'
 16          )
 17      )
 18  );

Table created.

SQL>
```



Payroll:

CREATE TABLE Payroll (

payroll_id INT NOT NULL,

salary FLOAT NOT NULL,

employee_id INT NOT NULL,

hours_worked FLOAT NOT NULL,

PRIMARY KEY (payroll_id),

FOREIGN KEY (employee_id) REFERENCES Employees(employee_id),

CONSTRAINT salary_constraint_violated CHECK (salary > 0),

CONSTRAINT hours_constraint_violated CHECK (hours_worked > 0 and hours_worked <= 40)

);

```
SQL> DROP TABLE Payroll CASCADE CONSTRAINTS;

Table dropped.

SQL> rem + ----------------------------------------------------------------------+
SQL> rem | Create Payroll Table                                                 |
SQL> rem + ----------------------------------------------------------------------+
SQL>
SQL> CREATE TABLE Payroll (
  2      payroll_id INT NOT NULL,
  3      salary FLOAT NOT NULL,
  4      employee_id INT NOT NULL,
  5      hours_worked FLOAT NOT NULL,
  6      PRIMARY KEY (payroll_id),
  7      FOREIGN KEY (employee_id) REFERENCES Employees(employee_id),
  8      CONSTRAINT salary_constraint_violated CHECK (salary > 0),
  9      CONSTRAINT hours_constraint_violated CHECK (hours_worked > 0 and hours_worked <= 40)
 10  );

Table created.

SQL>
```

```
SQL> select * from Payroll;

PAYROLL_ID     SALARY EMPLOYEE_ID HOURS_WORKED
---------- ---------- ----------- ------------
         1       5000         101           40
         2       6000         102         38.5
         3       4500         103           37
         4       5500         104         39.5
         5       4000         105           36
         6       6500         106           40
         7       4800         107         38.5
         8       5200         108           37
         9       5800         109         39.5
        10       4200         110           36
        11       5100         111           40

PAYROLL_ID     SALARY EMPLOYEE_ID HOURS_WORKED
---------- ---------- ----------- ------------
        12       5300         112         38.5
        13       4700         113           37
        14       5400         114         39.5
        15       3900         115           36
        16       5900         116           40
        17       4600         117         38.5
        18       5000         118           37
        19       5700         119         39.5
        20       4300         120           36

20 rows selected.

SQL>
```

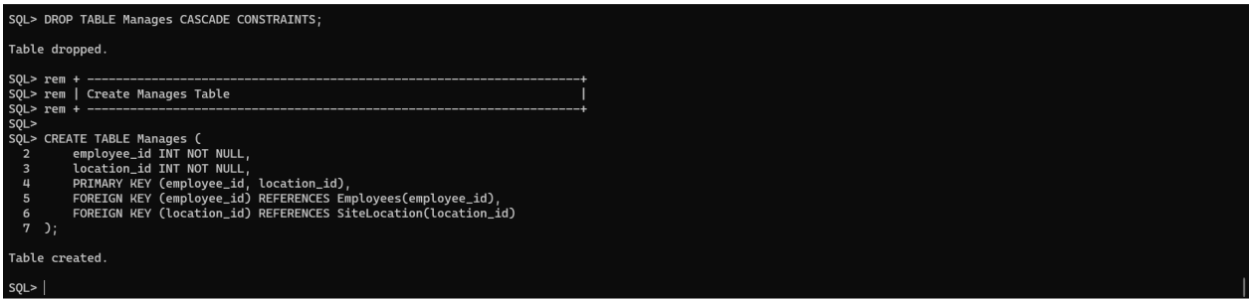
Sponsoring company

CREATE TABLE SponsoringCompany (
company_id INT NOT NULL,
company_name VARCHAR(255) NOT NULL UNIQUE,
PRIMARY KEY (company_id)
);

```
SQL> DROP TABLE SponsoringCompany CASCADE CONSTRAINTS;

Table dropped.

SQL> rem + ----------------------------------------------------------------------+
SQL> rem | Create SponsoringCompany Table                                       |
SQL> rem + ----------------------------------------------------------------------+
SQL>
SQL> CREATE TABLE SponsoringCompany (
  2      company_id INT NOT NULL,
  3      company_name VARCHAR(255) NOT NULL UNIQUE,
  4      PRIMARY KEY (company_id)
  5  );

Table created.

SQL>
```

Manages:

CREATE TABLE Manages (
employee_id INT NOT NULL,
location_id INT NOT NULL,
PRIMARY KEY (employee_id, location_id),
FOREIGN KEY (employee_id) REFERENCES Employees(employee_id),
FOREIGN KEY (location_id) REFERENCES SiteLocation(location_id)
);

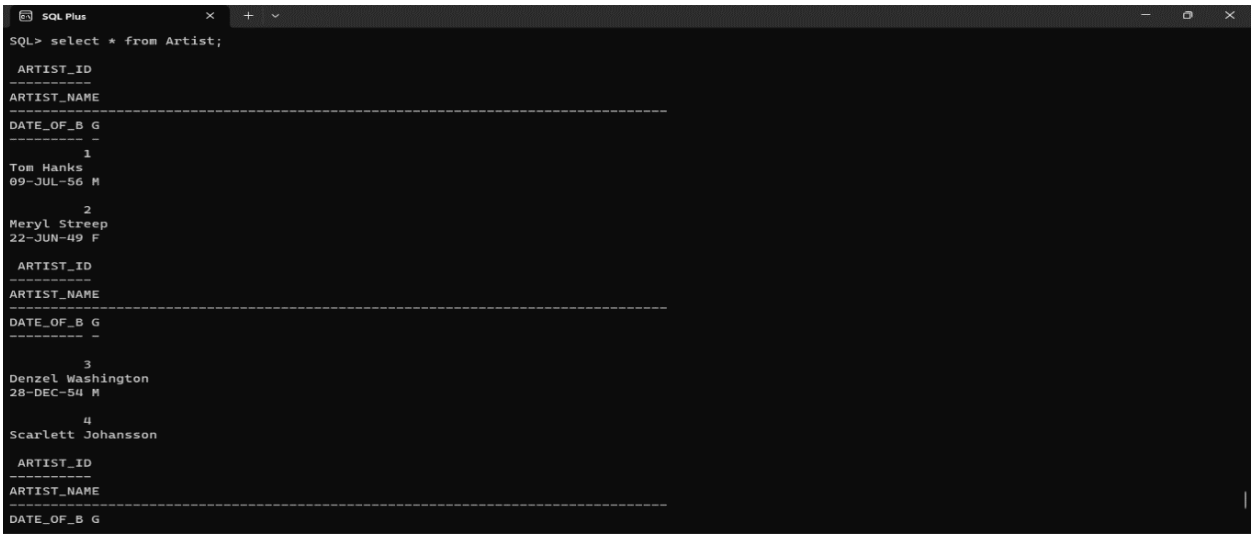


```
SQL> select * from Manages;

EMPLOYEE_ID LOCATION_ID
----------- -----------
        101           1
        102           2
        103           3
        104           4
        105           5
        106           6
        107           7
        108           8
        109           9
        110          10
        101           2

EMPLOYEE_ID LOCATION_ID
----------- -----------
        102           3
        103           4
        104           5
        105           6
        106           7
        107           8
        108           9
        109          10
        110           1

20 rows selected.

SQL>
```

Artist:

CREATE TABLE Artist (
artist_id INT PRIMARY KEY,
artist_name VARCHAR(255) NOT NULL,
date_of_birth DATE NOT NULL,
gender VARCHAR(1) NOT NULL,
CONSTRAINT gender_constraint_violated CHECK (gender in ('M', 'F', 'T'))
);

```
SQL> DROP TABLE Artist CASCADE CONSTRAINTS;

Table dropped.

SQL> rem + ------------------------------------------------------------------------+
SQL> rem | Create Artist Table                                                    |
SQL> rem + ------------------------------------------------------------------------+
SQL>
SQL> CREATE TABLE Artist (
  2    artist_id INT PRIMARY KEY,
  3    artist_name VARCHAR(255) NOT NULL,
  4    date_of_birth DATE NOT NULL,
  5    gender VARCHAR(1) NOT NULL,
  6    CONSTRAINT gender_constraint_violated CHECK (gender in ('M', 'F', 'T'))
  7  );

Table created.

SQL>
```



GetsPaidBy

CREATE TABLE getsPaidBy (
artist_id INT NOT NULL,
company_id INT NOT NULL,
PRIMARY KEY (artist_id, company_id),
FOREIGN KEY (artist_id) REFERENCES Artist(artist_id),
FOREIGN KEY (company_id) REFERENCES SponsoringCompany(company_id)
);

Produces:

```
CREATE TABLE Produces (
company_id INT NOT NULL,
movie_id INT NOT NULL,
PRIMARY KEY (company_id, movie_id),
FOREIGN KEY (company_id) REFERENCES SponsoringCompany(company_id),
FOREIGN KEY (movie_id) REFERENCES Movie(movie_id)
);
```

```
SQL> DROP TABLE Produces CASCADE CONSTRAINTS;

Table dropped.

SQL> rem + ------------------------------------------------------------------------+
SQL> rem | Create Produces Table                                                  |
SQL> rem + ------------------------------------------------------------------------+
SQL>
SQL> CREATE TABLE Produces (
  2      company_id INT NOT NULL,
  3      movie_id INT NOT NULL,
  4      PRIMARY KEY (company_id, movie_id),
  5      FOREIGN KEY (company_id) REFERENCES SponsoringCompany(company_id),
  6      FOREIGN KEY (movie_id) REFERENCES Movie(movie_id)
  7  );

Table created.

SQL>
```

```
SQL> select * from Produces;

COMPANY_ID   MOVIE_ID
---------- ----------
         1          1
         2          3
         3          6
         4          9
         5         12
         6         14
         7         16
         8         18
         9         19
        10         20
         2          1

COMPANY_ID   MOVIE_ID
---------- ----------
         3          2
         4          3
         5          4
         6          5
         7          6
         8          7
         9          8
        10          9
         1         10

20 rows selected.

SQL>
```

ActsIn:

```
CREATE TABLE ActsIn (
movie_id INT NOT NULL,
artist_id INT NOT NULL,
PRIMARY KEY (movie_id, artist_id),
FOREIGN KEY (movie_id) REFERENCES Movie(movie_id),
FOREIGN KEY (artist_id) REFERENCES Artist(artist_id)
);
```

```
SQL> DROP TABLE ActsIn CASCADE CONSTRAINTS;

Table dropped.

SQL> rem + ----------------------------------------------------------------------+
SQL> rem | Create ActsIn Table                                                  |
SQL> rem + ----------------------------------------------------------------------+
SQL>
SQL> CREATE TABLE ActsIn (
  2      movie_id INT NOT NULL,
  3      artist_id INT NOT NULL,
  4      PRIMARY KEY (movie_id, artist_id),
  5      FOREIGN KEY (movie_id) REFERENCES Movie(movie_id),
  6      FOREIGN KEY (artist_id) REFERENCES Artist(artist_id)
  7  );

Table created.

SQL>
```

```
SQL> select * from ActsIn;

  MOVIE_ID  ARTIST_ID
---------- ----------
         1          1
         2          2
         3          3
         4          4
         5          5
         6          6
         7          7
         8          8
         9          9
        10         10
        12         15

  MOVIE_ID  ARTIST_ID
---------- ----------
         7         14
        19         13
         2         12
        15         13
         4         10
         8         11
        18         12
        13         12
         5         12

20 rows selected.

SQL>
```

Individual Contribution:

Identified Functional Dependencies in Movie and Songs table.

Designed Movie and Songs tables in 2NF.

Modified Movie and Songs tables in 3NF.

Modified Movie and Songs tables to BCNF.

Found Dependency preserving and loss less join in Movie and Songs tables.

Created table for Movie and Songs.