PROJECT REPORT

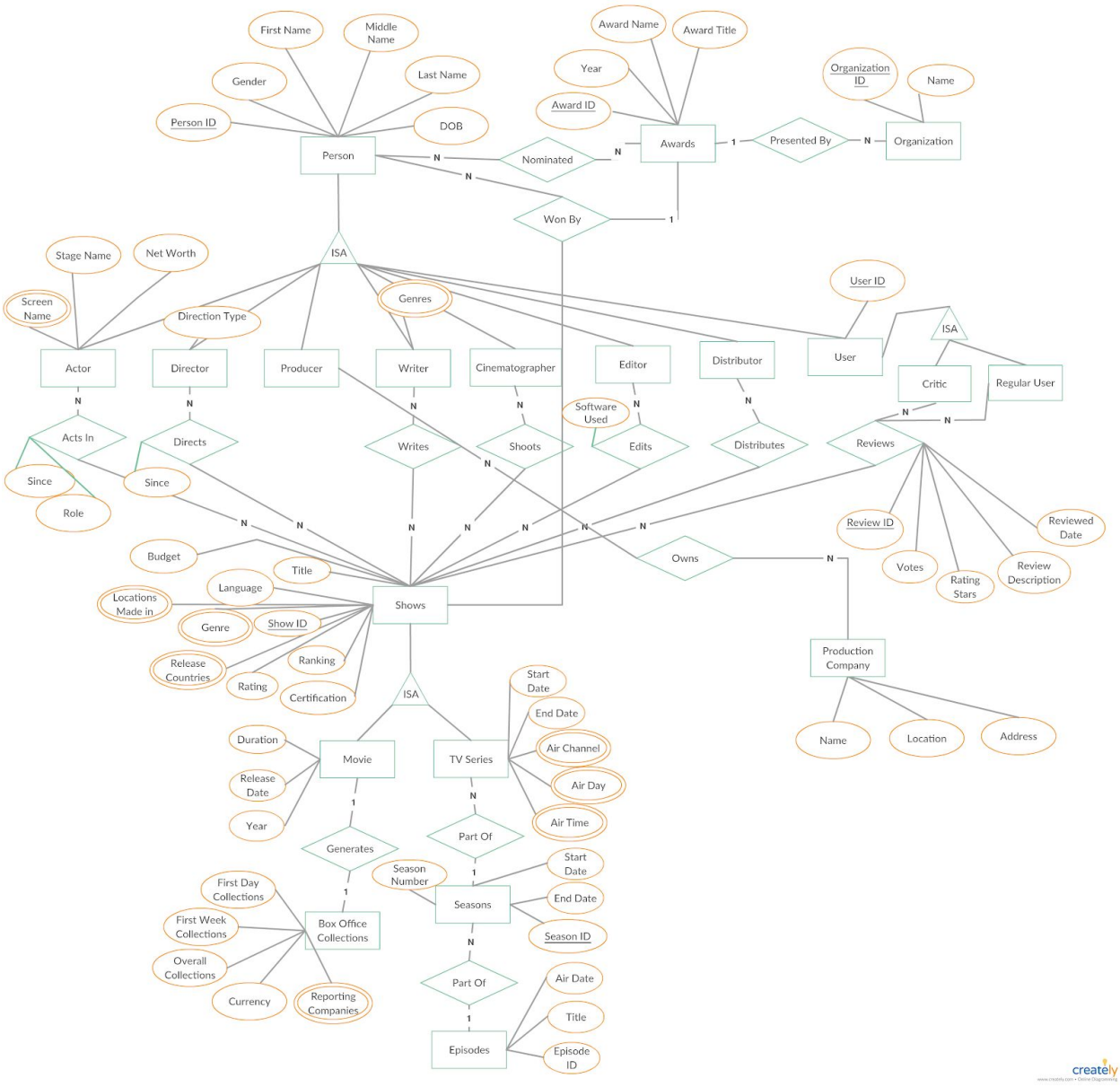
Group members:

1. Jahnavi Chowdary Tharigopula, 112078393
2. Vamsikrishna Kodumuru Meesala, 112006882

Highlighting the different stages of the project development:

We have developed a Movie database system, where the information regarding Actors, Directors, Movies, Reviews, ratings etc will be saved. Going through the project description and websites like “IMDB”, we have identified few entities, found the relationships between them, Constructed the database, scrapped the data from IMDB, Inserted the data into Database and Designed an UI in UBUNTU using mysql and PHP. Detailed information of the process flow in each step of the project is given below.

E-R Chart:



ER diagram of the Database system

Description of Entities in E-R diagram

Assumptions in E-R Diagram :

- **1** in the ER Diagram describes ‘exactly one’ cardinality
- **N** in the ER Diagram described ‘1 … *’ (i.e at least 1, up to N) cardinality.

The Key Entities identified are as follows:

Person:

- A person can be an Actor, Director, Producer, Writer, Cinematographer, Editor, Distributor, User.
- All the above are entities by themselves and they can be connected to Person using the **ISA** relationship.
- A person is identified using the following:
    - Person ID: This is the Primary Key that uniquely identifies a person
    - First Name
    - Middle Name
    - Last Name
    - DOB
- A person can be both Nominated or Win Awards (Awards is explained below). Hence, this is another Entity that a person is related to.
    - A person can receive N awards, and can also be nominated for N awards. Hence, the N-N cardinality relationship.
    - A person can win N awards, but 1 award can be won only by one person. Hence, the N-1 cardinality relationship.

Awards:

- An award can be identified with the following:
    - Award ID: An assumption made is that Award ID is unique for an award with a different Award Name and Award Title given in a different year. Hence, this can be considered as the Primary Key.
    - Award Name: Ex - Oscar, Golden Globe, Emmy etc
    - Award Title: Ex - Best Actor, Best Actress etc
    - Year
- An award is presented by an Organization.
    - Ex - Oscar is presented by Academy of Motion Picture Arts and Sciences.
    - Golden Globe is presented by the Hollywood Foreign Press Association.
    - The Organization can be identified by its Name and ID with Organization ID being the Primary Key.
    - One organization can give multiple awards, but one particular award can be given only by one organization. Hence, the 1-N cardinality relationship.

Actor:

- An actor can have the following attributes:
    - Stage Name - An actor can have a Stage Name that is different from their actual given name.

    - Screen Name - This indicates the names of the actors in the movies. An actor who has acted in more than 1 movie can have multiple screen names. Hence, describes as a multi value attribute.
    - Net Worth

Director:

- A director can have the following attributes:
    - A director could mean a Music Director, Movie Director, Dance Director/Choreographer etc.
    - Domain: Hence, this indicates whether the Director is a Music/Movie/Dance director.

Producer:

- A producer owns a Production Company.
- **Production Company** can have the following attributes:
    - Name
    - Location
    - Address
- A producer can own N companies and one Production company can be owned by N producers. Hence, the N-N cardinality relationship.

Writer:

- A writer can have the following attributes:
    - Genre - A writer can write different Genres. Hence, it is a multivalue attribute.

All the above people can be linked to **Shows** using the respective relationships as shown in the ER. (Ex - Actor, acts in; Director, directs ….)

User:

- A User can be a Critic or a Normal User. Hence, there can be an **ISA** relationship here.
- Both Critics and Normal Users provide Reviews. Reviews can have the following attributes:
    - Review ID which is the Primary key.
    - Votes
    - Rating Stars
    - Review Description
    - Reviewed Date

Shows:

- A show can be both a Movie or a TV Show. Hence, these can be linked using an **ISA** relationship.
- Shows can be identified using the following attributes:
    - Show ID: This serves as the primary key
    - Title
    - Genre - Multivalued attribute as 1 Movie/ TV show can fall under multiple genres.
    - Locations Made In - Multivalued attribute
    - Release Countries - Multivalued attribute
    - Language - Assumption made here is that a show is shot in a single language (although it can be dubbed and be made available in multiple languages)
    - Rating
    - Ranking
    - Certification - Ex: U, PG-13 etc
- Shows are produced by a Production Company.

Movie:

- A movie has the following attributes:
    - Release Date
    - Duration
    - Year of release
- A movie generates revenue. Hence, has another entity indicating the **Box Office Collections** which can have the following attributes:
    - First Day Collections
    - First Week Collections
    - Overall Collections
    - Currency in which it is being reported in
    - Reporting Companies - Multiple companies can produce multiple values to the above collections. Hence, this is a multivalued attribute.

TV Series:

- A TV Series has the following attributes:
    - Start Date
    - End Date
    - Air Channel
    - Air Day
    - Air Time
    - Air Channel, Day, and Time can be multivalued as it can air on multiple channels, on different days at different timings.
- Seasons are Part of a TV Series and have the following attributes

    - Start Date
    - End Date
    - Season Number: This is unique
- Episodes are part of the Season and have the following attributes
    - Air Date - Here it is not multivalued as one episode would have first aired on a unique day.
    - Title
    - Episode ID

Mapping E-R diagram to relationships:

From the E-R diagram which was drawn previously, the corresponding tables of the entities and relations are made using the Mysql Statements. The description of the each table and SQL Code for the table creation are as follows.

Table - Person:

- A person is a super-set of Actors, Directors, Producers, Writers, Cinematographers, Editors, Distributors and Users(includes normal users and critiques).
- person table will have the following attributes:
    - Person ID: This is the Primary Key that uniquely identifies a person
    - Gender
    - First Name
    - Middle Name
    - Last Name
    - DOB

```
CREATE TABLE Person(
        Person_Id INTEGER NOT NULL,
        Gender char(1),
        First_Name VARCHAR(30),
        Middle_Name VARCHAR(30),
        Last_Name VARCHAR(30),
        DOB DATE,
        PRIMARY KEY(Person_Id)
        Check (Gender in('M','F','O')));
```

Table - Award:

- An award will contain the following attributes:
    - Award ID: An assumption made is that Award ID is unique for an award with a different Award Name and Award Title given in a different year. Hence, this can be considered as the Primary Key.
    - Award Name: Ex - Oscar, Golden Globe, Emmy etc
    - Award Title: Ex - Best Actor, Best Actress etc

- Year
- This table contains the information of the awards that are given out.

```
CREATE TABLE Award(
        Award_Id INTEGER NOT NULL,
        Name VARCHAR(50),
        Title VARCHAR(30),
        Year INTEGER,
        PRIMARY KEY(Award_Id));
```

Note: Assumption is that the Award ID is different for different years. Hence, it can uniquely distinguish the tuples.

Table-Persons Nominated:

- A person or a show can get nominated for an award so we create two tables one for persons nominated and one for shows nominated
- This table contains the information of the persons who were nominated for a particular award.

```
CREATE TABLE PersonsNominated(
        Person_Id INTEGER NOT NULL,
        Award_Id INTEGER NOT NULL,
        PRIMARY KEY(Person_Id, Award_Id),
        FOREIGN KEY(Person_Id) REFERENCES Person(Person_Id),
        FOREIGN KEY(Award_Id) REFERENCES Award(Award_Id));
```

Table-Shows Nominated:

- This table contains the information of the shows which were nominated for a particular award.

```
CREATE TABLE ShowsNominated(
        Show_Id INTEGER NOT NULL,
        Award_Id INTEGER NOT NULL,
        PRIMARY KEY(Show_Id, Award_Id),
        FOREIGN KEY(Show_Id) REFERENCES Show(Show_Id),
        FOREIGN KEY(Award_Id) REFERENCES Award(Award_Id));
```

Table-Presented_By:

- This table provides the information on what all awards the organizations present.

```
CREATE TABLE Presented_By(
        Award_Id INTEGER NOT NULL,
        Organization_Id INTEGER NOT NULL,
```

```
PRIMARY KEY(Award_Id),
FOREIGN KEY(Award_Id) REFERENCES Award(Award_Id)),
FOREIGN KEY(Organization_Id) References Organization(Organization_Id);
```

Note: As the Award Ids are unique (i.e. Same awards (eg: the best actor awards) given by the same organization in different years will have different Id) This table can be removed by inserting a column “presented_by” in the Awards table.

Table-Won_by:

- This table provides the information on who (person/Show) won the award.

```
CREATE TABLE Won_by(
        Award_Id INTEGER NOT NULL,
        Person_Id INTEGER,
        Show_Id INTEGER,
        PRIMARY KEY(Award_Id),
        FOREIGN KEY(Award_Id) REFERENCES Awards(Award_Id),
        FOREIGN KEY(Person_Id) REFERENCES Person(Person_Id),
        FOREIGN KEY(Show_Id) REFERENCES Shows(Show_Id),
        Check ( (Show_Id IS NOT NULL AND Person_Id IS NULL) OR
                (Show_Id IS NULL AND Person_Id IS NOT NULL)));
```

Note: In the above table we are making use of a check statement to make sure that either Show_Id or Person_Id only one is not null. Here both cannot be “null” and both cannot be “not null” at the same time because an award can be won by either a person or show but it cannot be won by both.

Table - Actor:

- Actor table will have the following attributes:
    - Stage Name - An actor can have a Stage Name that is different from their actual given name.
    - Screen Name - This indicates the names of the actors in the movies. An actor who has acted in more than 1 movie can have multiple screen names. Hence, describes as a multi value attribute.
    - Net Worth
- Actor ISA person, hence Person ID in Actor references the Person Table.

```
CREATE TABLE Actor(
        Person_Id INTEGER NOT NULL,
        Screen_Name VARCHAR(30),
        Stage_Name VARCHAR(30),
        Net_Worth INTEGER,
```

```
        PRIMARY KEY(Person_Id),
        FOREIGN KEY(Person_Id) REFERENCES Person(Person_Id) ON DELETE
        CASCADE);
```

Table - Acting:

- This table provides the information on who all acted in the given film.

```
CREATE TABLE Acting(
        Actor_Id INTEGER NOT NULL,
        Show_Id INTEGER NOT NULL,
        Since DATE,
        Role VARCHAR(50),
        PRIMARY KEY(Actor_Id,Show_Id),
        FOREIGN KEY(Actor_Id) REFERENCES Actor(Person_Id)  ON DELETE
CASCADE,
        FOREIGN KEY(Show_Id) REFERENCES Shows(Show_Id) ON DELETE
CASCADE);
```

Table - Director:

- A director table will have the following special attribute:
    - Direction Type: Hence, this indicates whether the Director is a Music/Movie/Dance director.

```
CREATE TABLE Director(
        Person_Id INTEGER NOT NULL,
        Direction_Type Varchar(10),
        PRIMARY KEY(Person_Id),
        Check (Direction_Type  in('Music','Movie','Dance','Art'))
        FOREIGN KEY(Person_Id) REFERENCES Person(Person_Id) ON DELETE
        CASCADE);
```

Table - Direction:

- This table provides the information regarding which all movies were directed by a director.

```
CREATE TABLE Direction(
        Director_Id INTEGER NOT NULL,
        Show_Id INTEGER NOT NULL,
        Since DATE,
        PRIMARY KEY(Director_Id,Show_Id),
        FOREIGN KEY(Director_Id) REFERENCES Director(Person_Id) ON DELETE
CASCADE,
```

```
                FOREIGN KEY(Show_Id) REFERENCES Shows(Show_Id) ON DELETE
CASCADE);
```

Table-Producer:

- A producer is also a person and he owns a Production Company, Information of the production company that he/she owns will be presented in 'owns' table

```
CREATE TABLE Producer(
                Person_Id INTEGER NOT NULL,
                PRIMARY KEY(Person_Id),
                FOREIGN KEY(Person_Id) REFERENCES Person(Person_Id) ON DELETE
                CASCADE);
```

Table - Production Company:

- **Production Company** table will have the following attributes:
  - Name
  - Location
  - Address

```
CREATE TABLE Production_Company(
                Production_company_Id INTEGER  NOT NULL,
                Name VARCHAR(30),
                Location VARCHAR(30),
                Address VARCHAR(200),
                PRIMARY KEY(Prod_Comp_Id));
```

Table - Produces:

- It is assumed that one or multiple production companies will produce a show, This table provides the information related to the same.
- It is assumed that a show must have a production company

```
CREATE TABLE Produces(
                Production_company_Id : INTEGER NOT NULL,
                Show_Id : INTEGER NOT NULL,
                FOREIGN KEY(Show_Id ) REFERENCES shows(Show_Id ) ON DELETE
CASCADE,
                FOREIGN KEY(Production_company_Id) REFERENCES Production
                company(Production_company_Id) ON Delete CASCADE),
                PRIMARY KEY(Production_company_Id,Show_Id)
```

Table - Owns:

- A producer/s own a production company and he/she can own multiple companies and one Production company can be owned by multiple producers.
- This table provides the information on who owes a production company.
- In a case where multiple people owning a single production company multiple tuples will exist with same production company ID but with different person-Id's

```
CREATE TABLE Owns(
        Producer_Id: INTEGER NOT NULL,
        Production_company_Id: INTEGER NOT NULL,
        FOREIGN KEY(Producer_Id) References Producer(Person_Id) ON DELETE
CASCADE,
        FOREIGN KEY(Production_company_Id) REFERENCES Production
                company(Production_company_Id) ON Delete CASCADE),
        PRIMARY KEY(Producer_Id,Production_company_Id));
```

Table-Writer:

- A writer table will have the following attributes:
    - Genre - A writer can write different Genres. Hence, it is a multivalue attribute.
- This table contains the information of the Writer.

```
CREATE TABLE Writer(
                Person_Id INTEGER NOT NULL,
                Genre Varchar(10),
                PRIMARY KEY(Person_Id),
                FOREIGN KEY(Person_Id) REFERENCES Person(Person_Id) ON
        DELETE CASCADE,
                CHECK (Genre IN ('Action', 'Thriller', 'Sci-Fi','Comedy','Romantic'));
```

Note: A check statement is written to make sure that the genres exist in the given set of elements

Table Written:

- This table provides the information on who wrote the given show.

```
CREATE TABLE Written(
        Writer_Id INTEGER NOT NULL,
        Show_Id INTEGER NOT NULL,
        PRIMARY KEY(Writer_Id,Show_Id),
        FOREIGN KEY(Writer_Id) REFERENCES Writer(Person_Id) ON DELETE
CASCADE,
        FOREIGN KEY(Show_Id) REFERENCES Shows(Show_Id) ON DELETE
CASCADE);
```

Table-Cinematographer:

- A cinematographer shoots Movie/TV Show
- This table provides the information about the Cinematographers

```
CREATE TABLE Cinematographer(
        Person_Id INTEGER NOT NULL,
        PRIMARY KEY(Person_Id),
        FOREIGN KEY(Person_Id) REFERENCES Person(Person_Id) ON DELETE
        CASCADE);
```

Table-Shooting:

- This table contains the information of the Cinematographer who worked for a given film.

```
CREATE TABLE Shooting(
        Cinematographer_Id INTEGER NOT NULL,
        Show_Id INTEGER NOT NULL,
        PRIMARY KEY(Cinematographer_Id ,Show_Id),
        FOREIGN KEY(Cinematographer_Id) REFERENCES
        Cinematographer_I(Person_Id) ON DELETE CASCADE,
        FOREIGN KEY(Show_Id) REFERENCES Shows(Show_Id) ON DELETE
CASCADE);
```

Table-Editor:

- An editor Edits the Movie/TV Show
- This table contains the information of the Editors.

```
CREATE TABLE Editor(
        Person_Id INTEGER NOT NULL,
        PRIMARY KEY(Person_Id),
        FOREIGN KEY(Person_Id) REFERENCES Person(Person_Id) ON DELETE
        CASCADE);
```

Table-Editing:

- This Table provides the information about the editors worked for a given movie.

```
CREATE TABLE Editing(
        Editor_Id INTEGER NOT NULL,
        Show_Id INTEGER NOT NULL,
        Software_used Varchar(100),
        PRIMARY KE(Editor_Id ,Show_Id),
```

```
        FOREIGN KEY(Editor_Id ) REFERENCES Editor(Person_Id) ON DELETE
CASCADE,
        FOREIGN KEY(Show_Id) REFERENCES Shows(Show_Id) ON DELETE
CASCADE);
```

Table - Distributor:

- A Distributor distributes the Movie/TV Show.
- This table contains the information of a Distributor.

```
CREATE TABLE Distributor(
        Person_Id INTEGER NOT NULL,
        PRIMARY KEY(Person_Id),
        FOREIGN KEY(Person_Id) REFERENCES Person(Person_Id) ON DELETE
        CASCADE);
```

Table-Distributing:

- This table provides the information about the distributors who distributed the given movie.

```
CREATE TABLE Distributing(
        Distributor_Id Integer NOT NULL,
        Show_Id Integer NOT NULL,
        PRIMARY KEY(Distributor_Id ,Show_Id),
        FOREIGN KEY(Distributor_Id ) REFERENCES Distributor(Person_Id) ON
        DELETE CASCADE,
        FOREIGN KEY(Show_Id) REFERENCES Shows(Show_Id) ON DELETE
CASCADE);
```

Table - User:

- A user can be both critique and the normal user. All users will have a person Id as user is a person and they will be having another attribute called user-Id which will be used for logging in.
- This table contains the information of a User.

```
CREATE TABLE User(
        Person_Id INTEGER NOT NULL,
        User_Id VARCHAR(30),
        PRIMARY KEY(Person_Id),
        FOREIGN KEY(Person_Id) REFERENCES Person(Person_Id) ON DELETE
        CASCADE);
```

Table-Critic:

- This table contains the information of a Critic.

```
CREATE TABLE Critic(
        Person_Id INTEGER NOT NULL,
        User_Id VARCHAR(30),
        PRIMARY KEY(Person_Id),
        FOREIGN KEY(Person_Id) REFERENCES User(Person_Id) ON DELETE
        CASCADE);
        FOREIGN KEY(User_Id) REFERENCES User(User_Id) ON DELETE
        CASCADE);
```

Table-Regular_User:

- This table contains the information of a Regular User.

```
CREATE TABLE Regular_User(
        Person_Id INTEGER NOT NULL,
        User_Id VARCHAR NOT NULL,
        PRIMARY KEY(User_Id),
        FOREIGN KEY(Person_Id) REFERENCES User(Person_Id) ON DELETE
        CASCADE);
        FOREIGN KEY(User_Id) REFERENCES User(User_Id) ON DELETE
        CASCADE);
```

Table-Reviews:

- This table consists of the reviews given by all the users including critiques.
- For displaying the critique review separately we can filter the reviews based on the person_Id in the Critique table.
- The number of Up-Votes and Down-Votes of the review will also be recorded in the table.

```
CREATE TABLE Reviews(
        Review_Id INTEGER NOT NULL,
        User_Id INTEGER NOT NULL,
        UP_Votes INTEGER,
        Down_Votes INTEGER,
        Rating FLOAT,
        Review_Description VARCHAR(20000),
        Reviwed_Date DATE,
        PRIMARY KEY(Review_Id),
        FOREIGN KEY(User_Id) REFERENCES User(User_Id) ON DELETE
CASCADE);
```

Table - Shows:

- Shows include both a Movie or a TV Show.
- Shows can be identified using the following attributes:
    - Show ID: This serves as the primary key
    - Title
    - Genre - Multivalued attribute as 1 Movie/ TV show can fall under multiple genres.
    - Locations Made In - Multivalued attribute
    - Release Countries - Multivalued attribute
    - Language - Assumption made here is that a show is shot in a single language (although it can be dubbed and be made available in multiple languages)
    - Rating
    - Ranking
    - Certification - Ex: U, PG-13 etc
- Shows are produced by a Production Company. (This information will be presented in produced table)
- This table provides the information about the Shows. Shows includes both Movies and TV Shows.

```
CREATE TABLE Shows(
        Show_Id INTEGER  NOT NULL,
        Title VARCHAR(30),
        Language VARCHAR(30),
        Ranking INTEGER,
        Rating FLOAT,
        Certification Varchar (10),
        Budget INTEGER,
        Genre VARCHAR(30),
        Release_Countries VARCHAR(30),
        Locations_Made_In VARCHAR(30),
        PRIMARY KEY(Show_Id)
        CHECK (VALUE IN ('U','U/A','A','PG-13'));;
```

Table - Movie:

- A movie has the following attributes:
    - Release Date
    - Duration
    - Year of release
- This table provides the information about the Movies.

```
CREATE TABLE Movies(
        Show_Id INTEGER  NOT NULL,
```

```
        Duration FLOAT,
        Release_Date DATE,
        Year INTEGER,
        PRIMARY KEY(Show_Id)
        FOREIGN KEY(Show_Id) REFERENCE Shows(Show_Id) ON DELETE
CASCADE);
```

Table-Box Office Collections:

- A movie generates revenue. Hence, has another table indicating the **Box Office Collections** which can have the following attributes:
    - First Day Collections
    - First Week Collections
    - Overall Collections
    - Currency in which it is being reported in
    - Reporting Companies - Multiple companies can produce multiple values to the above collections. Hence, this is a multivalued attribute.
- This table provides the information about the Box Office Collections of the Movies.

```
CREATE TABLE Box_Office_Collections(
        Movie_Id INTEGER  NOT NULL,
        First_Day_Collections FLOAT,
        First_Week_Collections FLOAT,
        Overall_Collections FLOAT,
        Currency VARCHAR(10),
        Reporting_Companies VARCHAR(20),
        PRIMARY KEY(Show_Id)
        FOREIGN KEY(Show_Id) REFERENCE Movie(Show_Id) ON DELETE
CASCADE);
```

Note: We will not be creating a new table called generates as shown in the ER diagram because a movie can only have a single Box office collection entry, so having Movie Id in the Box-office collection will represent the relation between movie and its box-office collections.

Table - TV Series:

- A TV Series has the following attributes:
    - Start Date
    - End Date
    - Air Channel
    - Air Day
    - Air Time
    - Air Channel, Day, and Time can be multivalued as it can air on multiple channels, on different days at different timings.
- This table provides the information about the TVseries

```
CREATE TABLE TVSeries(
                Show_Id INTEGER  NOT NULL,
                Start_date DATE,
                End_date DATE,
                Air_Channel VARCHAR(50),
                Air_Day VARCHAR(30)(50),
                Air_time TIME,
                FOREIGN KEY(Show_Id) REFERENCES shows(Show_Id) on Delete
CASCADE,
                PRIMARY KEY(Show_Id));
```

Table - Seasons:

- Seasons are Part of a TV Series and have the following attributes
    - Start Date
    - End Date
    - Season Number: This is unique
- This table provides the episode information of the seasons of TV Series in the seasons table

```
CREATE TABLE Seasons(
                Season_Id INTEGER NOT NULL,
                Season_Name VARCHAR(30)(50),
                Start_date DATE,
                End_date DATE,
                Season_number INTEGER,
                Show_Id Integer NOT NULL,
                FOREIGN KEY(Show_Id) REFERENCES shows(Show_Id) ON DELETE
CASCADE);
```

Note: It is considered that the id of seasons of multiple shows will be unique.(season 1 of the game of thrones and the season-1 of the friends will have different season Id's. This is very important because we have attributes such as start date and end date.

Table - Episode:

- Episodes are part of the Season and have the following attributes
    - Air Date - Here it is not multivalued as one episode would have first aired on a unique day.
    - Title
    - Episode ID

- This table provides the episode information of the seasons of TV Series in the seasons table

```
CREATE TABLE Episodes(
        Episode_Id Int NOT NULL,
        Episode_Title VARCHAR(50),
        Air_date DATE,
        Duration float,
        Season_Id INTEGER NOT NULL,
        FOREIGN KEY(Season_Id) REFERENCES Seasons(Season_Id) ON DELETE
        CASCADE);
```

Supported Transactions and their Implementation

A Database is created with all the above mentioned tables and their constraints. The data has been scraped from IMDB using Beautiful soup and has been formatted and inserted into the tables.

The transactions that we have implemented can broadly be divided into 3 parts.

1. Transactions to 'View the existing Database'
2. Transactions to 'Search the Database'
3. Transactions to allow User to 'Add new entries into the Database'

Transactions to 'View the existing Database'

1. View Actors
    a. This allows us to view all the Actors present in the Database.
    b. This is done by performing a Join on
        i. The 'Actor' table - which contains information about who among the Persons are Actors,
        ii. The 'Person' Table - which contains information about every single Person present in the database
    c. SQL Code:

```
SELECT * FROM Actor t1 JOIN Person t2
ON t1.Person_Id = t2.Person_Id;
```

2. View Directors
    a. This allows us to view all the Directors present in the Database.
    b. This is done by performing a Join on
        i. The 'Director' table - which contains information about who among the Persons are Directors,
        ii. The 'Person' Table - which contains information about every single Person present in the database

c. SQL Code:
```
SELECT * FROM Director t1 JOIN Person t2
ON t1.Person_Id = t2.Person_Id;
```

3. View Movies
    a. This allows us to view all the Movies present in the Database.
    b. This is done by performing a Join on
        i. The ‘Movie’ table - which contains information about which among the Shows are Movies,
        ii. The ‘Shows’ Table - which contains information about every single Show present in the database, Show includes both Movies + TV Series
    c. SQL Code:
```
SELECT * FROM Movies t1 JOIN Shows t2
ON t1.Show_ID = t2.Show_Id;
```

4. View TV Shows
    a. This allows us to view all the TV Shows present in the Database.
    b. This is done by performing a Join on
        i. The ‘TV Series’ table - which contains information about which among the Shows are TV Series,
        ii. The ‘Shows’ Table - which contains information about every single Show present in the database, Show includes both Movies + TV Series
    c. SQL Code:
```
SELECT * FROM TVSeries t1 JOIN Shows t2
ON t1.Show_ID = t2.Show_Id;
```

5. Search People who are both Actors and Directors
    a. This allows us to view all the people who are both Actors and Directors.
    b. This is done by performing a Join on
        i. The ‘Director’ table - which contains information about who among the Persons are Directors,
        ii. The ‘Actor’ table - which contains information about who among the Persons are Actors,
        iii. The ‘Person’ Table - which contains information about every single Person present in the database.
    c. SQL Code:
```
SELECT * From Director t1 JOIN Actor t2
ON t1.Person_Id = t2.Person_Id JOIN Person t3
ON t1.Person_Id = t3.Person_Id;
```

6. View Reviews
    a. This allows us to view all the Reviews provided by the Users for the Shows existing in the Database.

b. This is done by performing a Join on
    i. The ‘Reviews’ table - which contains information about the Reviews provided by the Users,
    ii. The ‘Shows’ Table - which contains information about every single Show present in the database, Show includes both Movies + TV Series
c. SQL Code:

```
SELECT * FROM Reviews t1 JOIN Shows t2
ON t1.Show_Id = t2.Show_Id;
```

Transactions to ‘Search the Database’

1. Search Shows of Actors
    a. This allows us to Search the Database for all the Shows (Movies + TV Series) of a particular actor. The input for the Actor is provided by the User.
    b. This is done by performing a Join on
        i. The ‘Acting’ Table - which contains the Shows and Actor pairs, i.e which actors acted in which Shows,
        ii. The ‘Actor’ table - which contains information about who among the Persons are Actors,
        iii. The ‘Person’ Table - which contains information about every single Person present in the database,
        iv. The ‘Shows’ Table - which contains information about every single Show present in the database, Show includes both Movies + TV Series.
    c. SQL Code:

```
SELECT * FROM Acting t1 JOIN Actor t2
ON t1.Actor_Id = t2.Person_Id JOIN Person t3
ON t3.Person_Id = t2.Person_Id JOIN Shows t4
ON t1.Show_Id = t4.Show_Id
WHERE t3.First_Name LIKE '%Robert%' or t3.Last_Name LIKE '%Robert%';
```

2. Search Shows of Directors
    a. Similar to the above case, this allows us to Search the Database for all the Shows of a particular Director. The input for the Director is provided by the User.
    b. This is done by performing a Join on
        i. The ‘Directing’ Table - which contains the Shows and Director pairs, i.e which Directors directed in which Shows,
        ii. The ‘Director’ table - which contains information about who among the Persons are Directors,
        iii. The ‘Person’ Table - which contains information about every single Person present in the database,
        iv. The ‘Shows’ Table - which contains information about every single Show present in the database, Show includes both Movies + TV Series.
    c. SQL Code:

```
SELECT * FROM Direction t1 JOIN Director t2
```

```
ON t1.Director_Id = t2.Person_Id JOIN Person t3
ON t3.Person_Id = t2.Person_Id JOIN Shows t4
ON t1.Show_Id = t4.Show_Id
WHERE t3.First_Name LIKE '%John%' or t3.Last_Name LIKE '%John%';
```

3. Search Actors of Shows
    a. This allows us to Search the Database for all the Actors of a particular Show. The input for the Show is provided by the User.
    b. This is done by performing a Join on
        i. The 'Actor' table - which contains information about who among the Persons are Actors,
        ii. The 'Acting' Table - which contains the Shows and Actor pairs, i.e which actors acted in which Shows,
        iii. The 'Shows' Table - which contains information about every single Show present in the database, Show includes both Movies + TV Series,
        iv. The 'Person' Table - which contains information about every single Person present in the database.
    c. SQL Code:

```
SELECT * FROM Actor t1 JOIN Acting t2
ON t1.Person_Id = t2.Actor_Id JOIN Shows t3
ON t3.Show_Id = t2.Show_Id JOIN Person t4
ON t4.Person_Id = t1.Person_Id
WHERE t3.Title LIKE '%La La land%';
```

4. Search Rating Of Movie
    a. This allows us to Search the Database for the Rating of a particular Movie. The input for the Movie is provided by the User.
    b. This is done by performing a Join on
        i. The 'Movie' table - which contains information about which among the Shows are Movies,
        ii. The 'Shows' Table - which contains information about every single Show present in the database, Show includes both Movies + TV Series.
    c. SQL Code:

```
SELECT * FROM Movies t1 JOIN Shows t2
ON t1.Show_Id = t2.Show_Id WHERE t2.Title LIKE '%Avengers%';
```

5. Search Highest Grossing Movie by Year
    a. This allows us to Search the Database for the Highest Grossing Movies of a particular year. The input for the Year is provided by the User.
    b. This is done by performing a Join on
        i. The 'Box_Office_Collections' table - which contains information about the Box Office Collections of a particular Movie,

ii. The ‘Shows’ Table - which contains information about every single Show present in the database, Show includes both Movies + TV Series,
iii. The ‘Movie’ table - which contains information about which among the Shows are Movies.

c. The result provides all the Movies released in that year ordered in descending order of their Box Office Collections, the first entry indicating the highest grossing movie of that year.

d. SQL Code:

```
SELECT * FROM Box_Office_Collections t1 JOIN Shows t2
ON t1.Movie_Id = t2.Show_Id JOIN Movies t3
ON t1.Movie_Id = t3.Show_Id WHERE t3.Year = '2017'
ORDER BY Overall_Worldwide_Collections DESC;
```

6. Search Movies by Year
   a. This allows us to Search the Database for the Movies that were released in a particular year. The input for the Year is provided by the User.
   b. This is done by performing a Join on
      i. The ‘Movie’ table - which contains information about which among the Shows are Movies,
      ii. The ‘Shows’ Table - which contains information about every single Show present in the database, Show includes both Movies + TV Series.
   c. SQL Code:

```
SELECT * FROM Movies t1 JOIN Shows t2
ON t1.Show_Id = t2.Show_Id
WHERE t1.Year = 2017;
```

7. Search Shows by Genre
   a. This allows us to Search the Database for the Shows of a particular genre. The input for the Genre is provided by the User.
   b. This is done by performing a Join on
      i. The ‘In_Genre’ Table - which contains the Genre and Show pairs, i.e which Show belongs to which Genre,
      ii. The ‘Shows’ Table - which contains information about every single Show present in the database, Show includes both Movies + TV Series,
      iii. The ‘Genres’ table - which contains information about all the available Genres.
   c. SQL Code:

```
SELECT * from In_Genre t1 JOIN Shows t2
ON t1.Show_Id = t2.Show_Id JOIN Genres t3
ON t1.Genre_Id = t3.Genre_Id
WHERE t3.Name = "Action"
```

Transactions to allow User to ‘Add new entries into the Database’

- Our model allows the User to modify the database by adding new Actors and Directors into the database.
- A request to add a new entry (actor/director) into the database should insert the actor/director in the Person table as well, as the Person table keeps track of all the people existing in the database.
    - If an Actor already exists in the Person and Actor table, and we want to add him as a Director as well, then we first check whether the actor exists in the Person table, and if it does, we insert the entry only in the director table, and vice versa.
    - Whereas if the Actor already exists in the Person and Actor table, and we want to add the same person again as an Actor, then this request will be Failed as the entry already exists in both the tables, and vice versa for the Director as well.
- The following details of the Actor are provided by the User.
    - First Name
    - Last Name
    - DOB
    - Gender
    - Net Worth
    - Working Since Year

1. Add Actor
    a. This allows us to Add an Actor to the Database. The details of the Actor are provided by the User.
    b. This is done by performing an Insert into both the Person as well as actor table.
        i. The ‘Person’ Table - which contains information about every single Person present in the database. The Actor ID is obtained from the Person ID after inserting in to the Person Table.
        ii. The ‘Actor’ table - which contains information about who among the Persons are Actors.
    c. SQL Code:

```
INSERT INTO Person (Gender, First_Name, Last_Name, Middle_Name, DOB)
VALUES ('M','Mark','Ruffalo',null,'1967-09-22');

INSERT INTO Actor (Person_Id, Net_Worth, Since_Year)
VALUES ( (SELECT Person_Id FROM Person WHERE Gender = 'M' AND
First_Name = 'Mark' AND Last_Name = 'Ruffalo' AND DOB =
'1967-09-22'),30,1989);
```

    d. Result:
        i. If doesn’t exist in Person, inserts into both Person and Actor tables.
        ii. If exists in Person but doesn’t exist in Actor, inserts only into Actor table.
        iii. If exists in both Person and Actor table, the transaction Fails.

2. Add Director

a. Similar to the above case, this allows us to Add a Director to the Database. The details of the Director are provided by the User.
b. This is done by performing an Insert into both the Person as well as Director table.
    i. The ‘Person’ Table - which contains information about every single Person present in the database. The Director ID is obtained from the Person ID after inserting in to the Person Table.
    ii. The ‘Director’ table - which contains information about who among the Persons are Directors.
c. SQL Code:

```
INSERT INTO Person (Gender, First_Name, Last_Name, Middle_Name, DOB)
VALUES ('M','Nick','Cassavetes',null,'1954-05-21');

INSERT INTO Director (Person_Id, Direction_Type, Since_Year) VALUES (
(SELECT Person_Id FROM Person WHERE Gender = 'M' AND First_Name =
'Nick' AND Last_Name = 'Cassavetes' AND DOB = '1954-05-21'),'Movie',1970);
```

d. Result:
    i. If doesn’t exist in Person, inserts into both Person and Director tables.
    ii. If exists in Person but doesn’t exist in Director, inserts only into Director table.
    iii. If exists in both Person and Director table, the transaction Fails.

Results:

The results of all the transactions were reported in the 1_phase_report.pdf submitted under Assignment-3.

Contributions:

- Both have equally contributed in Refining the E-R Diagram and Finding the relationships between the tables.
- In addition to that Contributions during designing the UI are as follows,
    - **Jahnavi** - Scraping the data from IMDB, HTML part of webpages, UX Design, Refining the table Schema (50%), SQL Commands for Viewing and Searching data.
    - **Vamsikrishna** - Setting up the Environment for Database design, PHP part of Webpages, UX Design, Refining the table Schema (50%), SQL Commands for Searching and adding Data.