

**SYMBIOSIS INSTITUTE OF TECHNOLOGY**

**DBMS Project Report on**

**Movie and T.V. Series Recommendation System**

SUBMITTED BY

**DIVILI SATHVIK MAYANK ARORA**

**18070124025 18070124040**

**ARIPIRALA SAI SRIVATHSAV PRANAV SHARMA**

**18070124014 18070124051**

**Under the Guidance of**

**Prof. Shruti Patil**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**SYMBIOSIS INSTITUTE OF TECHNOLOGY, PUNE**

**PUNE-412115**

**2020-21**

# INDEX

1. Introduction 1
2. Problem Statement, System Architecture, Functional Requirements 1
   1. Problem Statement 1
   2. Objectives 1
   3. System Architecture 2
   4. Functionalities 2
3. Entities, Attributes, and their Relationships 4
4. ER /EER diagram 6
5. Relational Schema 7
6. Keys in each Relation 8
7. Codd’s Rules 10
8. Anomalies 13
9. Functional Dependency 13
   1. Functional Dependency 13
   2. Functional Dependency Chart 16
10. Normalization 22
11. Implementation 23
    1. Tables 23
    2. Table Data 26
12. Queries, Function, Procedures and Triggers 32
    1. 25 Queries 32
    2. 5 Functions 39
    3. 5 Procedures 42
    4. 5 Triggers 44
13. **Introduction**

This is a report for the project “**Movie and T.V. series recommendation system**” which consists of problem we are trying to solve along with the solution, functional requirements, and all other initially essential details of the project. This project is an implementation model of movie recommendation system’s database in MySQL.

1. **Problem Statement, System Architecture, Functional Requirements**
   1. **Problem Statement**

With the rapid development and advancement of digital world especially in the field of media and entertainment, a lot of content to watch online is being accumulated on various streaming platforms like OTTs etc. This availability of huge amount of content, now turned out into a boon and bane situation because though it provides users with various choices and options to get entertained on the other hand it also results in few practical problems like:

* User gets perplexed with more than enough choices in-hand.
* Finding the right OTT platform for a movie/T.V. series he wishes to watch becomes difficult.
* Lacks the right suggestion which suits his/her taste.
  1. **Objectives**

So, the project “Movie and T.V. series recommendation system” is a solution that helps the user to get recommended with a proper movie or T.V. series based upon various parameters like genre, actor/actress, rating, languages etc. and help the user to know on which OTT platform the respective movie/ T.V. series is available. This helps the user to know the list of movies that perfectly matches his choices thus reducing the chance of getting confused and intimating where to watch it.

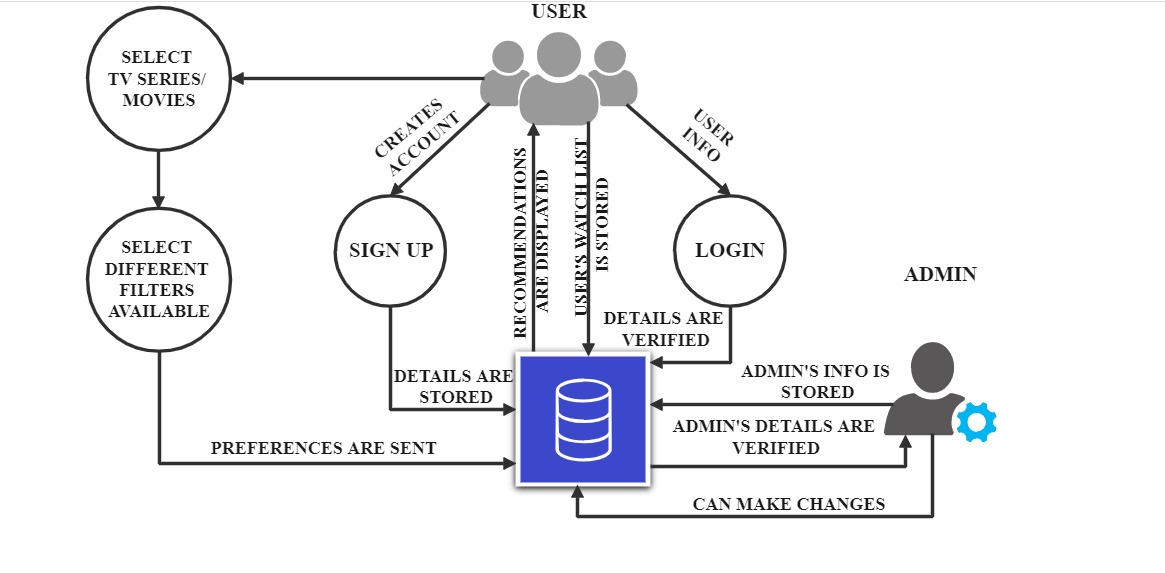
The System’s Architecture/Modules would be as follows:

**USER MODULE**

* Users need to login with his unique id and password.
* User can make use of various filters that are available, based upon his choices and can get appropriate recommendations.

**ADMIN MODULE**

* Admin needs to login with his unique id and password
* Admin can make changes to the database and can make changes to the filters (add new ones or delete few) that are available for users to use.
  1. **System Architecture**

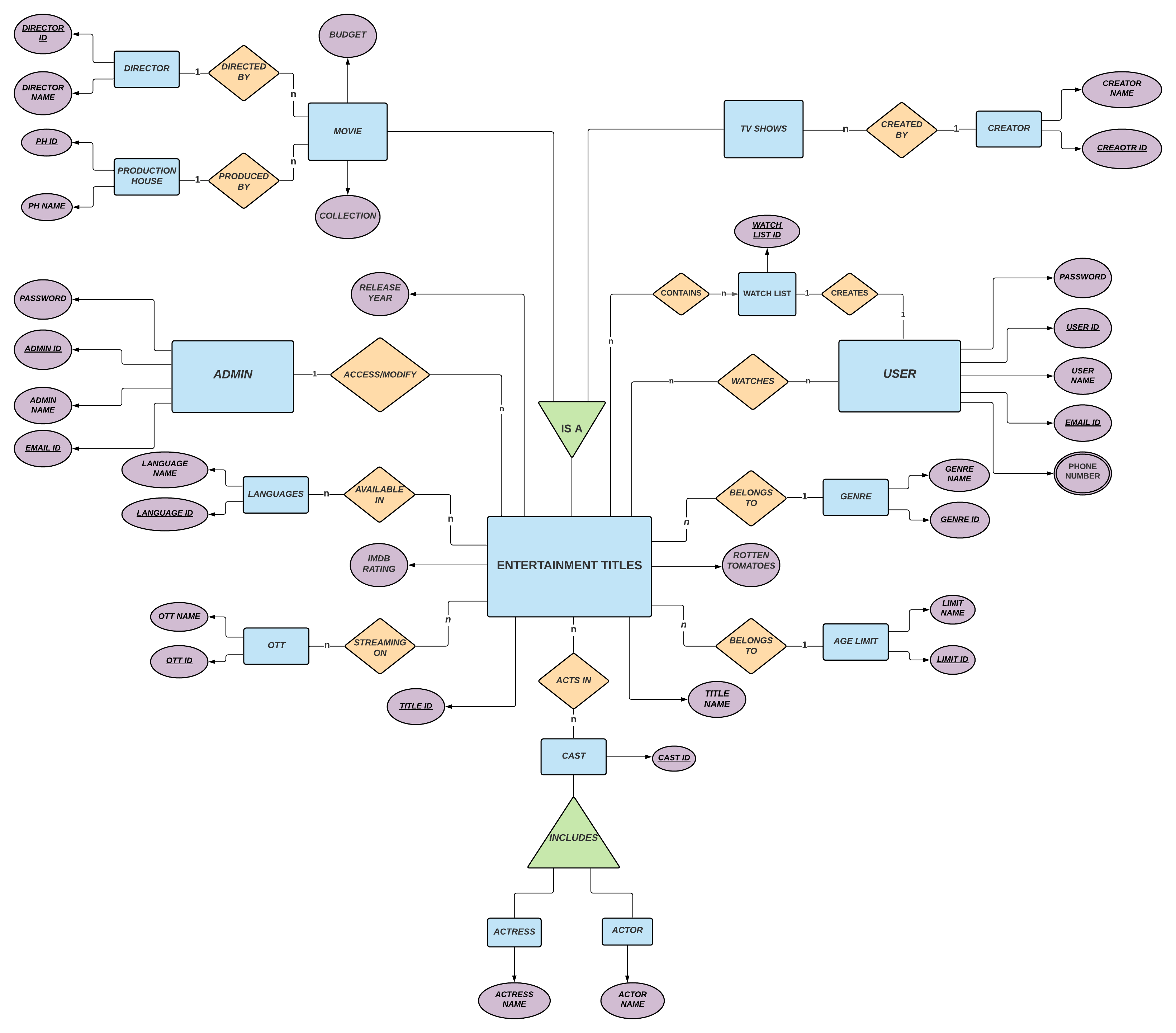


* 1. **Functionalities**
* Every user should sign up using his/her unique id and password.
* Initially user is provided with 2 options to select between movie and T.V. series which are under entertainment title.
* After selecting one of the options, respective filters which are available under movies and T.V. series are displayed to choose between them.
* Available filters for movies are name of the actor, actress, director, genre, age limit, ratings, release year, language, budget, box-office collection, streaming platform and production house.
* Available filters for T.V. shows are name of the actor, actress, genre, age limit, ratings, release year, language, streaming platform, creator of the show.
* User can also maintain a common list of all the movies and T.V. shows which he had already watched.
* Admin should also sign up using his/her unique id and password.
* Admin has the access to both movies and T.V. series to modify them.

|  |  |
| --- | --- |
| **ENTITES** | **ATTRIBUTES** |
| Entertainment Titles (ET) | titleName, titleID, imdbRating, rottenTomatoesRating, Release year, |
| Movie | budget, collection |
| TVshows | **-** |
| Cast | castID |
| Actor | actorName |
| Actress | actressName |
| OTT | ottName, ottID |
| AgeLimit | limitName, limitID |
| Genre | genreName, genreID |
| Language | languageName, languageID |
| Admin | adminName, adminID, emailID, password |
| User | userName, userID, emailID, password, PhoneNumber |
| Watchlist | WatchID |
| Creator | creatorName, creatorID |
| Director | directorName, directorID |
| ProductionHouse | phName, phID |

1. **Entities, Attributes, and their Relationships.**

|  |  |
| --- | --- |
| **RELATIONSHIP** | **CARDINALITY** |
| ET ***is a*** Movie. | **-** |
| ET ***is a*** TV show. | **-** |
| Cast ***acts in*** ET. | Many to many |
| Cast ***includes*** Actor. | **-** |
| Cast ***includes*** Actress. | **-** |
| ET ***has*** Age Limit. | Many to one |
| ET ***Streaming on*** OTT. | Many to many |
| ET ***belongs to a*** Genre. | Many to one |
| ET ***Available in*** Languages. | Many to many |
| Admin ***modifies*** ET. | One to many |
| User ***watches*** ET. | Many to many |
| User ***Creates*** Watchlist. | One to one |
| Watchlist ***Contains*** ET. | Many to many |
| TV Shows ***created by*** Creator. | Many to one |
| Movie ***directed by*** Director. | Many to one |
| Movie ***produced by*** Production House. | Many to one |

1. **ER (EER) Diagram**
2. **Relational Schema**
   1. Title

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| titleID | titleName | | rottenTomatoesRating | | imdbRating | releaseYear |
| *genreID* | *limitID* | *adminID* | |

* 1. Movie

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| titleID | budget | collection | *directorID* | *phID* |

* 1. TVshows

|  |  |
| --- | --- |
| titleID | *creatorID* |

* 1. Genre

|  |  |
| --- | --- |
| genreID | genreName |

* 1. AgeLimit

|  |  |
| --- | --- |
| limitID | limitName |

* 1. OTT

|  |  |
| --- | --- |
| ottID | ottName |

* 1. Languages

|  |  |
| --- | --- |
| languageID | languageName |

* 1. Cast

|  |  |
| --- | --- |
| castID | castName |

* 1. Director

|  |  |
| --- | --- |
| directorID | directorName |

* 1. ProductionHouse

|  |  |
| --- | --- |
| phID | phName |

* 1. Creator

|  |  |
| --- | --- |
| creatorID | creatorName |

* 1. User

|  |  |  |  |
| --- | --- | --- | --- |
| userID | userName | password | emailID |

* 1. PhoneNumber

|  |  |  |
| --- | --- | --- |
| phoneID | *userID* | phone |

* 1. Admin

|  |  |  |  |
| --- | --- | --- | --- |
| adminID | adminName | password | emailID |

* 1. ActsIn

|  |  |  |
| --- | --- | --- |
| actingID | *titleID* | *castID* |

* 1. StreamingOn

|  |  |  |
| --- | --- | --- |
| streamID | *titleID* | *ottID* |

* 1. AvailableIn

|  |  |  |
| --- | --- | --- |
| availID | *titleID* | *languageID* |

* 1. Watches

|  |  |  |
| --- | --- | --- |
| watchID | *titleID* | *userID* |

* 1. Watchlist

|  |  |
| --- | --- |
| listID | *userID* |

* 1. Contain

|  |  |  |
| --- | --- | --- |
| containID | *titleID* | *listID* |

1. **Keys in each Relation**
   1. Title
      1. Candidate Key: titleID, titleName
      2. Primary Key: titleID
      3. Foreign Key: genreID, limitID, adminID
      4. Alternate Key: titleName
   2. Movie
      1. Candidate Key: titleID
      2. Primary Key: titleID
      3. Foreign Key: directorID, phID
      4. Alternate Key: NULL
   3. TV Show
      1. Candidate Key: titleID
      2. Primary Key: titleID
      3. Foreign Key: creatorID
      4. Alternate Key: NULL
   4. Genre
      1. Candidate Key: genreID, genreName
      2. Primary Key: genreID
      3. Foreign Key: NULL
      4. Alternate Key: genreName
   5. AgeLimit
      1. Candidate Key: limitID, limitName
      2. Primary Key: limitID
      3. Foreign Key: NULL
      4. Alternate Key: limitName
   6. OTT
      1. Candidate Key: ottID, ottName
      2. Primary Key: ottID
      3. Foreign Key: NULL
      4. Alternate Key: ottName
   7. Language
      1. Candidate Key: languageID, languageName
      2. Primary Key: languageID
      3. Foreign Key: NULL
      4. Alternate Key: languageName
   8. Actor
      1. Candidate Key: castID
      2. Primary Key: castID
      3. Foreign Key: NULL
      4. Alternate Key: NULL
   9. Director
      1. Candidate Key: directorID
      2. Primary Key: directorID
      3. Foreign Key: NULL
      4. Alternate Key: NULL
   10. ProductionHouse
       1. Candidate Key: phID, phName
       2. Primary Key: phID
       3. Foreign Key: NULL
       4. Alternate Key: phName
   11. Creator
       1. Candidate Key: creatorID
       2. Primary Key: creatorID
       3. Foreign Key: NULL
       4. Alternate Key: NULL
   12. User
       1. Candidate Key: userID, emailID
       2. Primary Key: userID
       3. Foreign Key: NULL
       4. Alternate Key: emailID
   13. PhoneNumber
       1. Candidate Key: phoneID, phone
       2. Primary Key: phoneID
       3. Foreign Key: userID
       4. Alternate Key: phone
   14. Admin
       1. Candidate Key: adminID, emailID
       2. Primary Key: adminID
       3. Foreign Key: NULL
       4. Alternate Key: emailID
   15. ActsIn
       1. Candidate Key: actingID
       2. Primary Key: actingID
       3. Foreign Key: titleID, castID
       4. Alternate Key: NULL
   16. StreamingOn
       1. Candidate Key: streamID
       2. Primary Key: streamID
       3. Foreign Key: titleID, ottID
       4. Alternate Key: NULL
   17. AvailableIn
       1. Candidate Key: availID
       2. Primary Key: availID
       3. Foreign Key: titleID, languageID
       4. Alternate Key: NULL
   18. Watches
       1. Candidate Key: watchID
       2. Primary Key: watchID
       3. Foreign Key: titleID, userID
       4. Alternate Key: NULL
   19. Watchlist
       1. Candidate Key: listID
       2. Primary Key: listID
       3. Foreign Key: userID
       4. Alternate Key: NULL
   20. Contains
       1. Candidate Key: containID
       2. Primary Key: containID
       3. Foreign Key: titleID, listID
       4. Alternate Key: NULL
2. **Codd’s Rules**

### **Rule 0:**

Definition: This rule states that for a system to qualify as an **RDBMS**, it must be able to manage database entirely through the relational capabilities.

Justification: Yes, this the primary rule for all databases to satisfy and this database is also able to manage entirely through its relational capabilities. Hence this rule is applied.

### **Rule 1: Information rule:**

Definition: All information (including metadata) is to be represented as stored data in cells of tables. The rows and columns have to be strictly unordered.

Justification: This database has all the data in the form of rows and columns, also the data in each row is unique, every table has its own and unique value called primary key, each cell contains only one value and the order of rows and columns doesn’t affect the meaning of the tables. Hence this rule is applied.

### **Rule 2: Guaranteed Access:**

Definition: Each unique piece of data (atomic value) should be accessible by: **Table Name + Primary Key (Row) + Attribute(column)**.

**NOTE:** Ability to directly access via POINTER is a violation of this rule.

Justification: Since each table in this database has its own primary key it is possible to access every atomic value logically without using any pointer or physical address of the data. Hence this rule is applied.

### **Rule 3: Systematic treatment of NULL:**

Definition: Null has several meanings, it can mean missing data, not applicable or no value. It should be handled consistently. Also, Primary key must not be null, ever. Expression on NULL must give null.

Justification: This database will not have any null value (missing value, unknown value or unacceptable value) and most importantly any primary key will not be null. Hence this rule is applied.

### **Rule 4: Active Online Catalog:**

Definition: Database dictionary(catalogue) is the structure description of the complete **Database** and it must be stored online. The Catalogue must be governed by same rules as rest of the database. The same query language should be used on catalogue as used to query database.

Justification: As this database do not have any database dictionary this rule is not applied.

**Rule 5: Powerful and Well-Structured Language:**

Definition: One well-structured language must be there to provide all manners of access to the data stored in the database. Example: **SQL**, **QBE** etc. If the database allows access to the data without the use of this language, then that is a violation.

Justification: Since our database accepts SQL (which is one of SQL/QBE) to access, modify and manipulate this rule is also applied.

### **Rule 6: View Updation Rule:**

Definition: All the view that are theoretically updatable should be updatable by the system as well.

Justification: The view that is given in the database to the user is also in the form of tables only and not any other alternative virtual way. Hence this rule is not applied.

### **Rule 7: Relational Level Operation:**

Definition: There must be Insert, Delete, Update operations at each level of relations. Set operation like Union, Intersection and minus should also be supported.

Justification: This database supports all the set operations and relational algebra which include join, union, intersection, minus, delete, insert etc. Hence this rule is also applied.

### **Rule 8: Physical Data Independence:**

Definition: The physical storage of data should not matter to the system. If say, some file supporting table is renamed or moved from one disk to another, it should not affect the application.

Justification: This database and its data access methods will not be affected with the change in physical storage of data hence it satisfies the rule of physical data independence.

### **Rule 9: Logical Data Independence:**

Definition: If there is change in the logical structure (table structures) of the database the user view of data should not change. Say, if a table is split into two tables, a new view should give result as the join of the two tables. This rule is most difficult to satisfy.

Justification: This rule is not applied by this database.

### **Rule 10: Integrity Independence:**

Definition: The database should be able to enforce its own integrity rather than using other programs. Key and Check constraints, trigger etc, should be stored in Data Dictionary. This also make **RDBMS** independent of front-end.

Justification: Irrespective of the front-end, database has its own integrity and key values are stored in data dictionary. Hence this rule is applied.

### **Rule 11: Distribution Independence:**

Definition: A database should work properly regardless of its distribution across a network. Even if a database is geographically distributed, with data stored in pieces, the end user should get an impression that it is stored at the same place. This lays the foundation of **distributed database**.

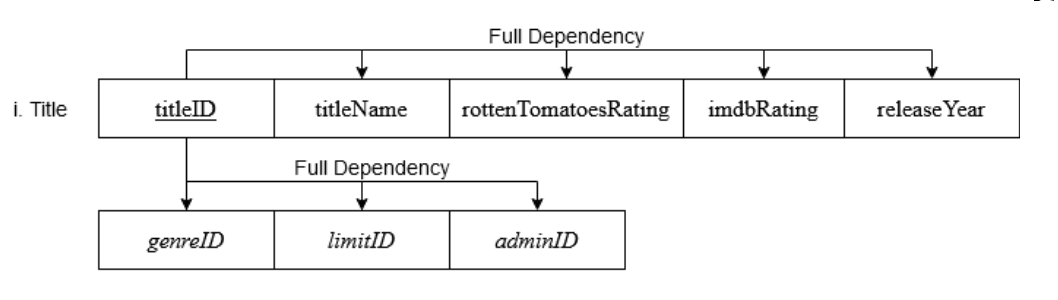
Justification: As we are not sure about the distribution of the database and how it could affect the data manipulation, we are not sure this rule could be applied or not. Hence considering this rule is not applied

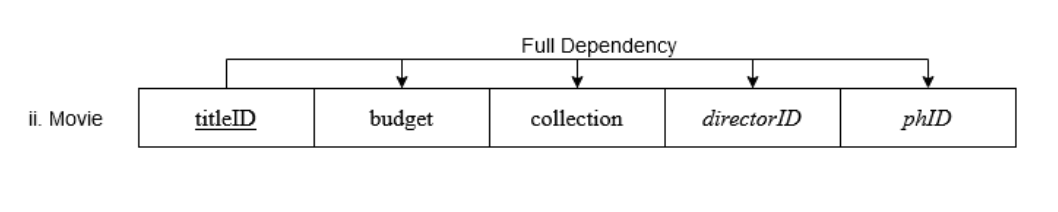
### **Rule 12: Non-Subversion Rule:**

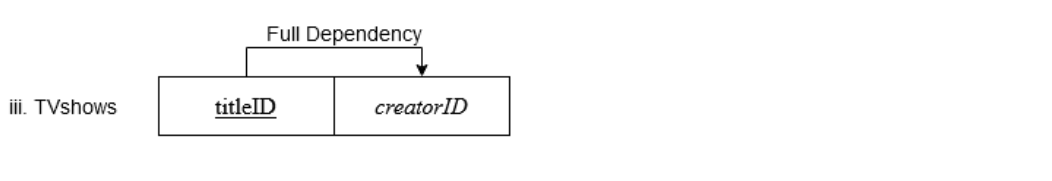
Definition: If low level access is allowed to a system it should not be able to subvert or bypass integrity rules to change the data. This can be achieved by some sort of looking or encryption.

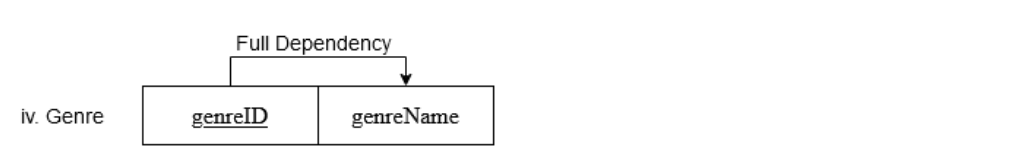
Justification: Since this database is made in MySQL and it forms an instant of the tables while we view data there will be no changes applied to the actual tables and data which will not allow to circumvent data integrity and security at low level access. Hence this rule is applied.

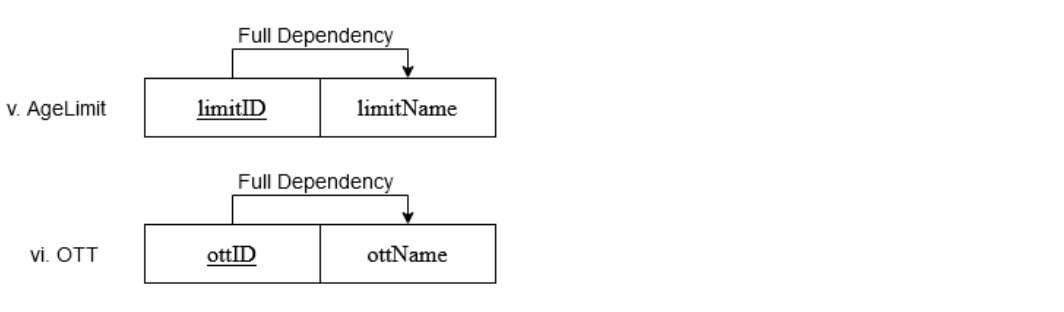
1. **Anomalies**
   1. **Insertion: -**
      1. A new title cannot be added without a valid genreID, limitID, adminID, directorID, creatorID and phID.
      2. A new watchlist cannot be created without a valid userID
   2. **Updation: -**
      1. A title cannot be updated when the genreID, limitID, adminID, directorID, creatorID and phID is invalid.
      2. A watchlist cannot be updated so it contains an invalid userID.
   3. **Deletion: -**
      1. A user cannot be deleted if he/she has a non-empty watchlist.
2. **Functional Dependencies**
   1. **Functional Dependencies**

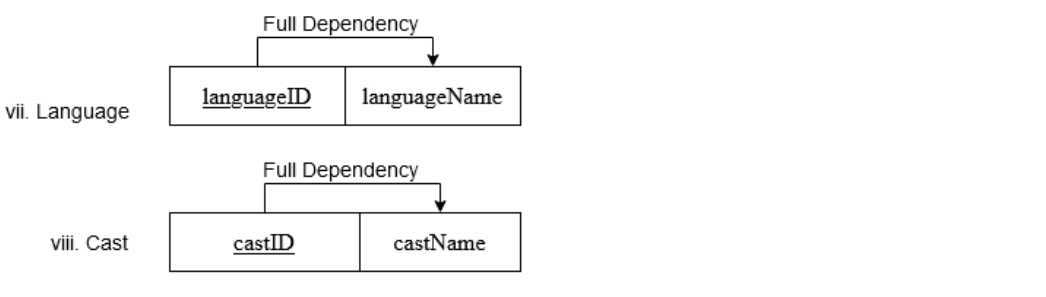


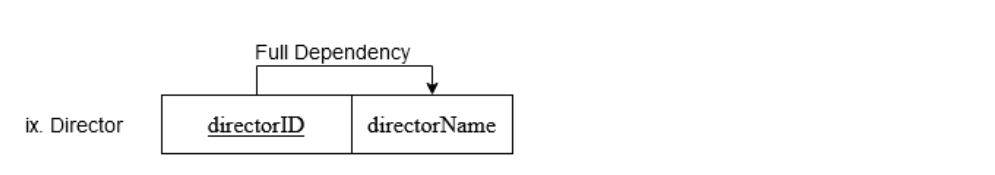


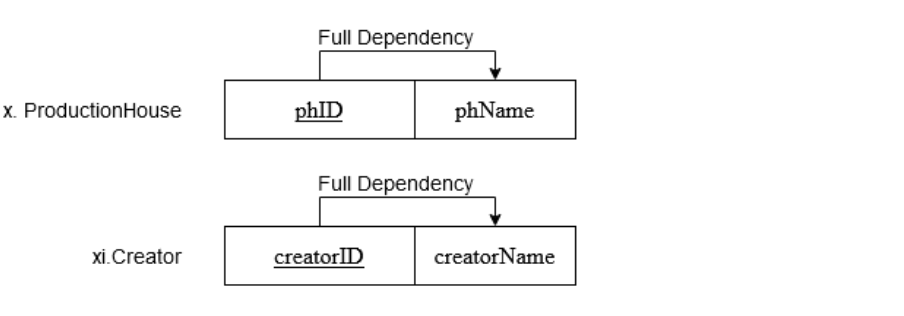


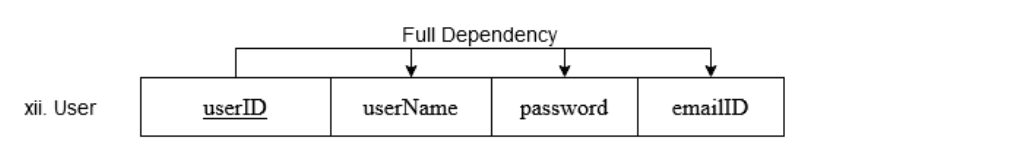


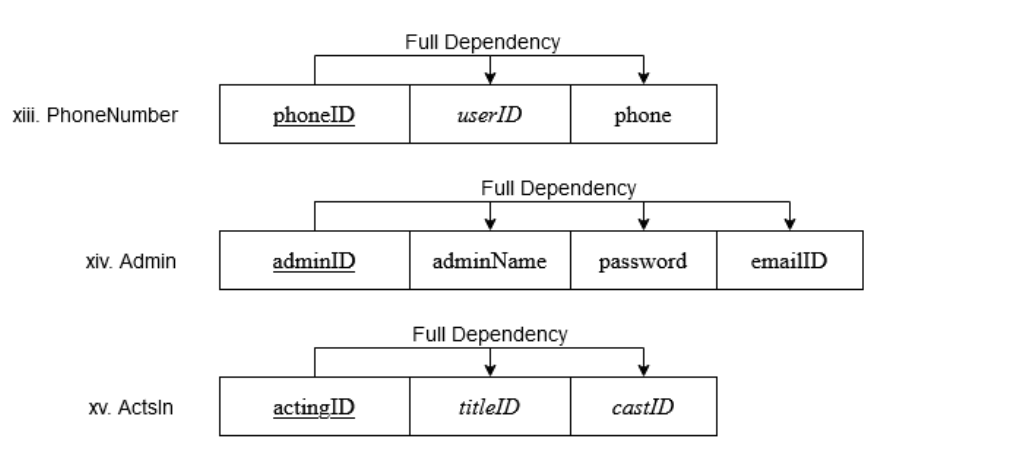


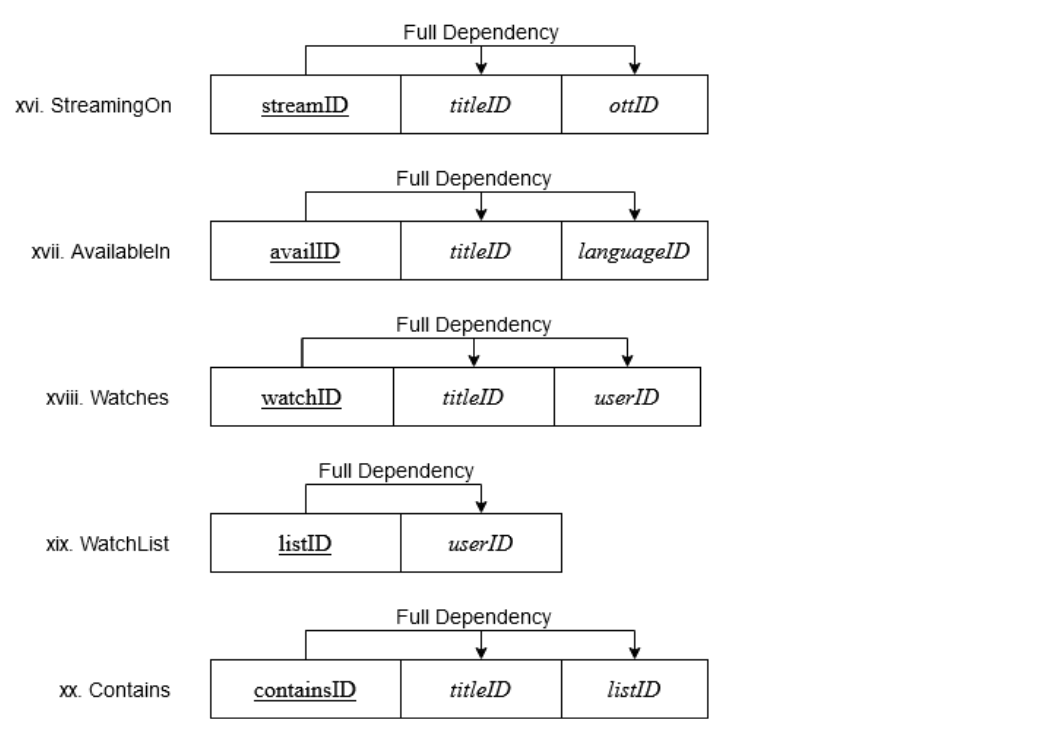




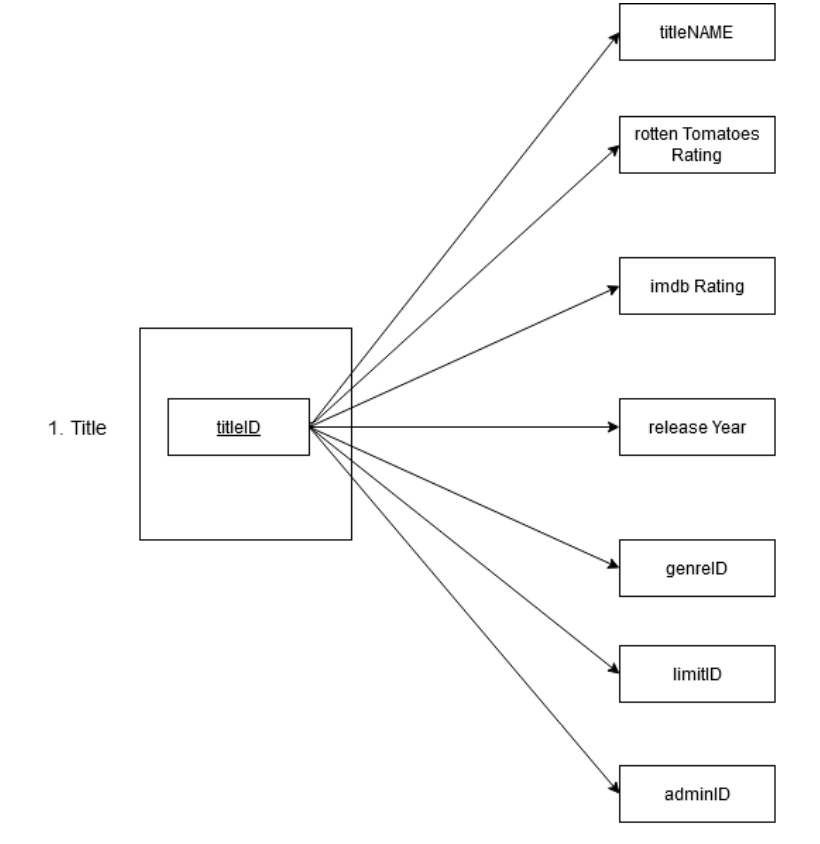


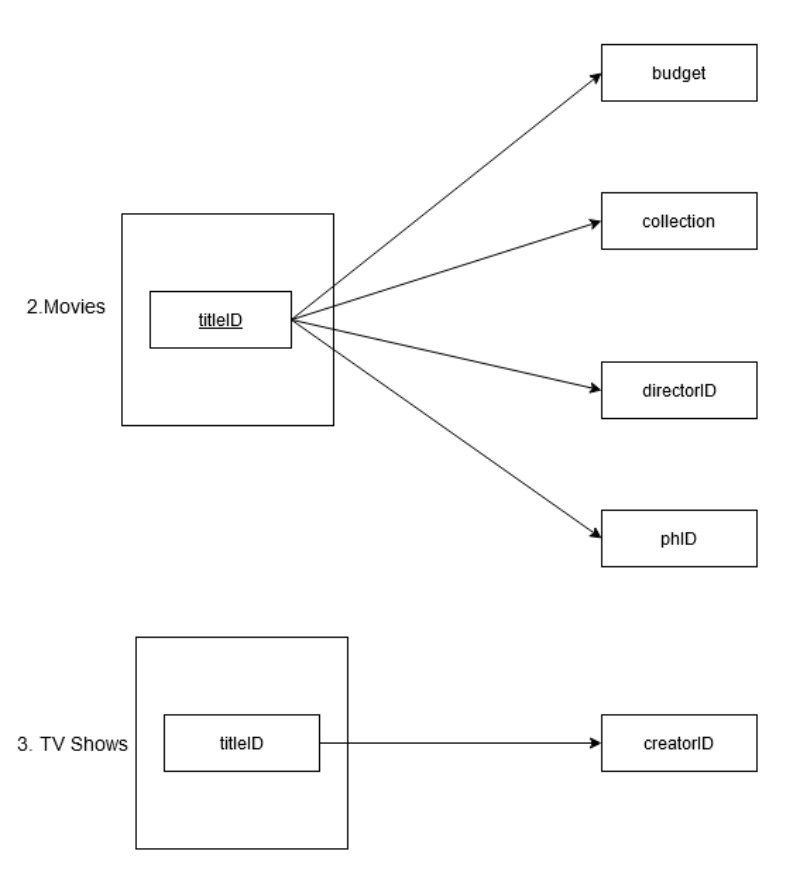


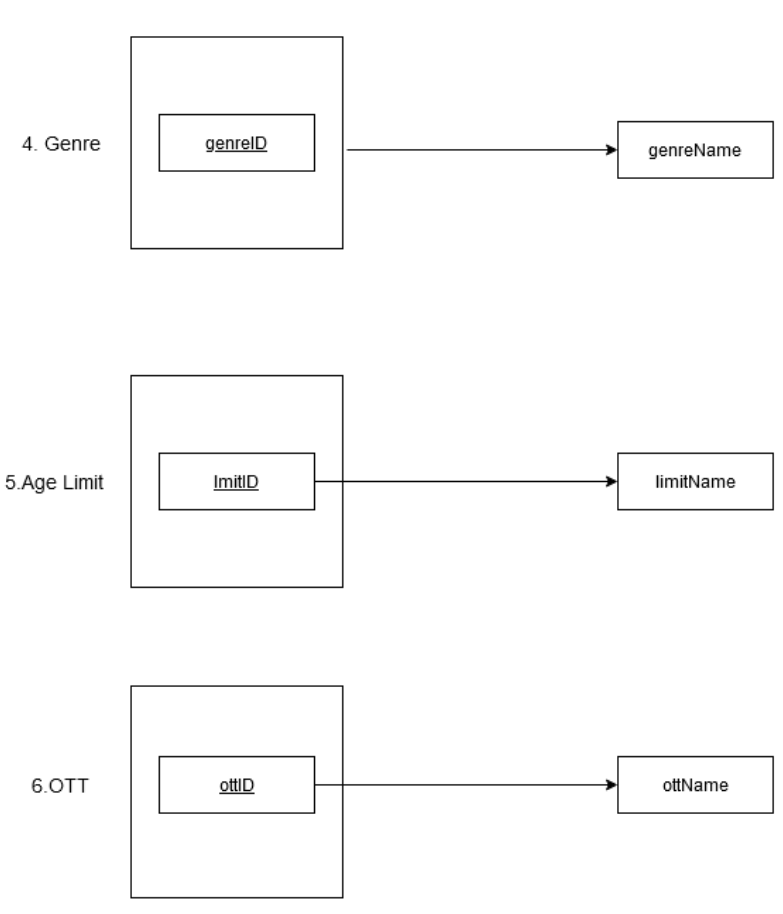


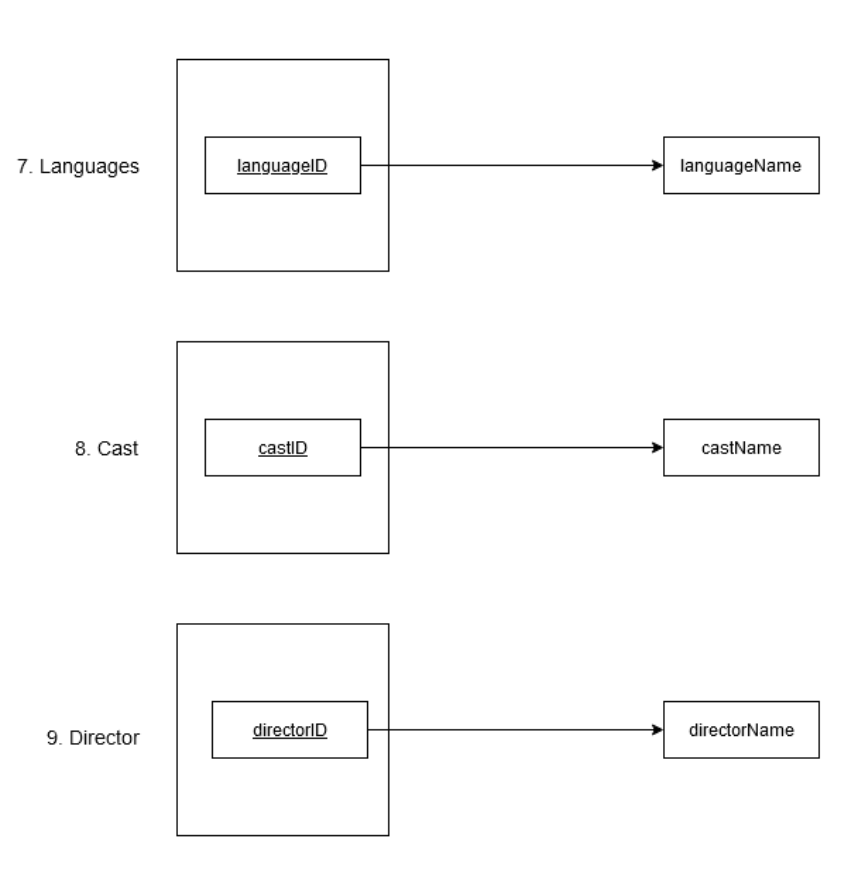


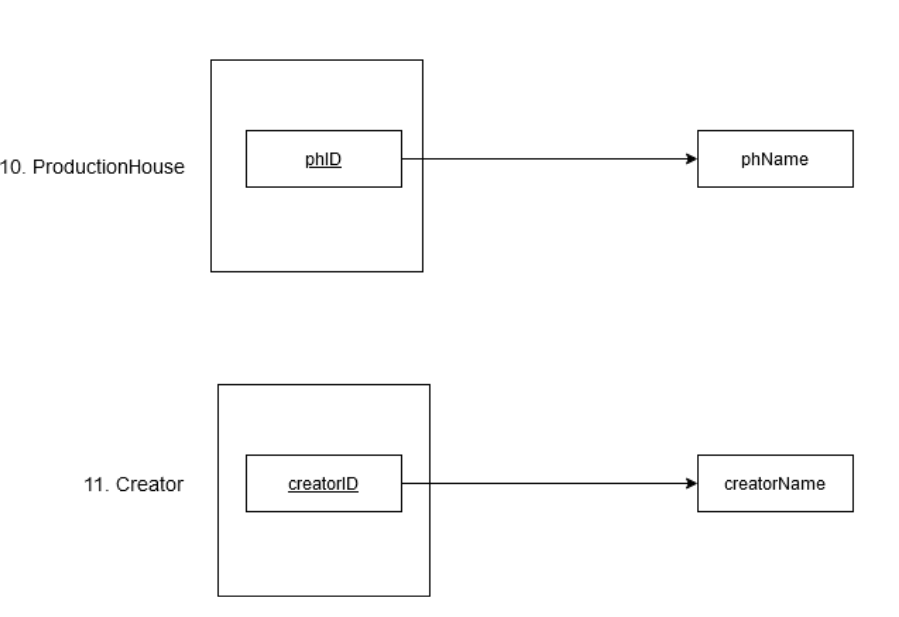
* 1. **Functional Dependency Chart**

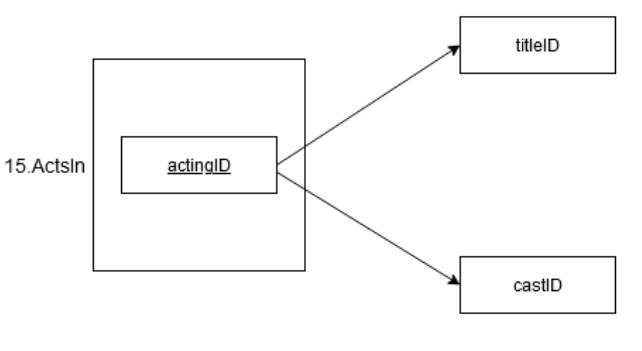


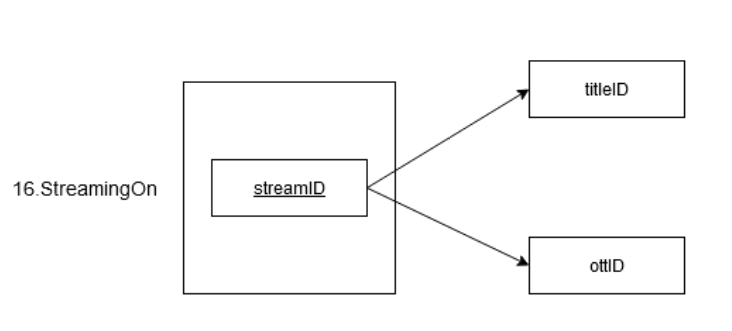


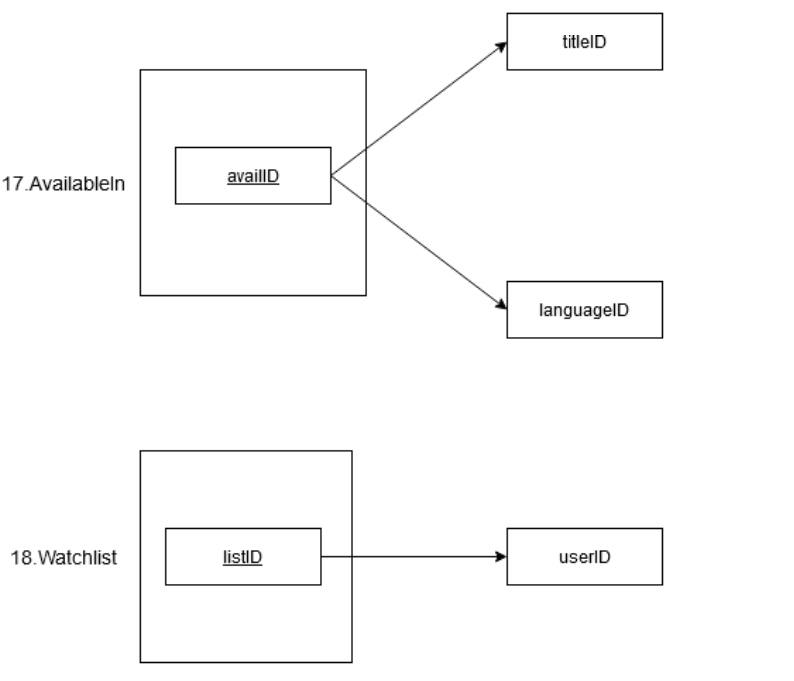


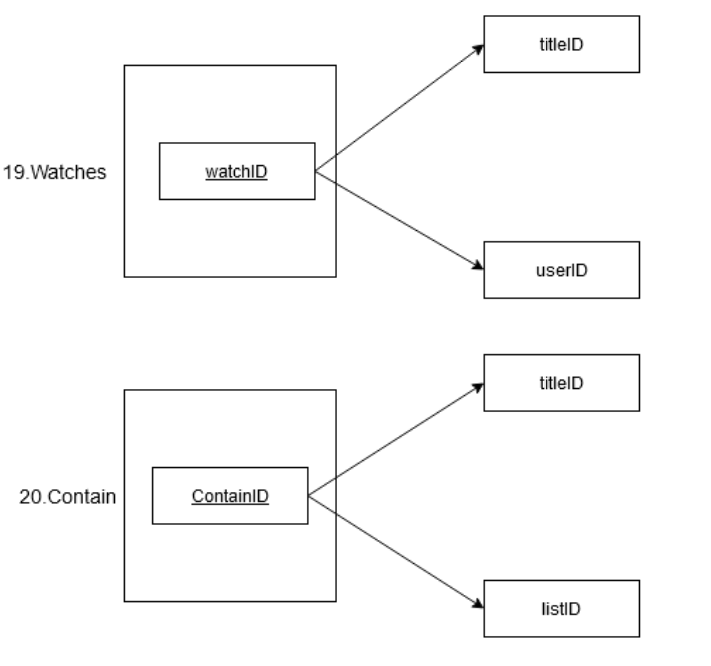








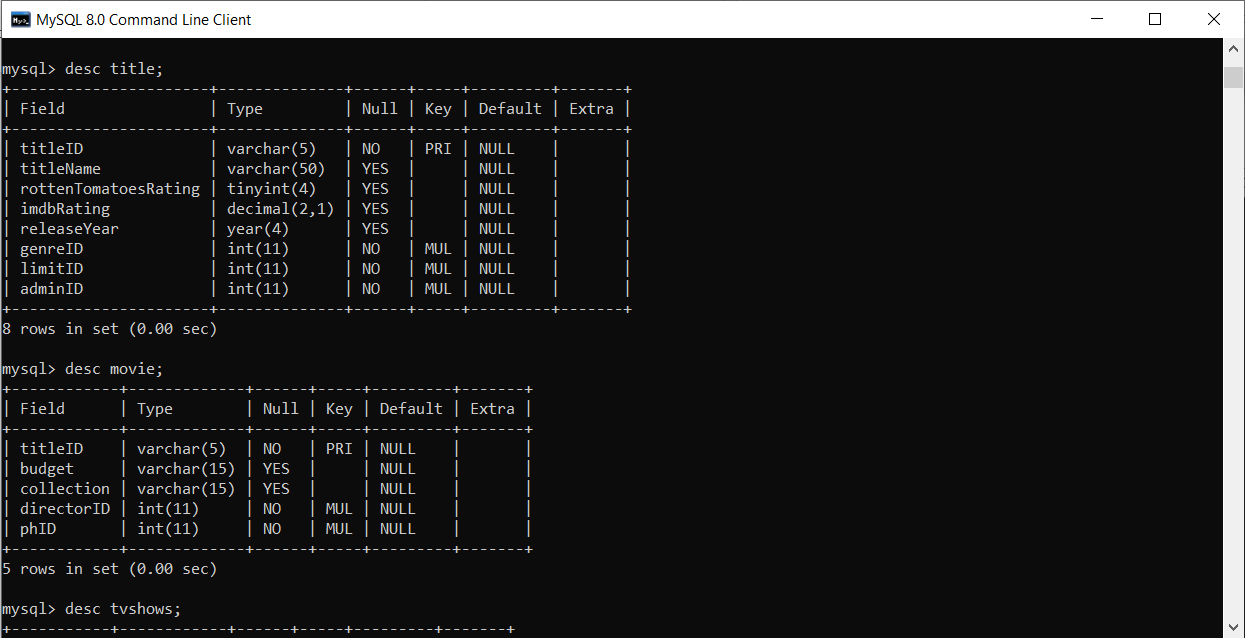


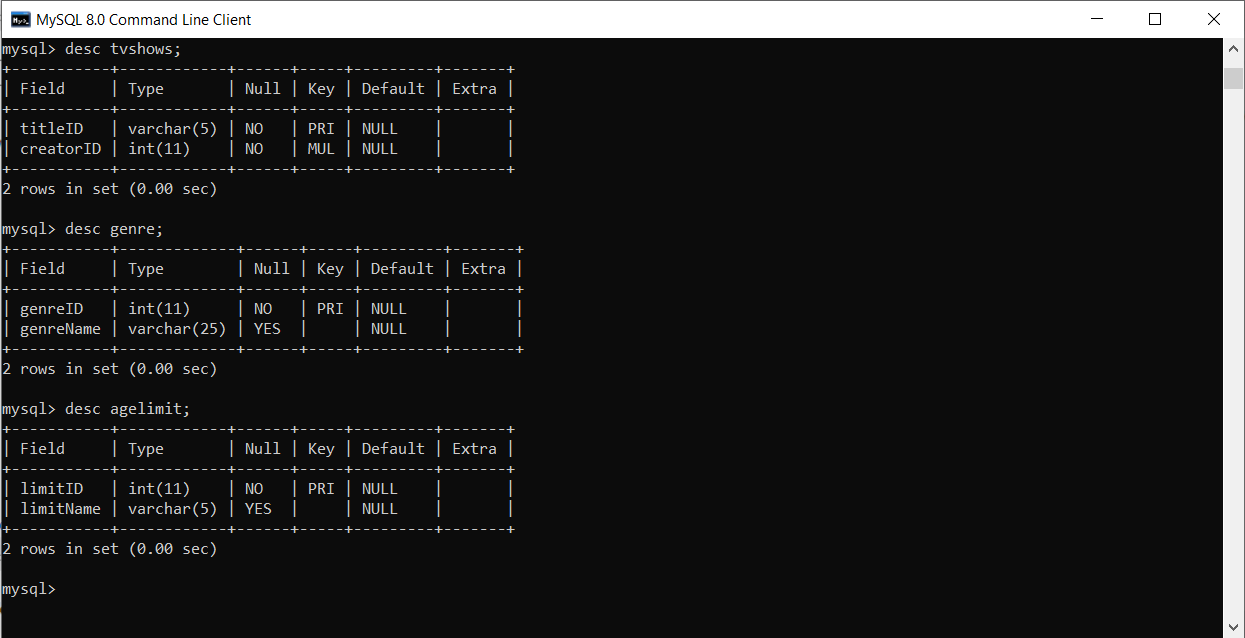


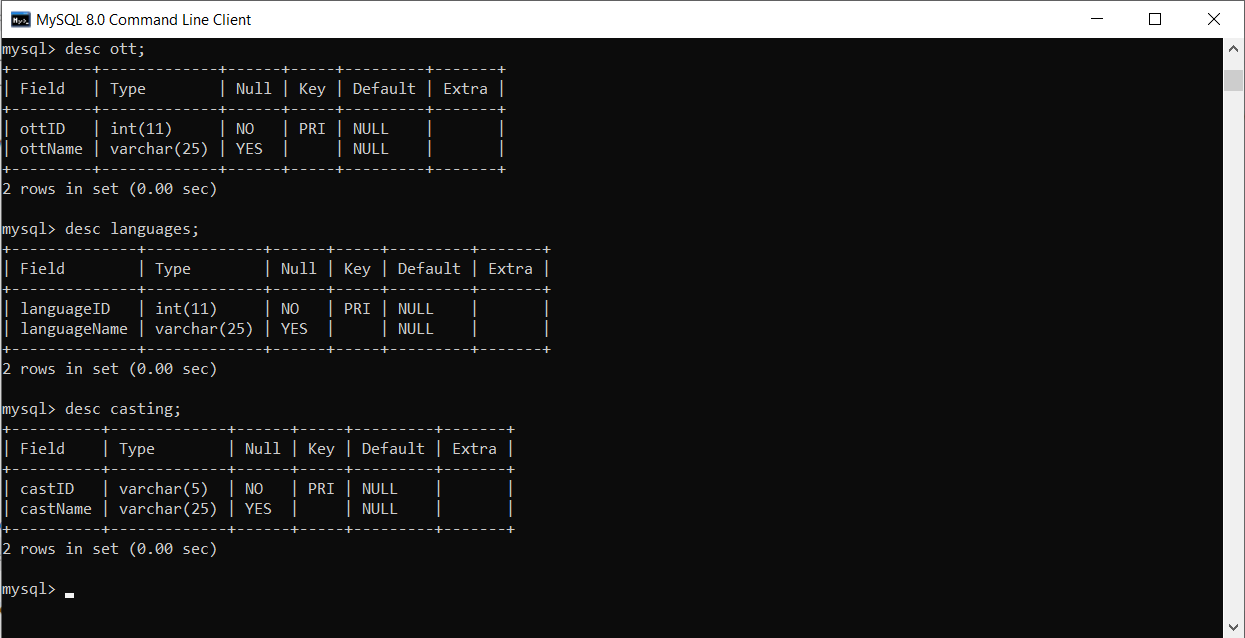
1. **Normalization**

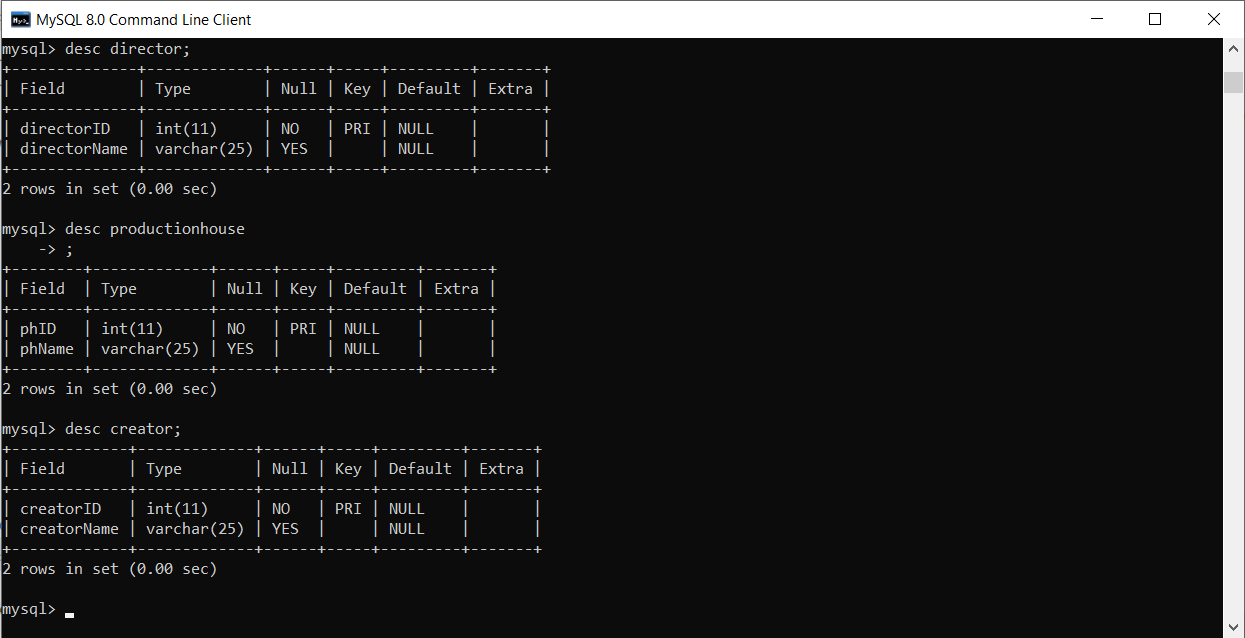
The database is in 3NF because there are no Transitive Dependencies.

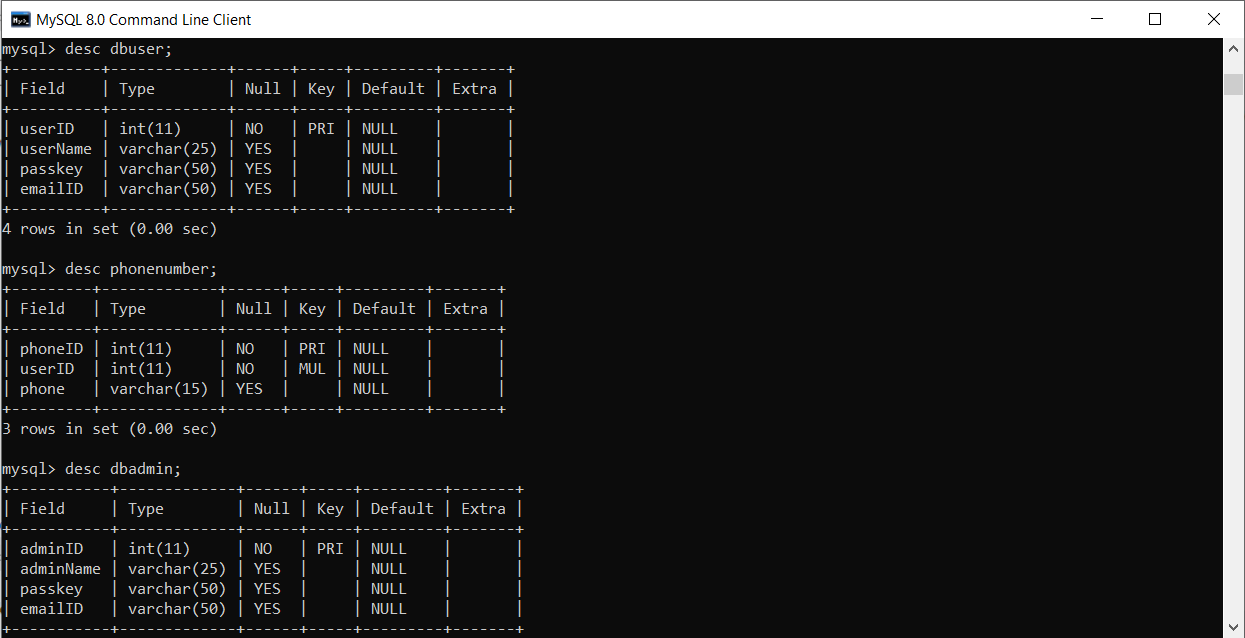
1. **Implementation**
   1. **Tables**

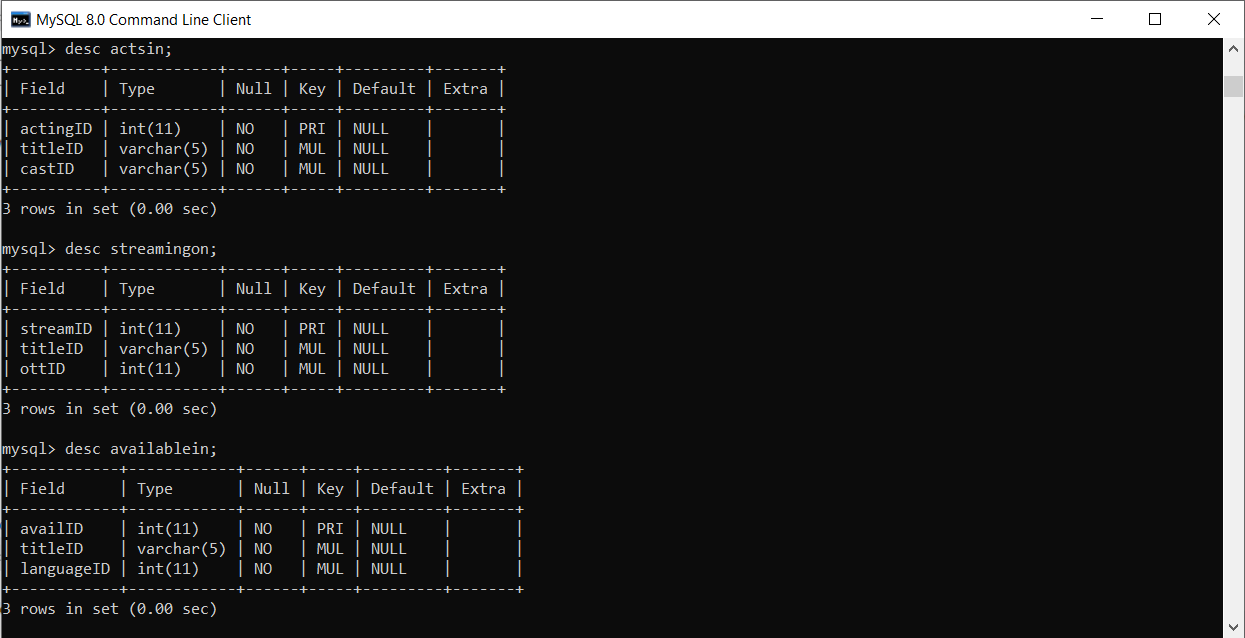
****

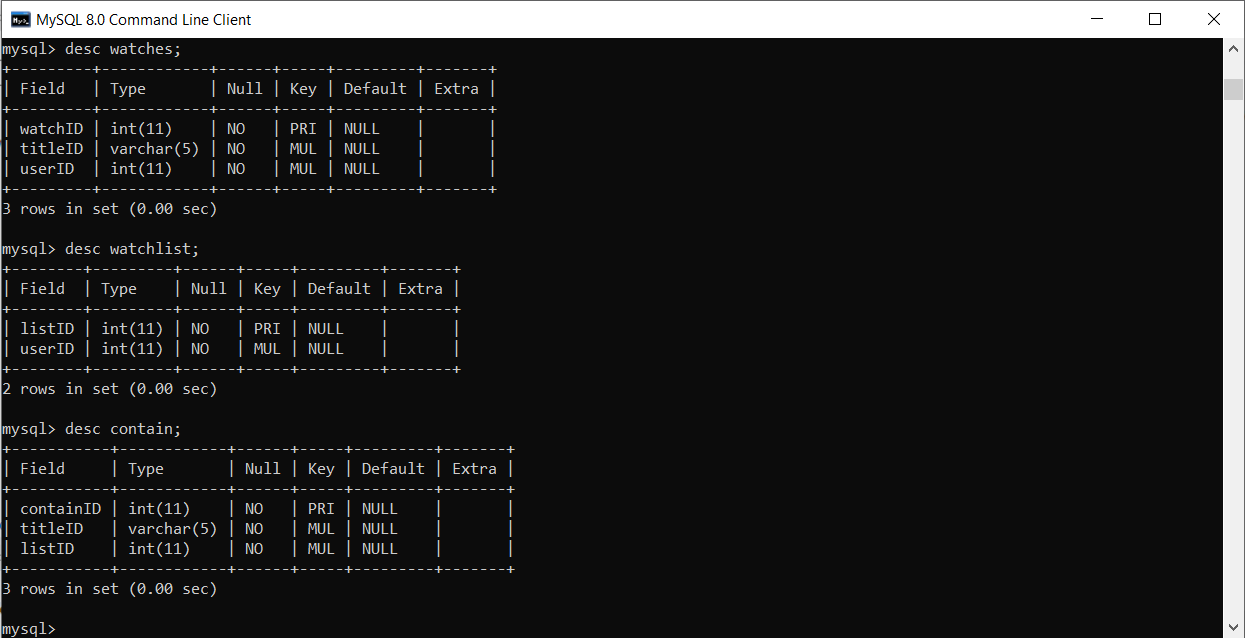
****

****

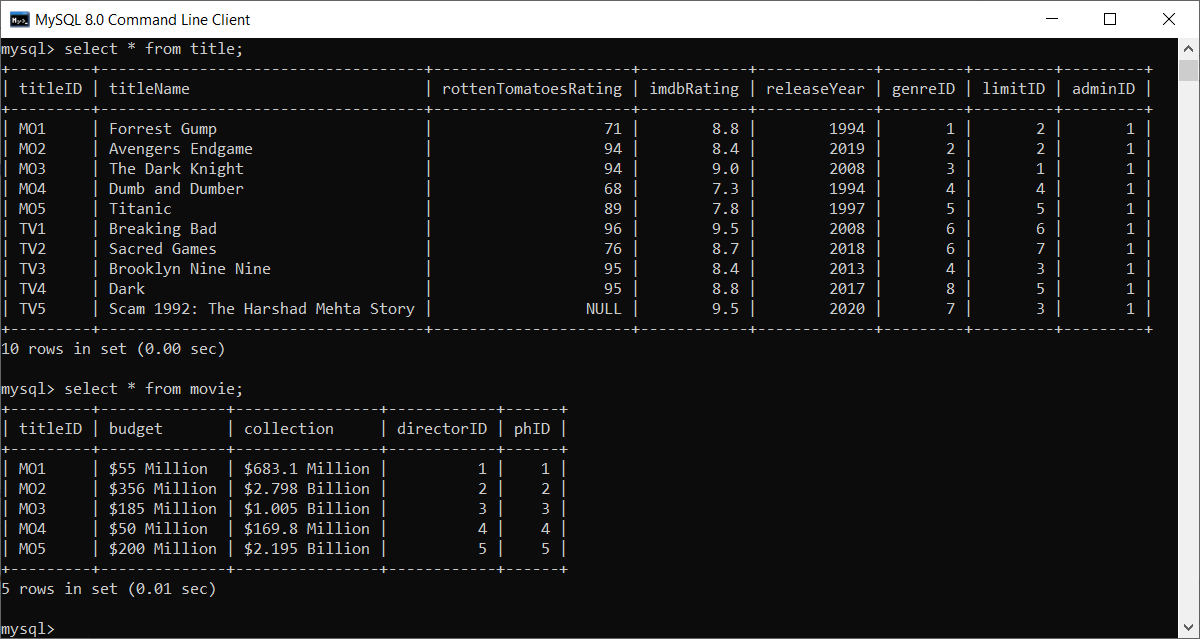
****

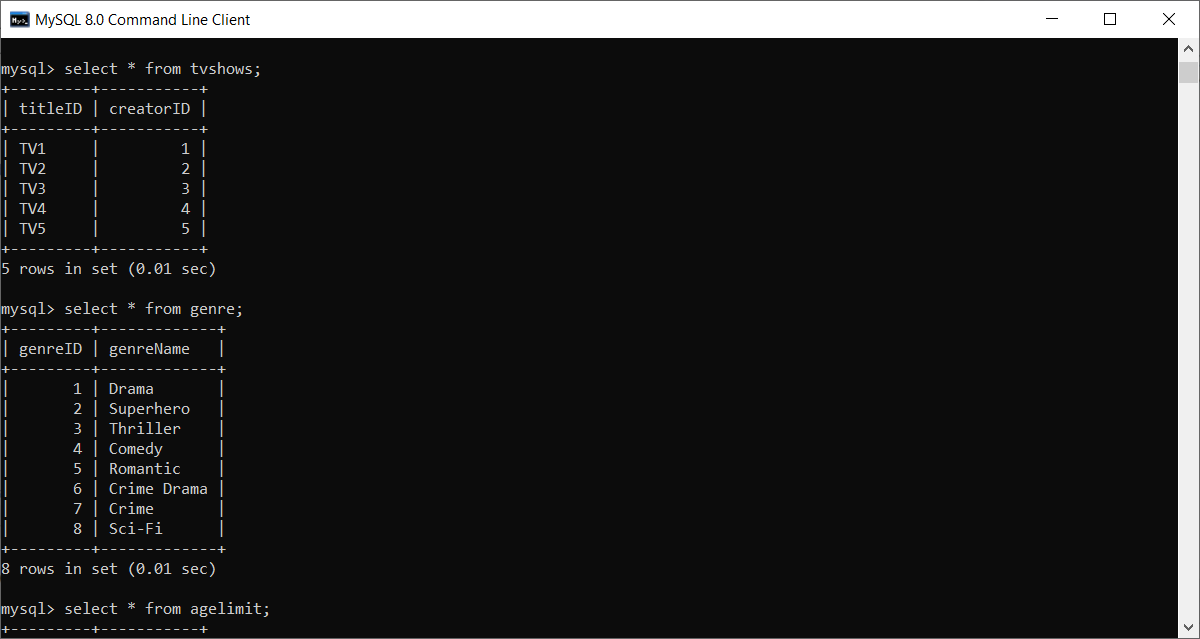
****

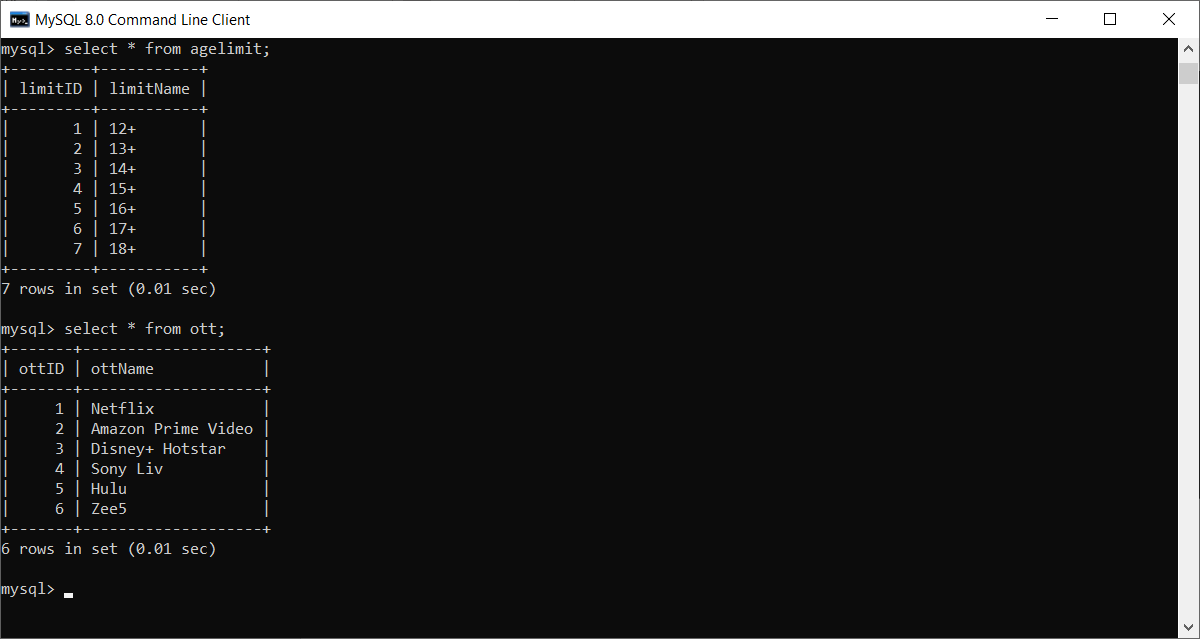
****

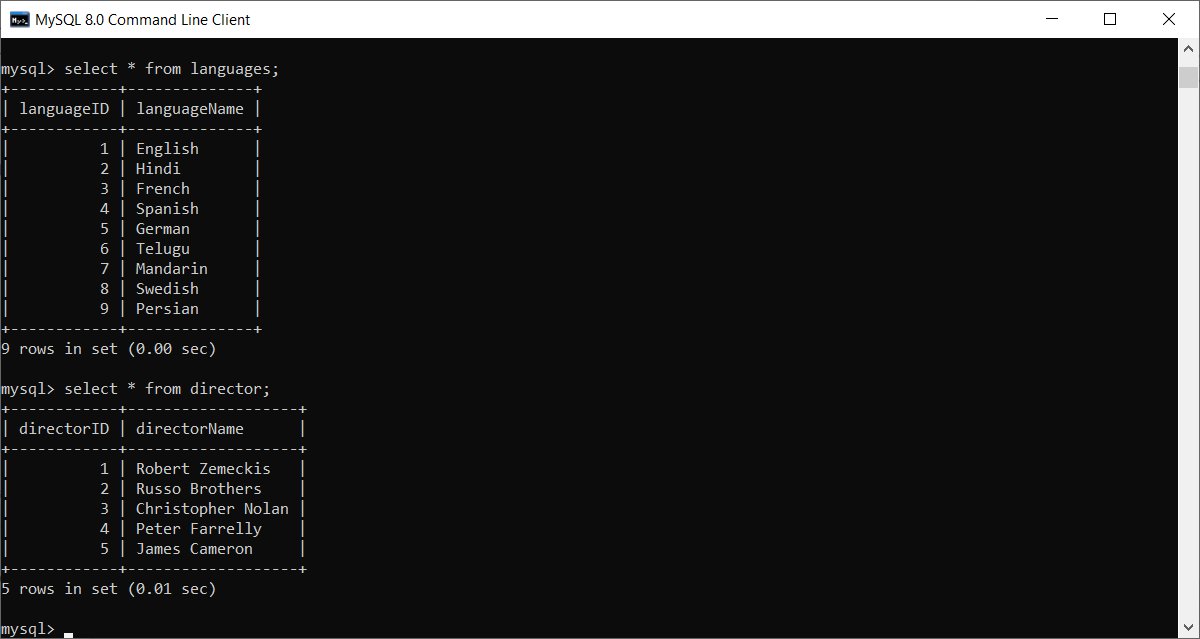
****

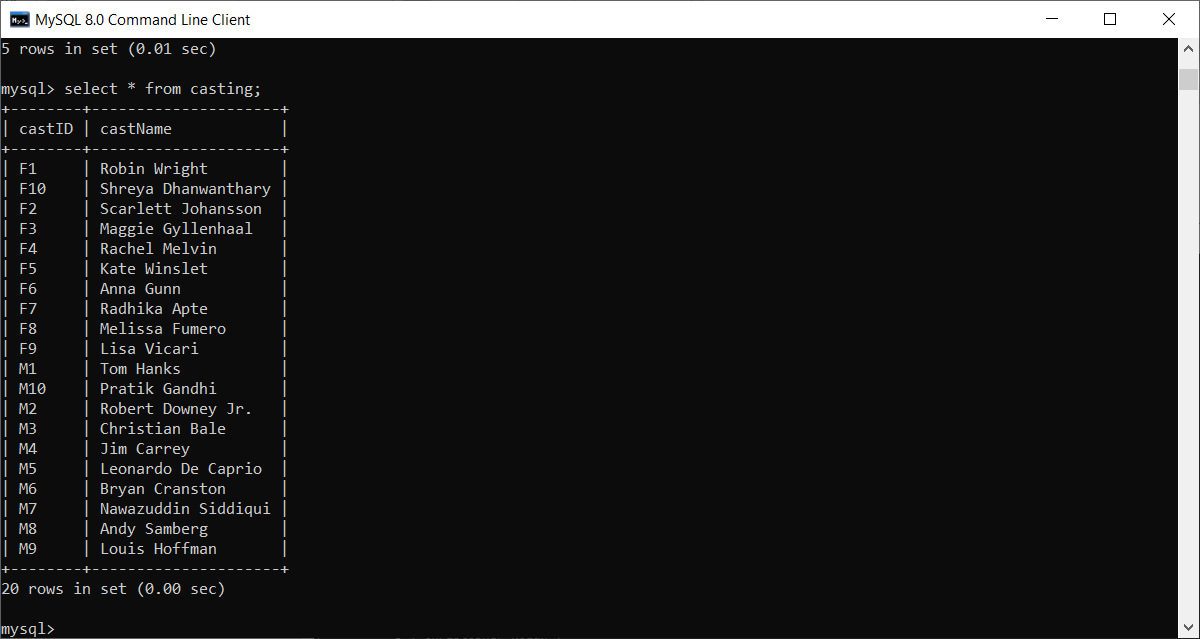
* 1. **Table Data**

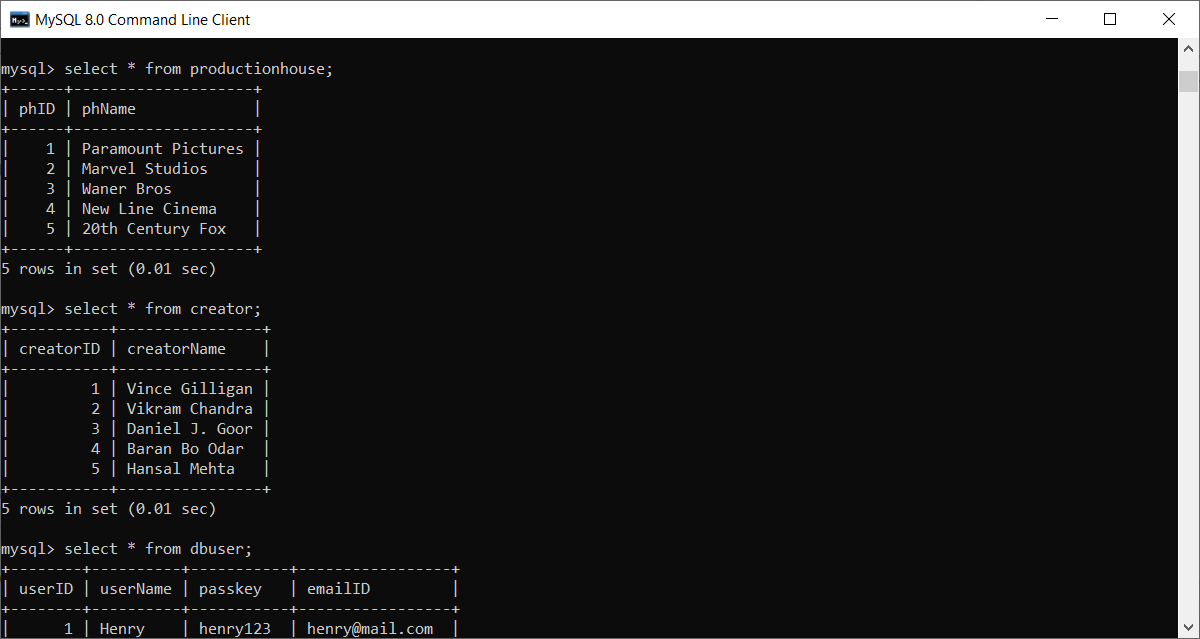
****

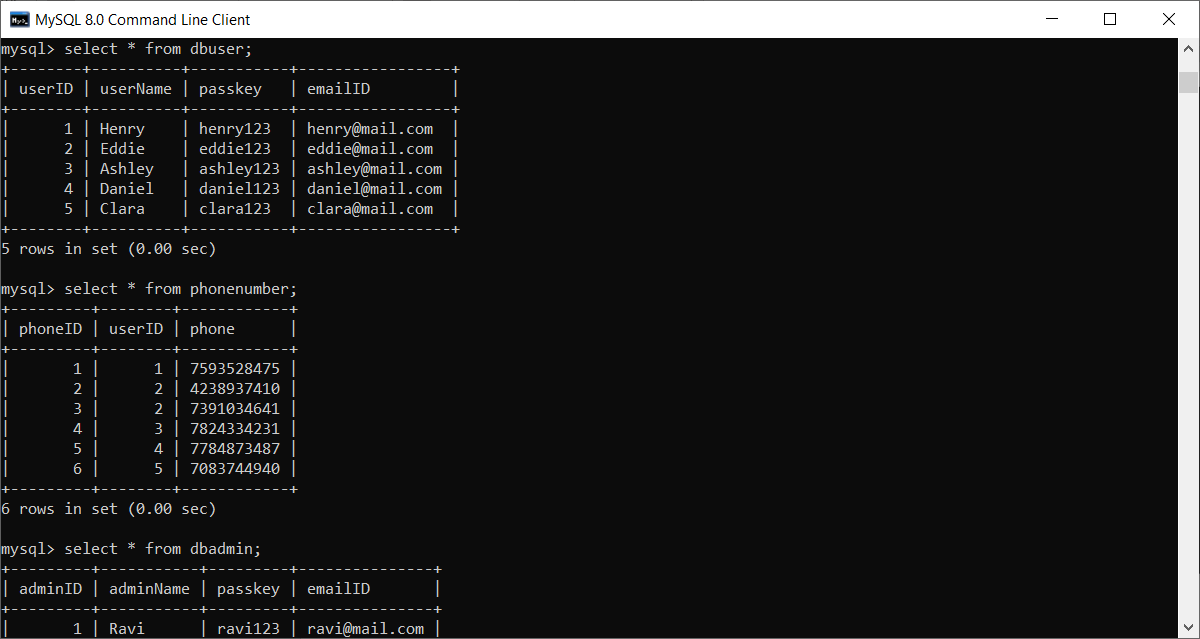
****

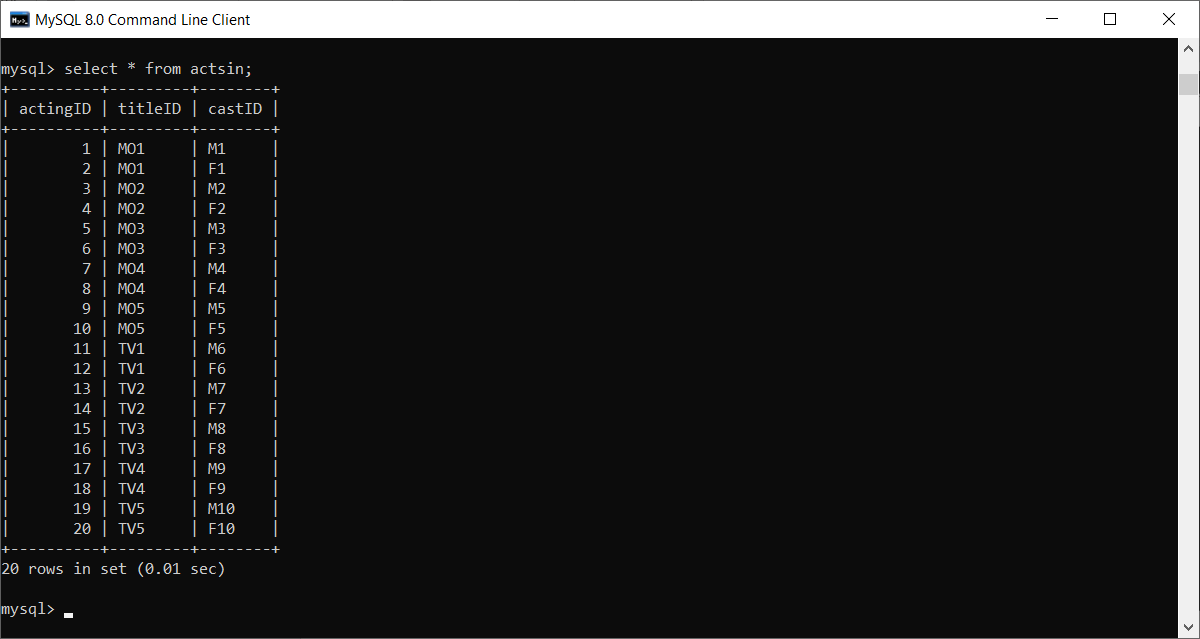
****

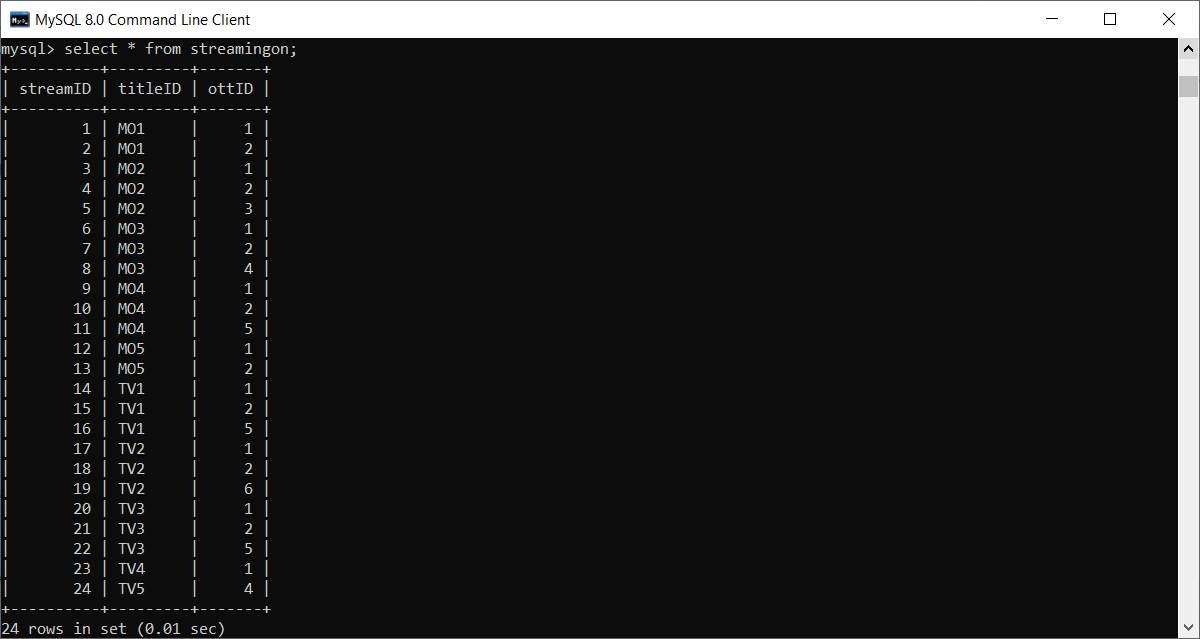
****

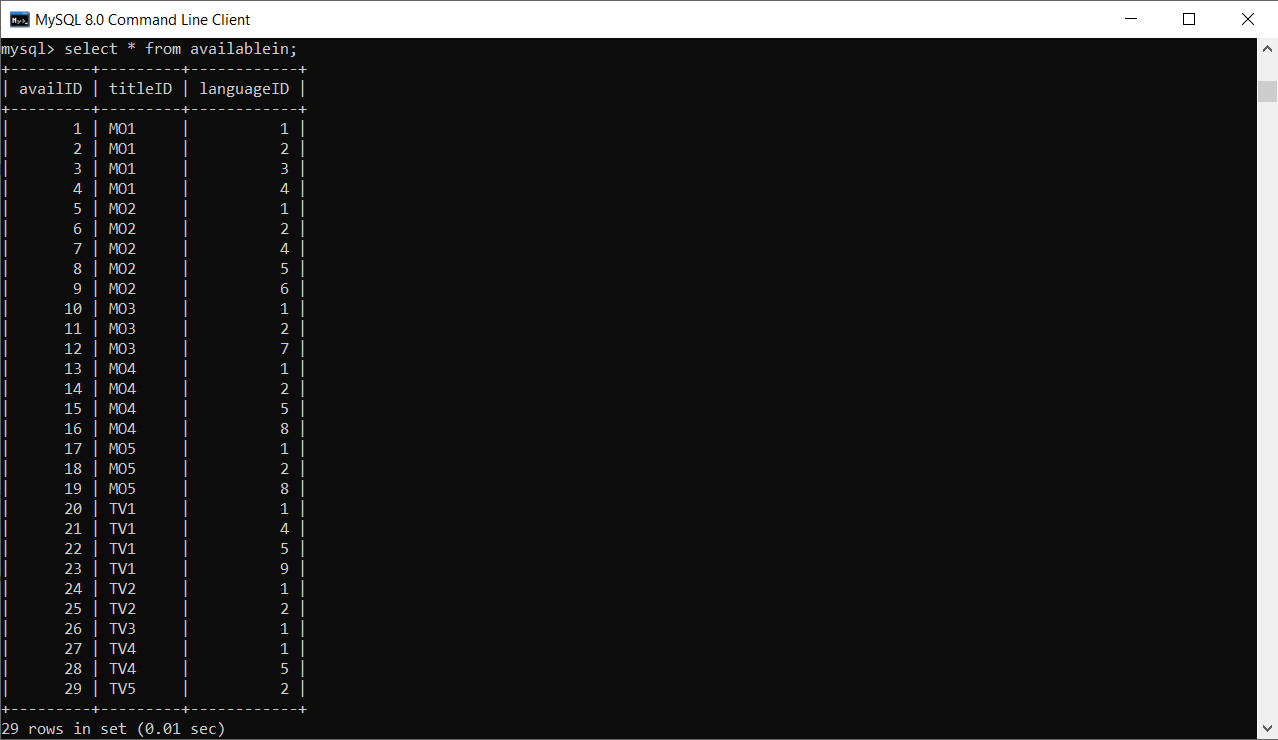
****

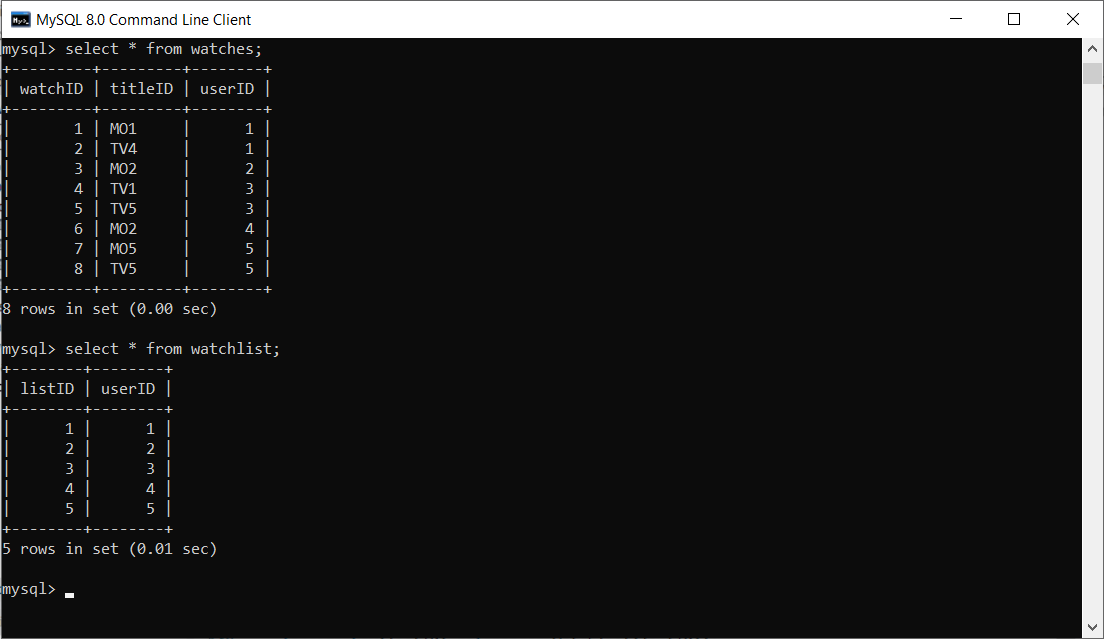
****

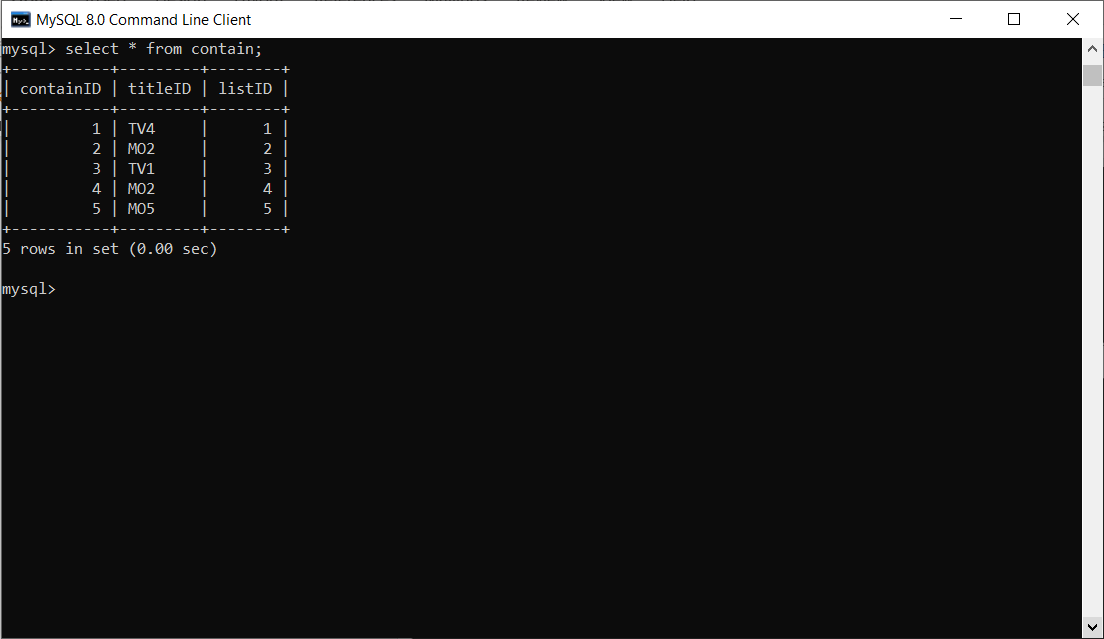
****

****

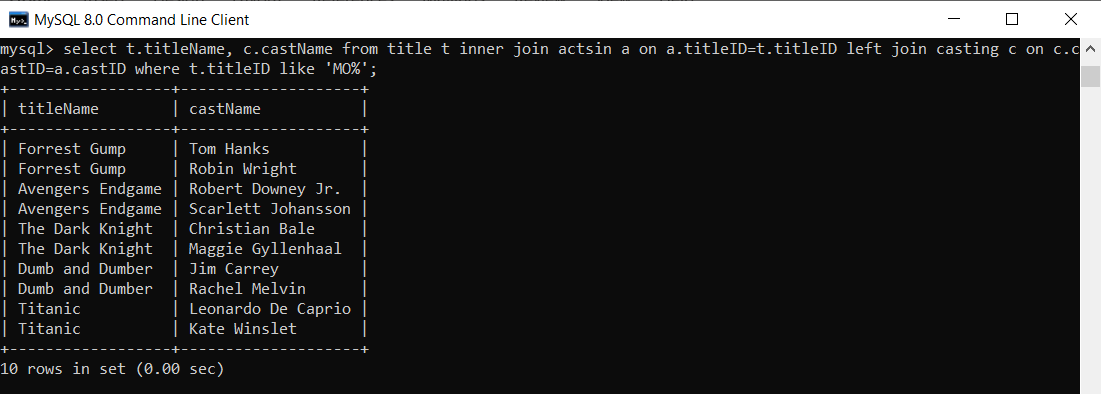
****

****

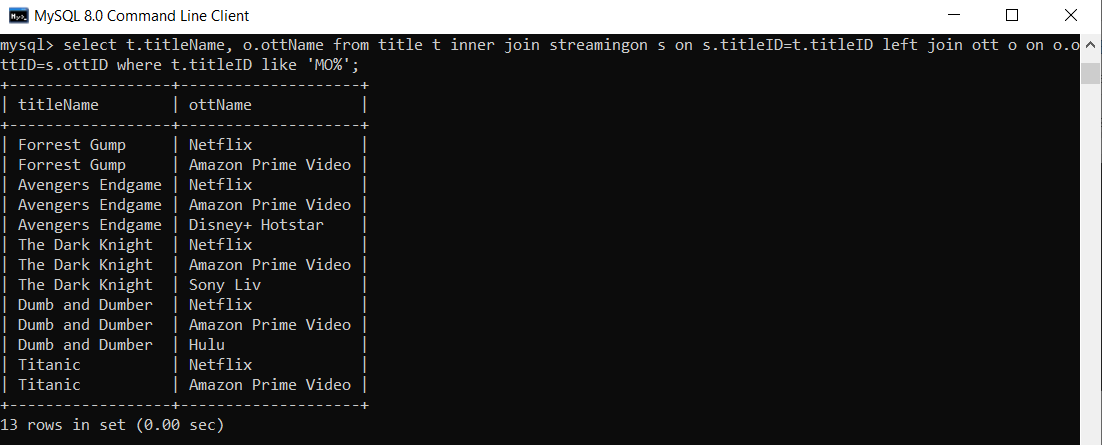
****

****

1. **25 Queries, 5 Functions, 5 Procedures, 5 Triggers**
   1. **25 Queries**
      1. Get the name of the cast for all the Movies.



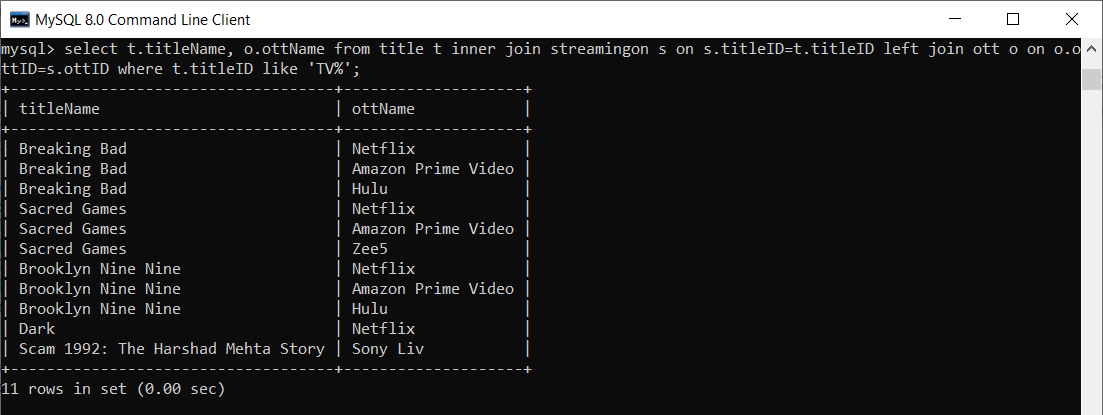
* + 1. Get the name of OTTs on which the Movies are available.



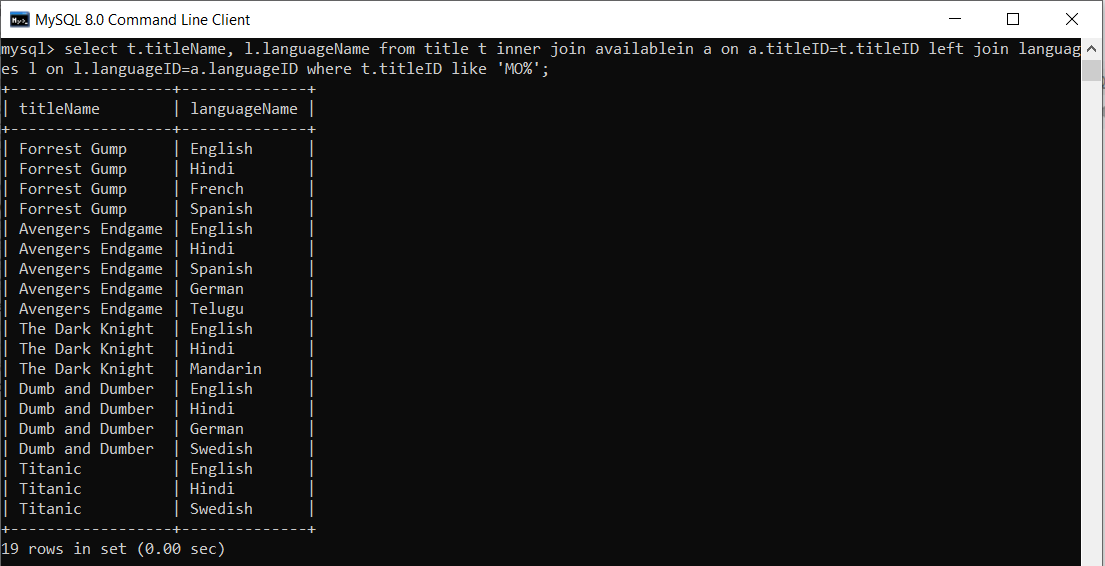
* + 1. Get the name of cast for all TV Shows.



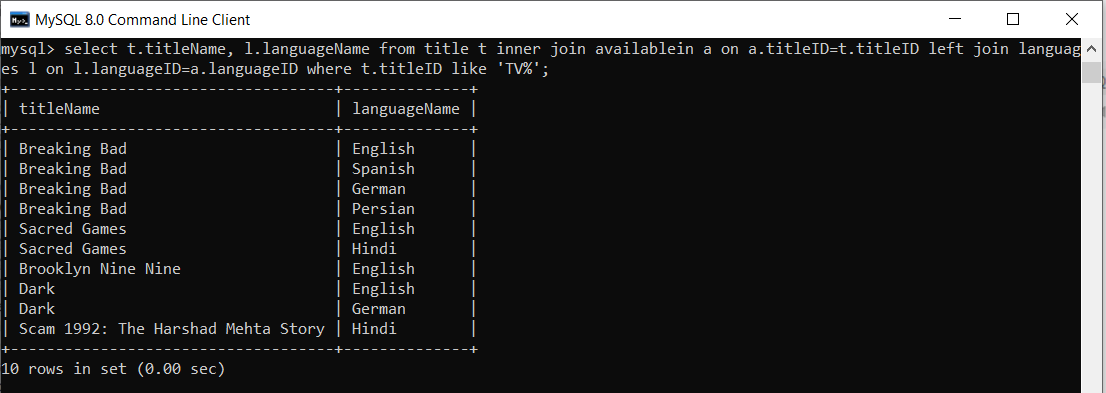
* + 1. Get the name of OTTs on which the TV Shows are available.



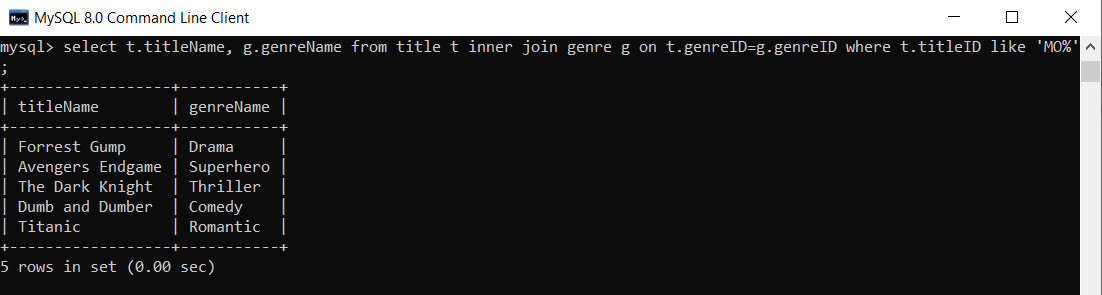
* + 1. Get the name of Languages in which Movies are available.



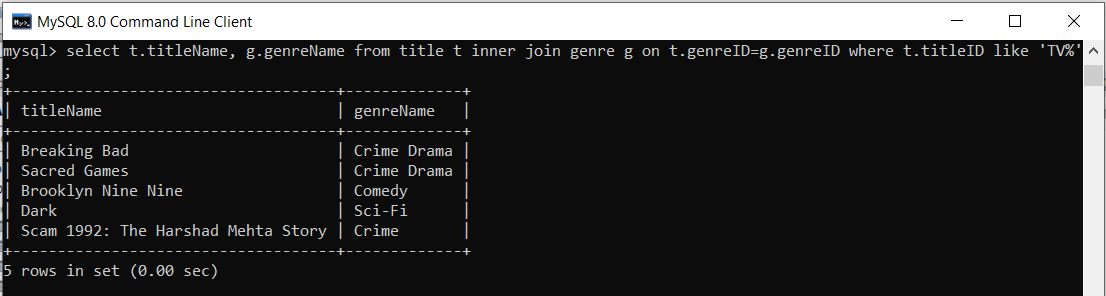
* + 1. Get the name of Languages in which TV Shows are available.



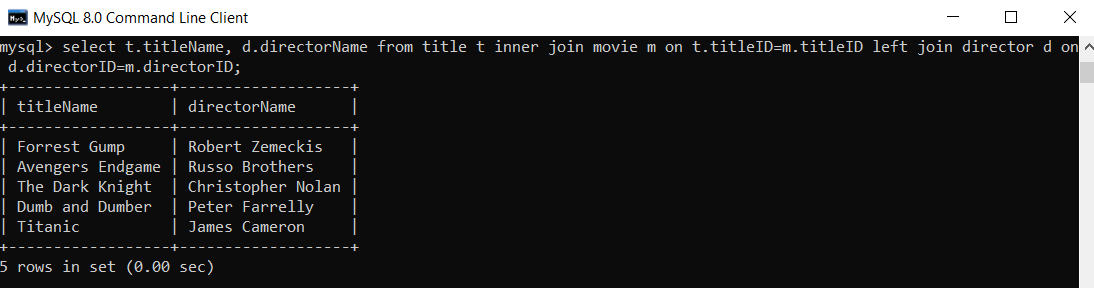
* + 1. Get the name of Genre to which the Movies belong.



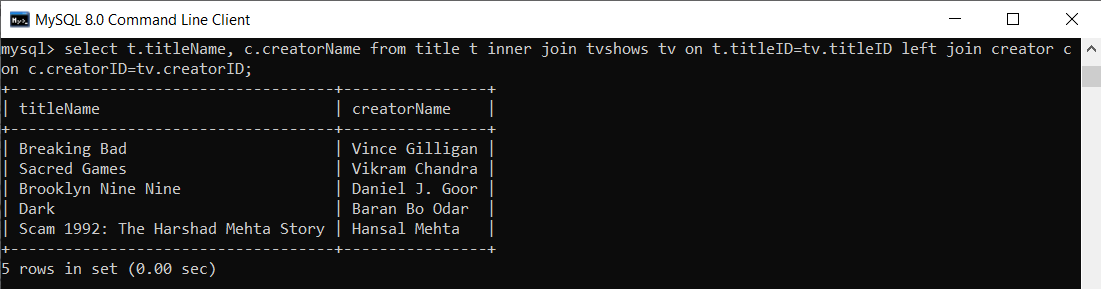
* + 1. Get the name of Genre to which the TV Shows belong.



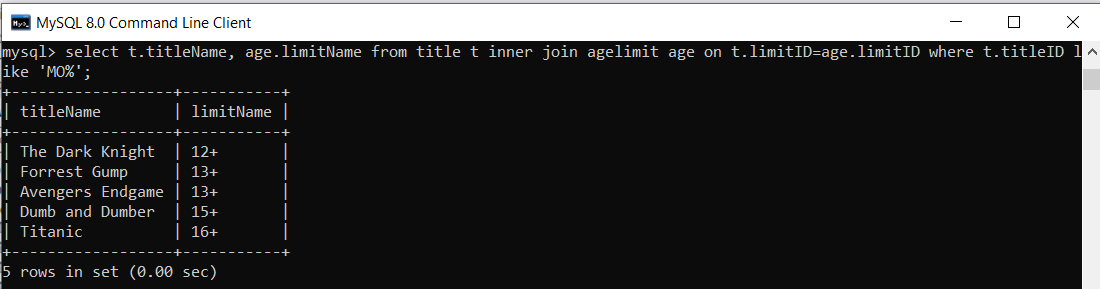
* + 1. Get the name of director of every Movie.



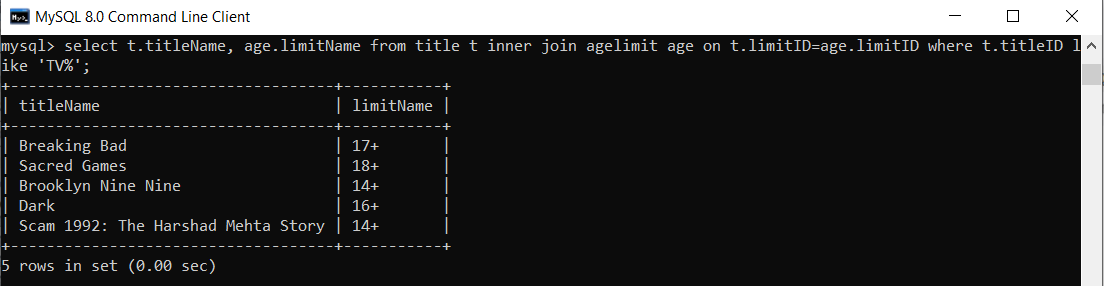
* + 1. Get the name of Creator of every TV Show.



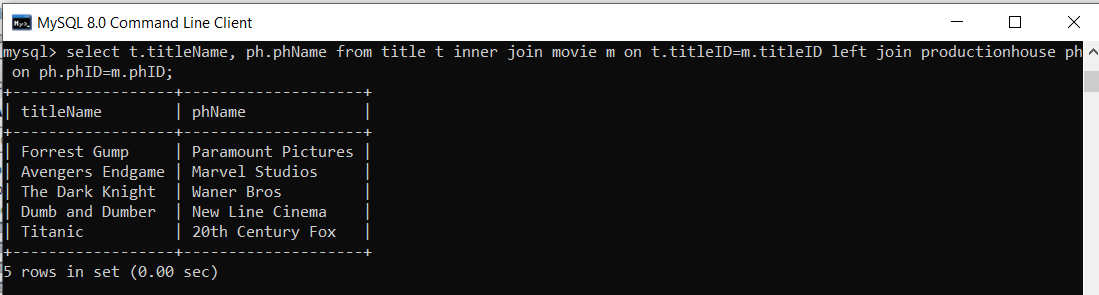
* + 1. Get the Age Limit for every Movie.



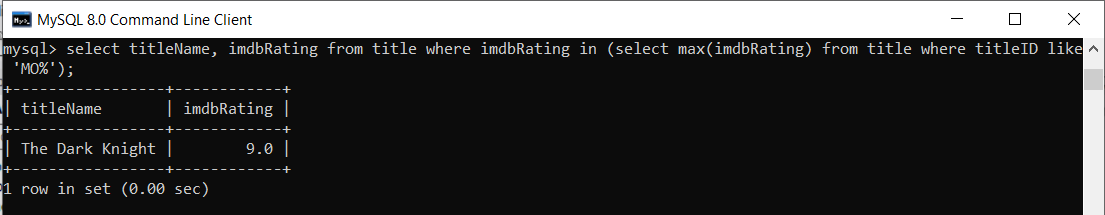
* + 1. Get the Age Limit for every TV Show.



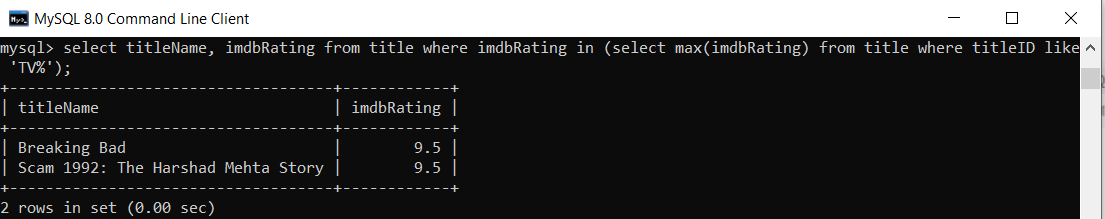
* + 1. Get the name of the Production House of every Movie.



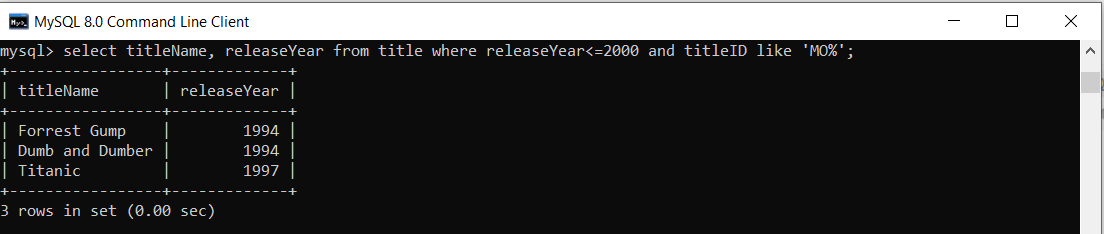
* + 1. Get the name of the Highest Rated (IMDB) Movie.



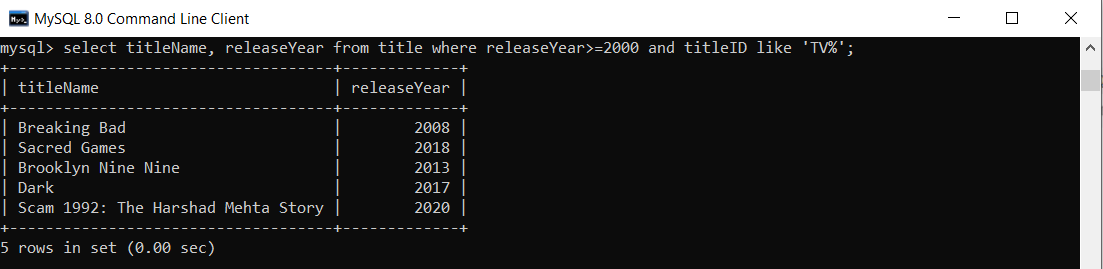
* + 1. Get the name of the Highest Rated (IMDB) TV Show.



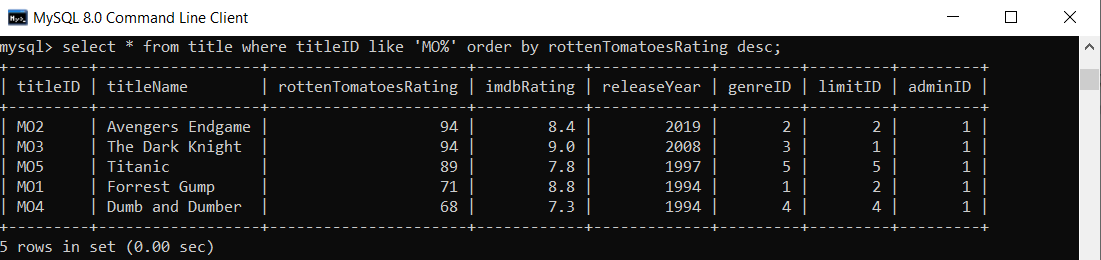
* + 1. Get the names of Movies released before 2000.



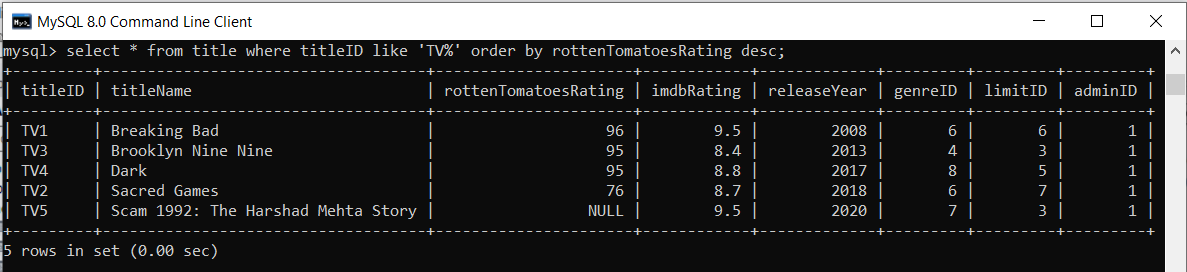
* + 1. Get the names of TV Shows released after 2000.



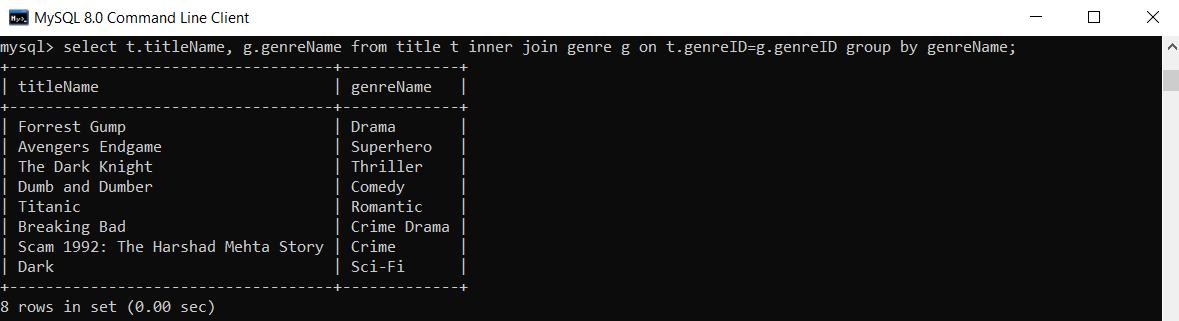
* + 1. Order the Movies by Rotten Tomatoes Rating.



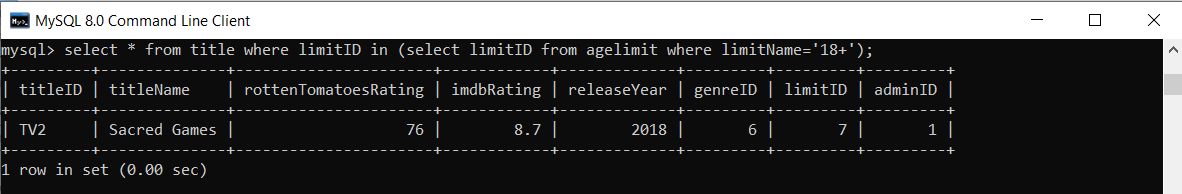
* + 1. Order the TV Shows by Rotten Tomatoes Rating.



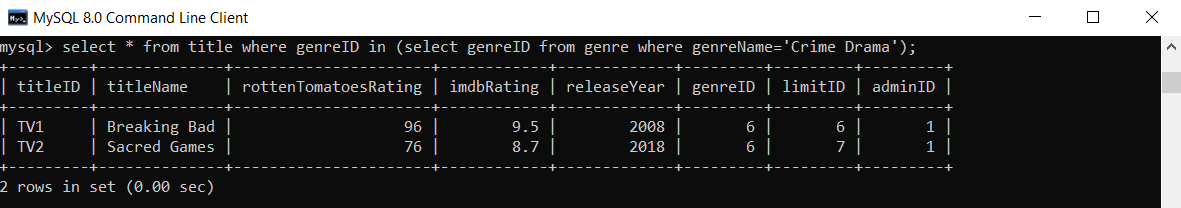
* + 1. Group the Titles according to their Genre.



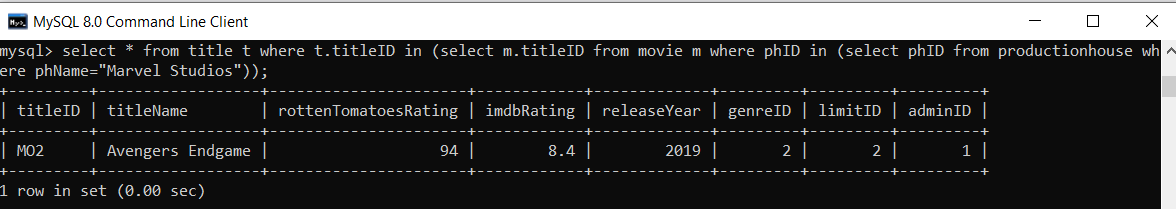
* + 1. Get all the Titles that have an Age Limit of ‘18+’.



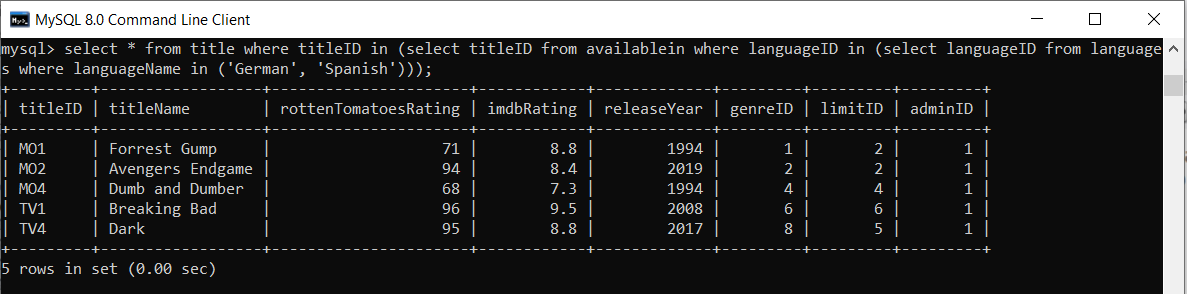
* + 1. Get all the Titles whose Genre is ‘Crime Drama’.



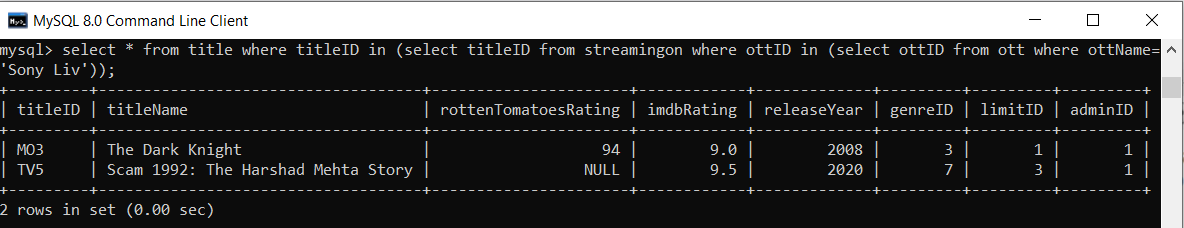
* + 1. Get all the Movies produced by ‘Marvel Studios’.



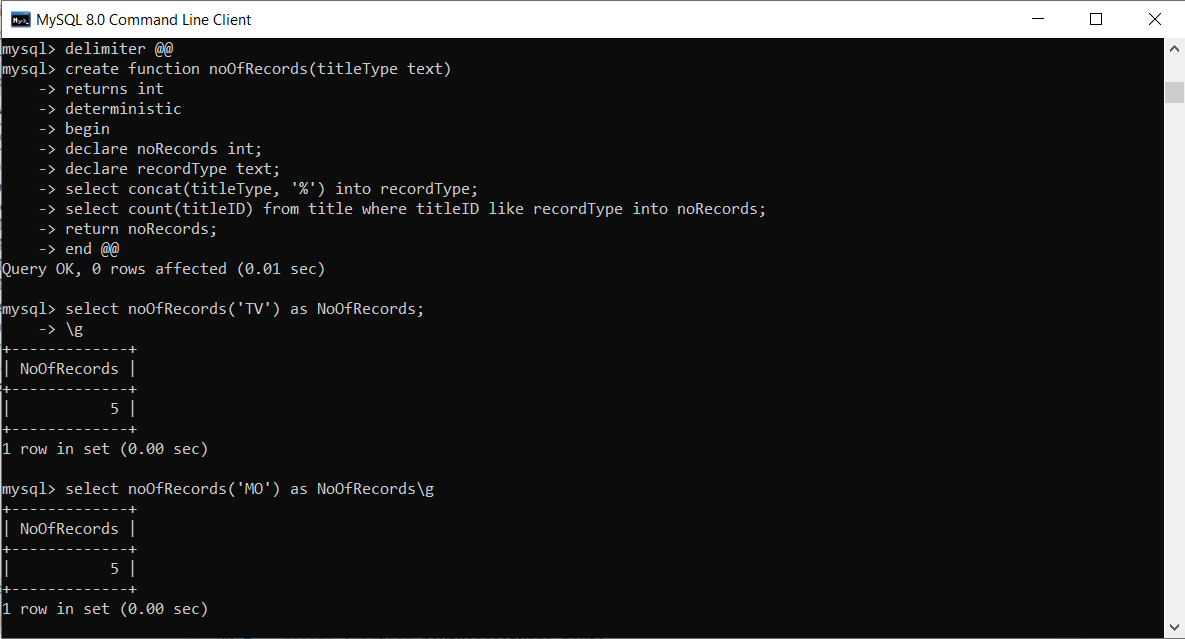
* + 1. Get all the Titles which are available in ‘German’ or ‘Spanish’.



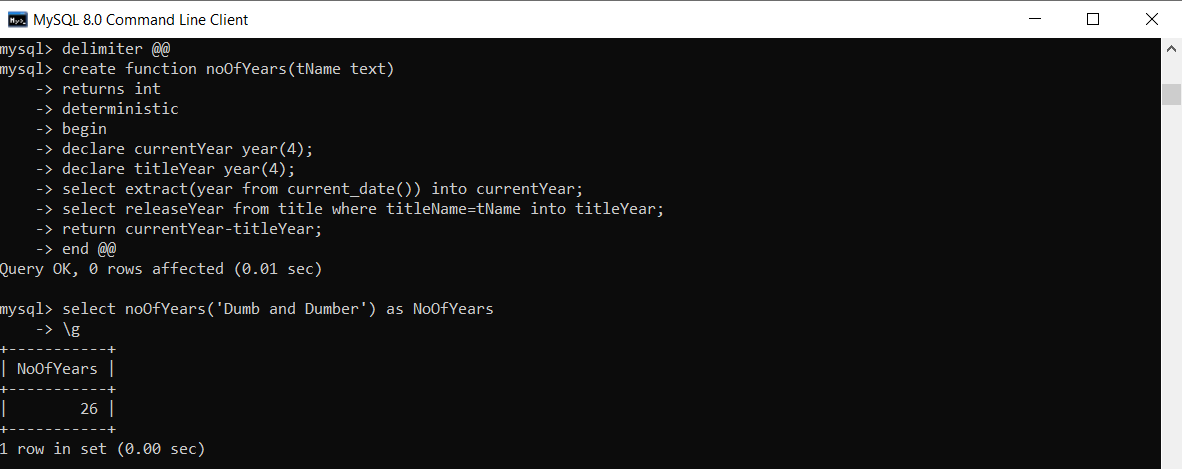
* + 1. Get all the Titles streaming on ‘Sony Liv’.



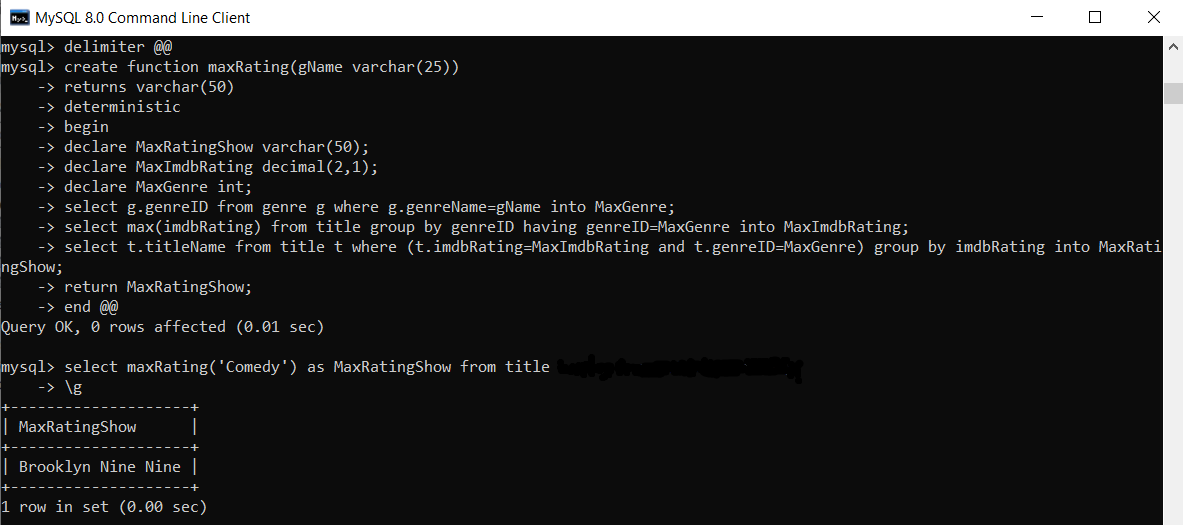
* 1. **5 Functions**
     1. Function to get the number of Movies/TV Shows that are present in the database.

****

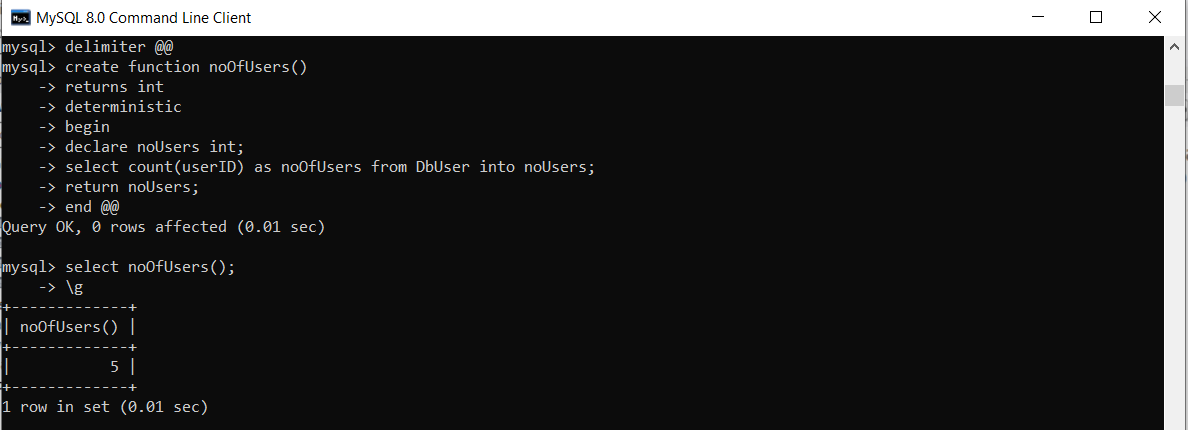
* + 1. Function to get the number of years that have passed since a particular title was released.



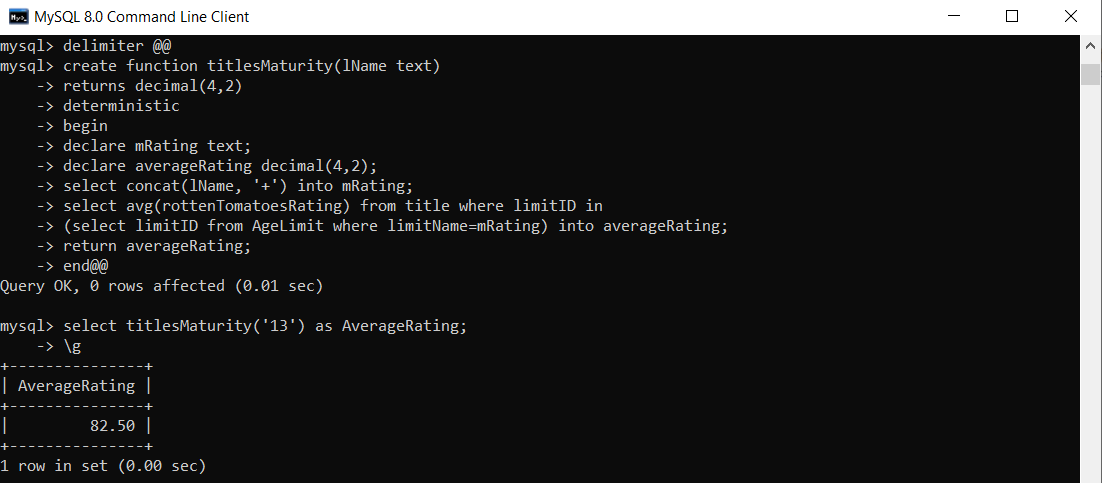
* + 1. Function to get the highest rated title (IMDB) of a particular genre.

****

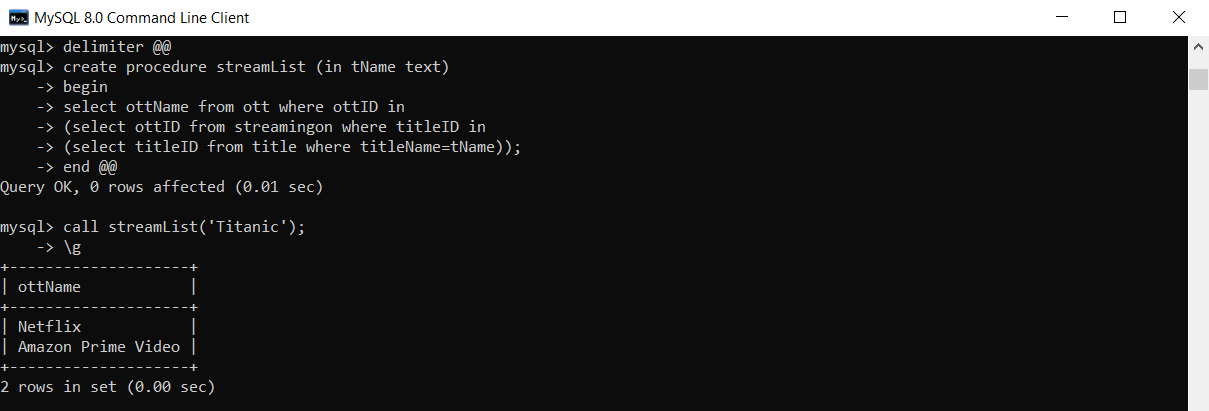
* + 1. Function to get the number of users in the database.



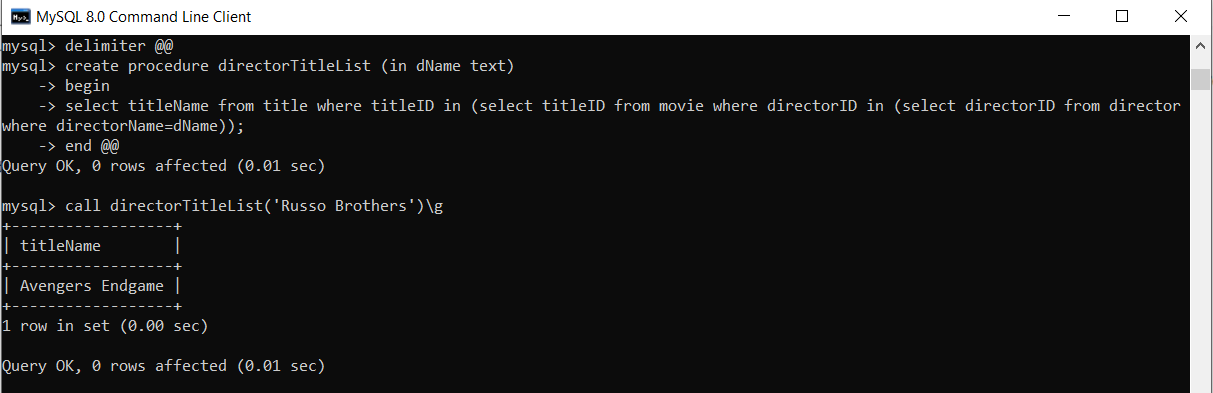
* + 1. Function to get the average Rotten Tomatoes Rating of titles of a particular Maturity Rating.



* 1. **5 Procedures**
     1. Procedure to get the names of OTTs on which a particular title is available.



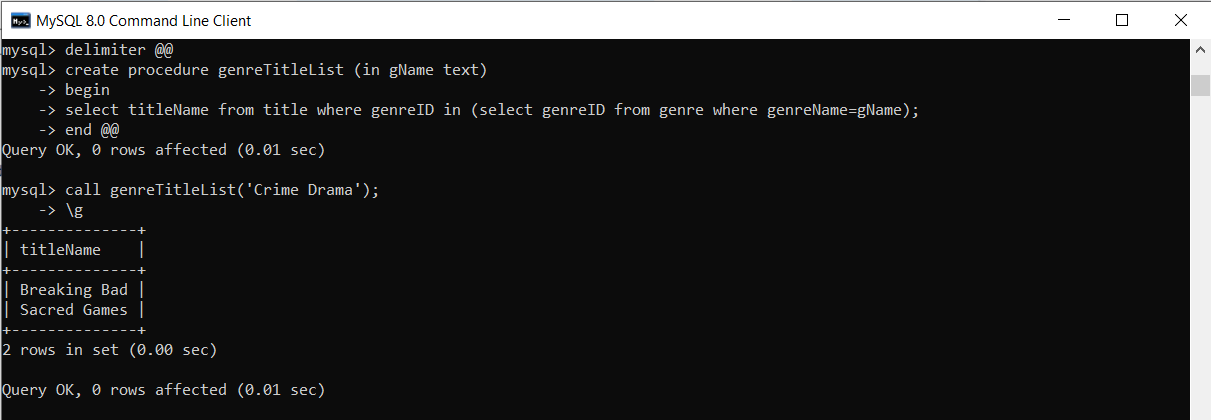
* + 1. Procedure to get the names of the Titles directed by a particular Director.



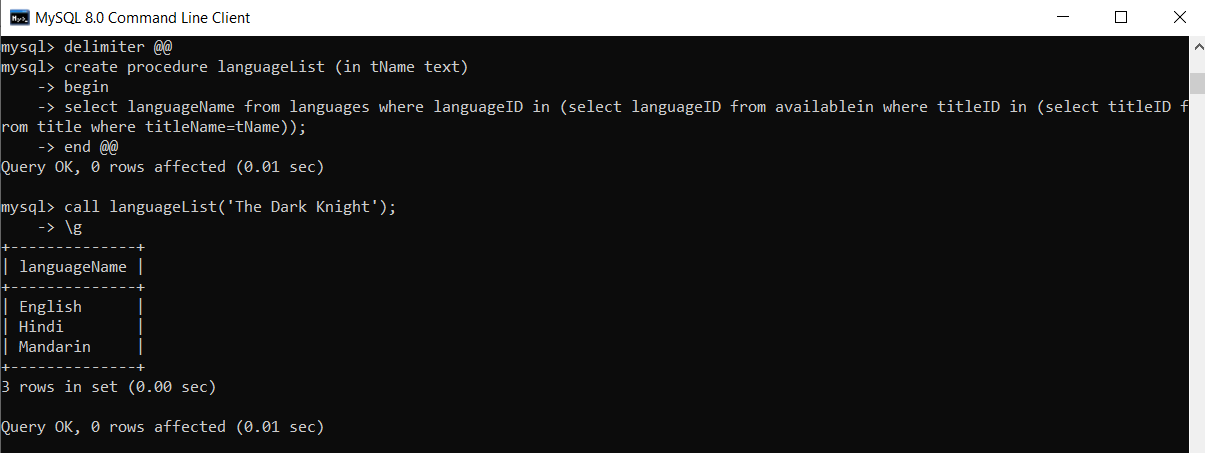
* + 1. Procedure to get the name of Actress of a particular Title.



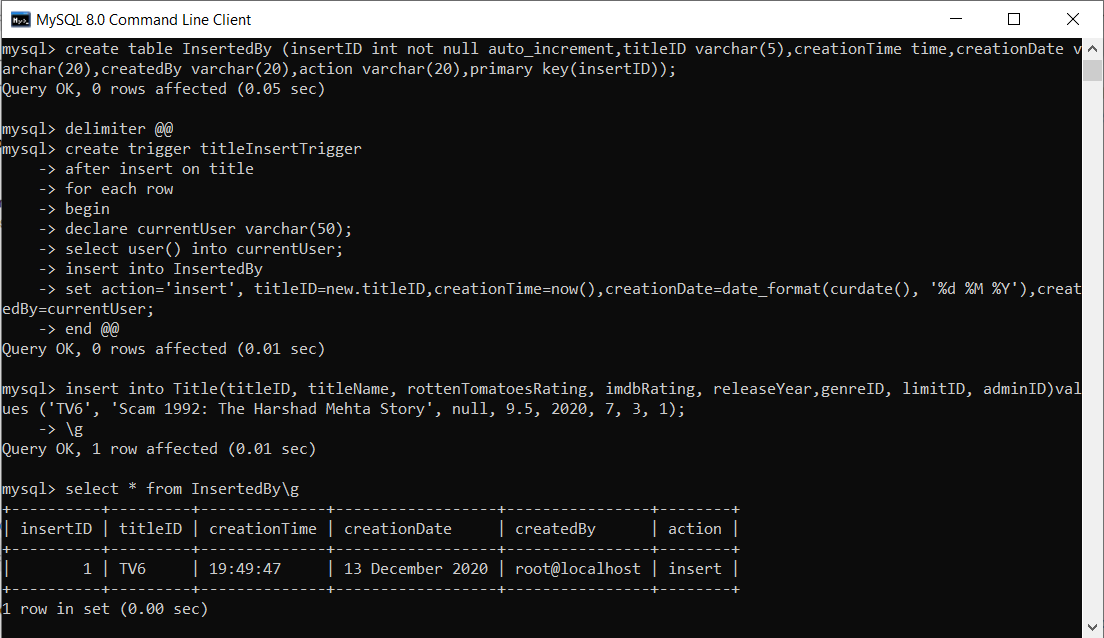
* + 1. Procedure to get the name of Titles belonging to a particular Genre.



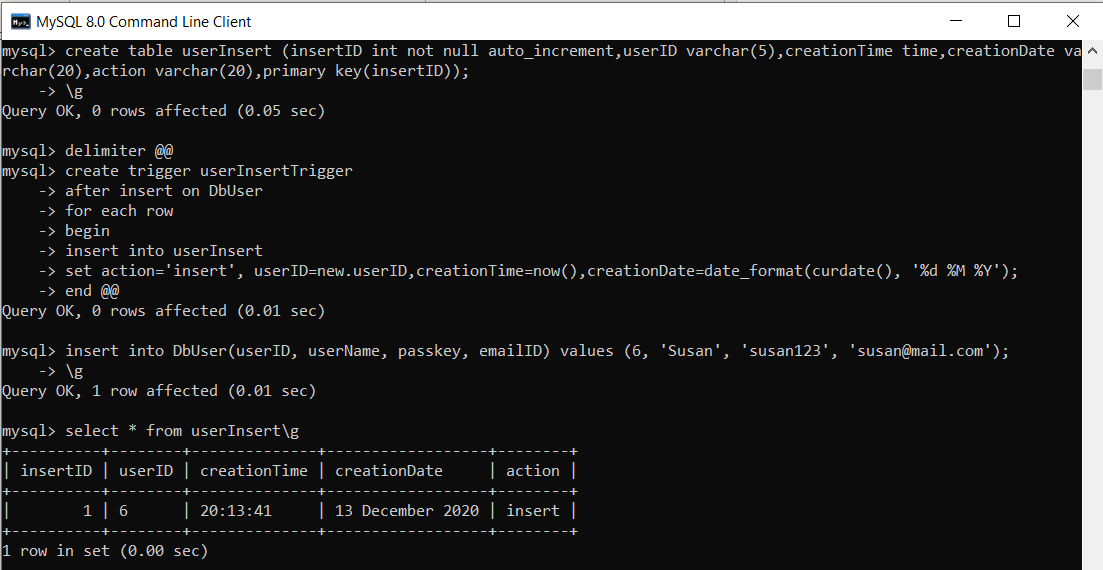
* + 1. Procedure to get the languages in which the title is available.



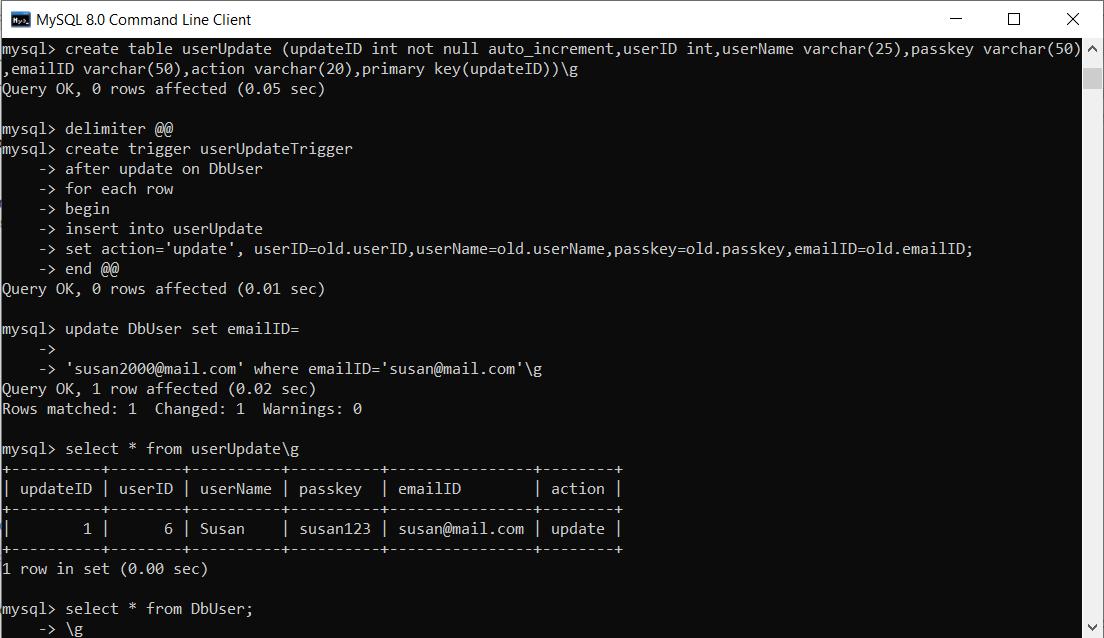
* 1. **5 Triggers**
     1. Trigger to store the time at which a Title was inserted into the ‘title’ table.

****

* + 1. Trigger to store the time and date at which user information was inserted into the database.

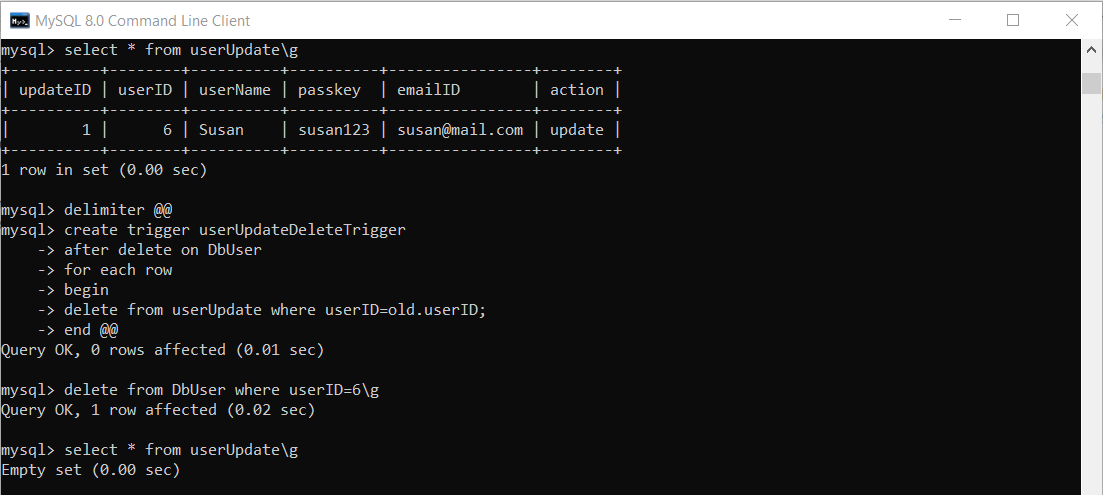


* + 1. Trigger to store old user data when user updates his/her data.

****



* + 1. Trigger to delete the old user data (which gets stored when user data is updated) if the user gets deleted



* + 1. Trigger to store the time and date at which the user information was deleted from the database.

