Final Assessment Report

Oscars Database

Done by: Mohamed Ahmed Mohamed Yassin Harby

Introduction:

This database has six entities: actors, directors, movies, Oscars, awards, and the host. I selected the years 2015–2018 for this database. Although there are many more awards at the actual Oscars, I only included the following six: Best Leading Actor, Best Supporting Actor, Best Leading Actress, Best Supporting Actress, Best Director, and Best Picture. This was because there would be too much information to gather precisely in a short amount of time. I started by creating the ER diagram, relational schema, and data description. Next, I generated the tables and filled them with real data. After that, I made ten distinct queries and then I made procedures, triggers, and views. Finally, I used Python to establish a connection to the database.

Design of the database:

Why did I select these entities?

I selected these entities by taking reference from the real Oscar ceremony, which features a host, awards for actors and directors, and movies. I also utilised appropriate data types, including VARCHAR, INT, and DATE, for each attribute.

Entity Sets:

Entity Set	keys	Attributes
Actors	ActorID	firstName, lastName, gender, birthDate
Directors	directorID	firstName, lastName, gender, birthdate
Movies	movieID	movieName, year, genre
Oscars	oscarsID	year
AwardsCategory	categoryID	name
Host	hostID	firstName, lastName, gender, birthdate

Relationship Tables:

Relationship Sets	Between Entity Sets	Attributes
Acts	Actors, movies	role
Direct	Director, movies	
Actor_Nominated	Actors, AwardsCategory, Oscars	MovieName, won
Director_Nominated	Director, AwardsCategory, Oscars	MovieName, won
Movie_Nominated	Movies, AwardsCategory	MovieName, won
Presents	Oscars, AwardsCategory	
Hosts	Host, Oscars	

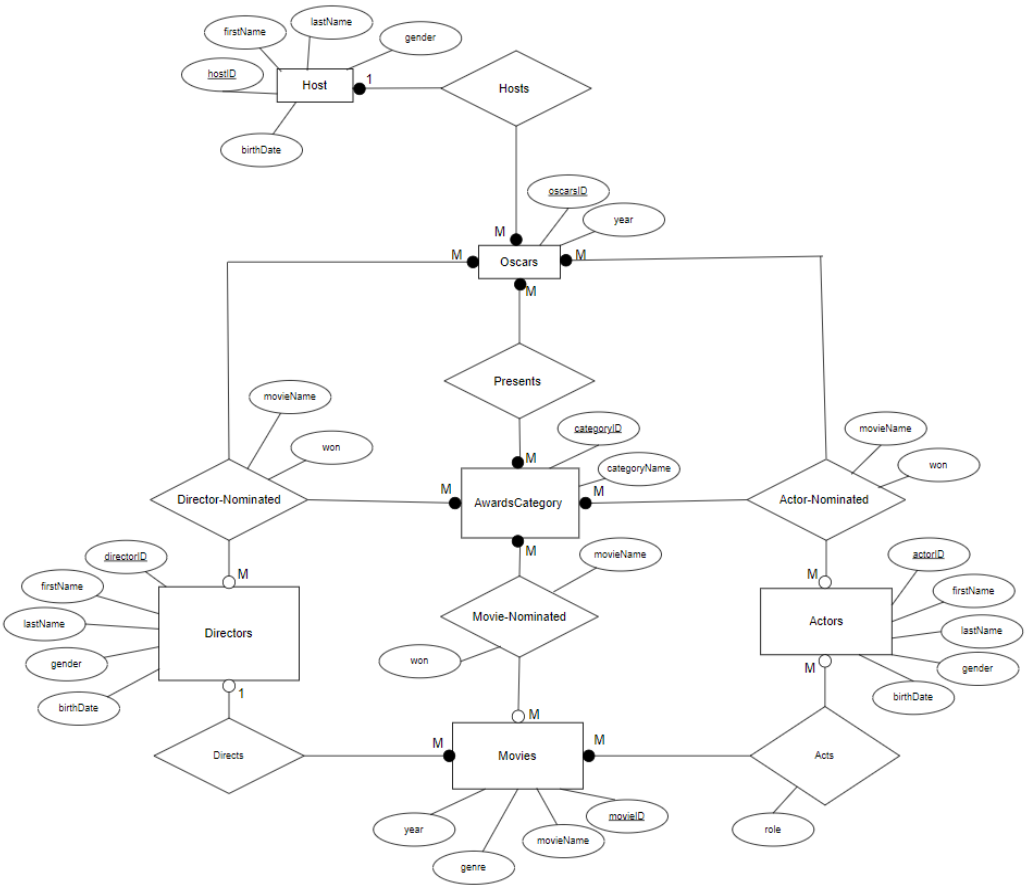
Constraints Table (Cardinality):

Relationship Sets	Cardinality	Description
Acts	M : M	One actor could act in many movies, and one movie could have many actors
Direct	1 : M	One director could direct many movies, but one movie can only be directed by one director
Actor_Nominated	M: M: M	One actor could be nominated to many awards categories, and one award category could have many actors nominated, in many different Oscars versions
Director_Nominated	M: M: M	One director could be nominated to many awards categories, and one award category could have many directors nominated, in many different Oscars versions
Movie_Nominated	M: M	One director could be nominated to many awards categories, and one award category could have many movies nominated
Presents	M: M	One Oscars version could present many awards categories, one award category could be present by many Oscars version
Hosts	1: M	One host can host many Oscars, and one Oscars could have many hosts

Constraints Table (Participation):

Relationship Sets	Participation	Description

Acts	Actors- partial, Movies- total	An actor can exist without acting in a movie, a movie cannot exist without actors
Direct	Directors - partial, Movies- total	A director can exist without acting in a movie, a movie cannot exist without a director
Actor_Nominated	Actors- partial, AwardsCategory – total, Oscars – total	An actor can exist without getting nominated for any award, an award category cannot exist without having any nominees, and Oscars cannot exist without actors getting nominated
Director_Nominated	Directors - partial, AwardsCategory – total	A director can exist without getting nominated for any award, an award category cannot exist without having any nominees, and Oscars cannot exist without directors getting nominated
Movie_Nominated	Movies- partial, AwardsCategory – total	A movie can exist without getting nominated for any award, an award category cannot exist without having any nominees
Presents	Oscars- total, AwardsCategory – total	An Oscars can only exist with AwardsCategory, AwardsCategory can only exist in the Oscars
Hosts	Host- total, Oscars- total	A host can only exist while hosting the Oscars, the Oscars cannot exist without a host



Assumption:

I searched for actor IDs and attempted to use a composite key as the primary key in an effort to make this database as authentic as possible, but I could not discover any actor IDs. Based on my research, I assumed that the Oscars, if they have a database, would likely employ a primary key that is some sort of confidential data, for example, the passport number. I therefore made fake IDs for each individual because it is not possible to produce a unique composite key for each of these things. Additionally, since the names of the movies are not protected by copyright, I created a fake ID for each one so that it could be used as a primary key. In addition, I am assuming that there can only be one host for every Oscars and that every movie can only have one director (I'm not accounting for co-directors) and at least one actor.

Relational scheme

Host (hostID, firstName, lastName, gender, birthDate)

Oscars (oscarsID, year, hostID)

fk hostID references host(hostID) ON DELETE SET NULL, ON UPDATE CASCADE

AwardsCategory(categoryID, categoryName)

Presents (oscarsID, categoryID)

fk oscarsID references Oscars(oscarsID) ON DELETE RESTRICT, ON UPDATE CASCADE

fk categoryID references AwardsCategory(categoryID) ON DELETE RESTRICT, ON UPDATE CASCADE

Actors (actorID, firstName, lastName, gender, birthdate)

Directors (directorID, firstName, lastName, gender, birthdate)

Movies (movieID, movieName, year, genre, directorID)

fk directorID references Directors(directorID) ON DELETE SET NULL, ON UPDATE CASCADE

Acts (movieID, actorID, role)

fk movieID references Movies(movieID) ON DELETE RESTRICT, ON UPDATE CASCADE

fk actorID references Actors(actorID) ON DELETE RESTRICT, ON UPDATE CASCADE

Actor_Nominated (oscarsID , actorID, categoryID, movieName, won)

fk actorID references Actors(actorID) ON DELETE RESTRICT, ON UPDATE CASCADE

fk categoryID references AwardsCategory(categoryID) ON DELETE RESTRICT, ON UPDATE CASCADE

fk oscarsID references Oscars(oscarsID) ON DELETE RESTRICT, ON UPDATE CASCADE

Director_Nominated (oscarsID, directorID, categoryID, movieName, won)

fk directorID references Directors(directorID) ON DELETE RESTRICT, ON UPDATE CASCADE

fk categoryID references AwardsCategory(categoryID) ON DELETE RESTRICT, ON UPDATE CASCADE

fk oscarsID references Oscars(oscarsID) ON DELETE RESTRICT, ON UPDATE CASCADE

Movie_Nominated (movieID, categoryID, movieName, won)

fk movieID references Movies(movieID) ON DELETE RESTRICT, ON UPDATE CASCADE

fk categoryID references AwardsCategory(categoryID) ON DELETE RESTRICT, ON UPDATE CASCADE

All the entities are in 3NF and BCNF.

Data Description:

Table: Hosts

Attribute	Type	Size	Null	Primary Key	Description	Other Constraints
hostID	INT	-	N	Y	Host ID	-
firstName	VARCHAR	50	-	-	Host's First name	-
lastName	VARCHAR	50	-	-	Host's Last name	-
gender	VARCHAR	10	-	-	Host's Gender	-
birthDate	DATE	-	-	-	Host's Date of birth	-

Table: Oscars

Attribute	Type	Size	Null	Primary Key	Description	Other Constraints
oscarsID	INT	-	N	Y	Oscars ID	-
year	INT	-	-	-	Year of the Oscars	-
hostID	INT	-	N	-	Host ID	FOREIGN KEY (hostID) REFERENCES Hosts(hostID) ON DELETE SET NULL ON UPDATE CASCADE

Table: AwardsCategory

Attribute	Type	Size	Null	Primary Key	Description	Other Constraints
categoryID	INT	-	N	Y	Category ID	-
categoryName	VARCHAR	100	-	-	Name of the Category	-

Table: Presents

Attribute	Type	Size	Null	Primary Key	Description	Other Constraints
oscarsID	INT	-	N	Y	OscarsID	FOREIGN KEY (oscarsID) REFERENCES Oscars(oscarsID) ON DELETE RESTRICT ON UPDATE CASCADE
Category ID	INT	-	N	Y	Category ID	

Table: Actors

Attribute	Type	Size	Null	Primary Key	Description	Other Constraints
actorID	INT	-	N	Y	Actor ID	-
firstName	VARCHAR	50	-	-	Actor's First name	-
lastName	VARCHAR	50	-	-	Actor's Last name	-
gender	VARCHAR	10	-	-	Actor's Gender	-
birthDate	DATE	-	-	-	Actor's Date of birth	-

Table: Directors

Attribute	Type	Size	Null	Primary Key	Description	Other Constraints
directorID	INT	-	N	Y	Director ID	-
firstName	VARCHAR	50	-	-	Director's First name	-
lastName	VARCHAR	50	-	-	Director's Last name	-
gender	VARCHAR	10	-	-	Director's Gender	-
birthDate	DATE	-	-	-	Director's Date of birth	-

Table: Movies

Attribute	Type	Size	Null	Primary Key	Description	Other Constraints
movieID	INT	-	N	Y	Movie ID	-
movieName	VARCHAR	100	-	-	Movie name	-
year	INT	-	-	-	Year of release	-
genre	VARCHAR	50	-	-	Genre of the movie	-
directorID	INT	-	N	-	Director ID	FOREIGN KEY (directorID) REFERENCES Directors(directorID) ON DELETE SET NULL ON UPDATE CASCADE

Table: Acts

Attribute	Type	Size	Null	Primary Key	Description	Other Constraints
movieID	INT	-	N	Y	Movie ID	FOREIGN KEY (movieID) REFERENCES Movies(movieID) ON DELETE RESTRICT ON UPDATE CASCADE
actorID	INT	-	N	Y	Actor ID	FOREIGN KEY (actorID) REFERENCES Actors(actorID) ON DELETE RESTRICT ON UPDATE CASCADE
role	VARCHAR	50	-	-	Role played by Actor	-

Table: Actor_Nominated

Attribute	Type	Size	Null	Primary Key	Description	Other Constraints
oscarsID	INT	-	N	Y	Oscars ID	FOREIGN KEY (oscarsID) REFERENCES Oscars(oscarsID) ON DELETE RESTRICT ON UPDATE CASCADE
actorID	INT	-	N	Y	Actor ID	FOREIGN KEY (actorID) REFERENCES Actors(actorID) ON DELETE RESTRICT ON UPDATE CASCADE
categoryID	INT	-	N	Y	Category ID	FOREIGN KEY (categoryID) REFERENCES AwardsCategory(categoryID) ON DELETE RESTRICT ON UPDATE CASCADE
movieName	VARCHAR	100	N	-	Name of the Movie	-
won	INT	-	N	-	Won or not	-

Table: Director_Nominated

Attribute	Type	Size	Null	Primary Key	Description	Other Constraints
oscarsID	INT	-	N	Y	Oscars ID	FOREIGN KEY (oscarsID) REFERENCES Oscars(oscarsID) ON DELETE RESTRICT ON UPDATE CASCADE
directorID	INT	-	N	Y	Director ID	FOREIGN KEY (actorID) REFERENCES Actors(actorID) ON DELETE RESTRICT ON UPDATE CASCADE
categoryID	INT	-	N	Y	Category ID	FOREIGN KEY (categoryID) REFERENCES AwardsCategory(categoryID) ON DELETE RESTRICT ON UPDATE CASCADE
movieName	VARCHAR	100	N	-	Name of the Movie	-
won	INT	-	N	-	Won or not	-

Table: Movie_Nominated

Attribute	Type	Size	Null	Primary Key	Description	Other Constraints
movieID	INT	-	N	Y	Movie ID	FOREIGN KEY (movieID) REFERENCES Movies(movieID) ON DELETE RESTRICT ON UPDATE
categoryID	INT	-	N	Y	Category ID	FOREIGN KEY (categoryID) REFERENCES AwardsCategory(categoryID) ON DELETE RESTRICT ON UPDATE CASCADE
movieName	VARCHAR	100	N	-	Name of the Movie	-
won	INT	-	N	-	Won or not	-

Implementation of the database:

First, I created a new database called Osacrs_21445116. Then, I used my produced data description to build the tables in MYSQL, making sure they had the right data types and constraints.

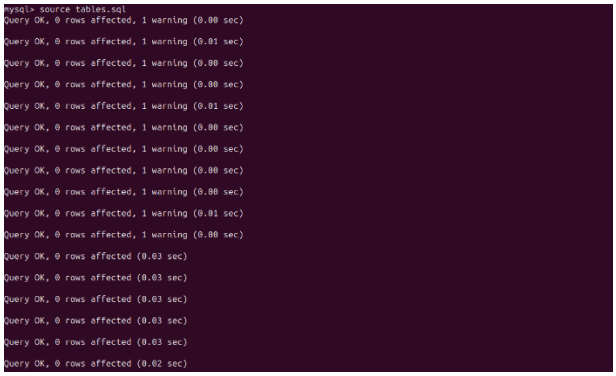
```sql
/* Drop tables if they exist */
DROP TABLE IF EXISTS Actor_Nominated;
DROP TABLE IF EXISTS Director_Nominated;
DROP TABLE IF EXISTS Movie_Nominated;
DROP TABLE IF EXISTS Acts;
DROP TABLE IF EXISTS Actors;
DROP TABLE IF EXISTS Directors;
DROP TABLE IF EXISTS Movies;
DROP TABLE IF EXISTS Presents;
DROP TABLE IF EXISTS AwardsCategory;
DROP TABLE IF EXISTS Oscars;
DROP TABLE IF EXISTS Host;

/* Create Host table, this will contain all the data regarding hosts*/
CREATE TABLE Host (
    hostID INT PRIMARY KEY,
    firstName VARCHAR(50),
    lastName VARCHAR(50),
    gender VARCHAR(10),
    birthDate DATE
);
```

```sql
/* Create Oscars table,  this will contain data about the Oscars */
CREATE TABLE Oscars (
    oscarsID INT PRIMARY KEY,
    year INT,
    hostID INT,
    FOREIGN KEY (hostID) REFERENCES Host(hostID) ON DELETE SET NULL ON UPDATE CASCADE
);

/* Create AwardsCategory table, this will contain data about the different awards */
CREATE TABLE AwardsCategory (
    categoryID INT PRIMARY KEY,
    categoryName VARCHAR(100)
);

/* Create Presents table, this will contain data regarding the Oscars presenting different awards */
CREATE TABLE Presents (
    oscarID INT,
    categoryID INT,
    PRIMARY KEY (oscarID, categoryID),
    FOREIGN KEY (oscarID) REFERENCES Oscars(oscarsID) ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (categoryID) REFERENCES AwardsCategory(categoryID) ON DELETE RESTRICT ON UPDATE CASCADE
);
```

```sql
CREATE TABLE Actors (
    actorID INT PRIMARY KEY,
    firstName VARCHAR(50),
    lastName VARCHAR(50),
    gender VARCHAR(10),
    birthdate DATE
);

/* Create Directors table, this will contain data about the directors  */
CREATE TABLE Directors (
    directorID INT PRIMARY KEY,
    firstName VARCHAR(50),
    lastName VARCHAR(50),
    gender VARCHAR(10),
    birthdate DATE
);

/* Create Movies table, this will contain data about the movies */
CREATE TABLE Movies (
    movieID INT PRIMARY KEY,
    movieName VARCHAR(100),
    year INT,
    genre VARCHAR(50),
    directorID INT,
    FOREIGN KEY (directorID) REFERENCES Directors(directorID) ON DELETE SET NULL ON UPDATE CASCADE
);
```

```sql
/* Create Acts table, this will contain data about movies and their actors */
CREATE TABLE Acts (
    movieID INT,
    actorID INT,
    role VARCHAR(100),
    PRIMARY KEY (movieID, actorID),
    FOREIGN KEY (movieID) REFERENCES Movies(movieID) ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (actorID) REFERENCES Actors(actorID) ON DELETE RESTRICT ON UPDATE CASCADE
);

/* Create Actor_Nominated table, this will contain the nominees for best actor for leading and supporting role, best actress for leading and supporting role */
CREATE TABLE Actor_Nominated (
    oscarsID INT,
    actorID INT,
    categoryID INT,
    movieName VARCHAR(100) NOT NULL,
    won INT NOT NULL,
    PRIMARY KEY (oscarsID, actorID, categoryID),
    FOREIGN KEY (oscarsID) REFERENCES Oscars(oscarsID) ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (actorID) REFERENCES Actors(actorID) ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (categoryID) REFERENCES AwardsCategory(categoryID) ON DELETE RESTRICT ON UPDATE CASCADE
);
```

```sql
/* Create Director_Nominated table, this will contain the nominees for best director */
CREATE TABLE Director_Nominated (
    oscarsID INT,
    directorID INT,
    categoryID INT,
    movieName VARCHAR(100) NOT NULL,
    won INT NOT NULL,
    PRIMARY KEY (oscarsID, directorID, categoryID),
    FOREIGN KEY (oscarsID) REFERENCES Oscars(oscarsID) ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (directorID) REFERENCES Directors(directorID) ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (categoryID) REFERENCES AwardsCategory(categoryID) ON DELETE RESTRICT ON UPDATE CASCADE
);

/* Create Movie_Nominated table, this will contain the nominees for best movie */
CREATE TABLE Movie_Nominated (
    movieID INT,
    categoryID INT,
    movieName VARCHAR(100) NOT NULL,
    won INT NOT NULL,
    PRIMARY KEY (movieID, categoryID),
    FOREIGN KEY (movieID) REFERENCES Movies(movieID) ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (categoryID) REFERENCES AwardsCategory(categoryID) ON DELETE RESTRICT ON UPDATE CASCADE
);
```



Sample data:

This part took the longest. First, I selected which Oscar years to gather data from—I decided that the years 2015–2018 would provide me with sufficient information. Next, I gathered information about random actors, directors, and hosts. Finally, I gathered all the information required for the nominees for each year. Finally, I ensured that every film in the database had at least one actor and only one director.

```sql
INSERT INTO Actors VALUES
    (1, 'Chris', 'Evans', 'Male', '1981-06-13'),
    (2, 'Emma', 'Watson', 'Female', '1990-04-15'),
    (3, 'Tom', 'Hardy', 'Male', '1977-09-15'),
    (4, 'Jessica', 'Chastain', 'Female', '1977-03-24'),
    (5, 'John', 'Krasinski', 'Male', '1979-10-20'),
    (6, 'Kristen', 'Stewart', 'Female', '1990-04-09'),
    (7, 'Idris', 'Elba', 'Male', '1972-09-06'),
    (8, 'Anna', 'Kendrick', 'Female', '1985-08-09'),
    (9, 'Chris', 'Hemsworth', 'Male', '1983-08-11'),
    (10, 'Emily', 'Blunt', 'Female', '1983-02-23'),
    (11, 'Michael', 'B. Jordan', 'Male', '1987-02-09'),
    (12, 'Lupita', 'Nyong''o', 'Female', '1983-03-01'),
    (13, 'Tom', 'Hiddleston', 'Male', '1981-02-09'),
    (14, 'Zoe', 'Saldana', 'Female', '1978-06-19'),
    (15, 'Idina', 'Menzel', 'Female', '1971-05-30'),
    (16, 'Jake', 'Gyllenhaal', 'Male', '1980-12-19'),
    (17, 'Amy', 'Adams', 'Female', '1974-08-20'),
    (18, 'Ryan', 'Reynolds', 'Male', '1976-10-23'),
    (19, 'Brie', 'Larson', 'Female', '1989-10-01'),
    (20, 'Michael', 'Fassbender', 'Male', '1977-04-02'),
    (21, 'Mahershala', 'Ali', 'Male', '1974-02-16'),
    (22, 'Christian', 'Bale', 'Male', '1974-01-30'),
    (23, 'Willem', 'Dafoe', 'Male', '1955-07-22'),
    (24, 'Sam', 'Elliott', 'Male', '1944-08-09'),
    (25, 'Richard E.', 'Grant', 'Male', '1957-05-05'),
    (26, 'Rami', 'Malek', 'Male', '1981-05-12'),
    (27, 'Viggo', 'Mortensen', 'Male', '1958-10-20'),
    (28, 'Sam', 'Rockwell', 'Male', '1968-11-05'),
    (29, 'Yalitza', 'Aparicio', 'Female', '1993-12-11'),
    (30, 'Marina', 'de Tavira', 'Female', '1974-11-30'),
    (31, 'Lady', 'Gaga', 'Female', '1986-03-28'),
```

```sql
INSERT INTO Directors
VALUES
    (1, 'Alfonso', 'Cuarón', 'Male', '1901-11-28'),
    (2, 'Spike', 'Lee', 'Male', '1957-03-20'),
    (3, 'Adam', 'McKay', 'Male', '1968-04-17'),
    (4, 'Yorgos', 'Lanthimos', 'Male', '1973-05-27'),
    (5, 'Pawel', 'Pawlikowski', 'Male', '1957-09-15'),
    (6, 'Greta', 'Gerwig', 'Female', '1983-08-04'),
    (7, 'Guillermo', 'del Toro', 'Male', '1964-10-09'),
    (8, 'Jordan', 'Peele', 'Male', '1979-02-21'),
    (9, 'Paul', 'Thomas Anderson', 'Male', '1970-06-26'),
    (10, 'Christopher', 'Nolan', 'Male', '1970-07-30'),
    (11, 'Barry', 'Jenkins', 'Male', '1979-11-19'),
    (12, 'Damien', 'Chazelle', 'Male', '1985-01-19'),
    (13, 'Denis', 'Villeneuve', 'Male', '1967-10-03'),
    (14, 'Kenneth', 'Lonergan', 'Male', '1962-10-16'),
    (15, 'Mel', 'Gibson', 'Male', '1956-01-03'),
    (16, 'Alejandro G.', 'Iñárritu', 'Male', '1963-08-15'),
    (17, 'George', 'Miller', 'Male', '1945-03-03'),
    (18, 'Lenny', 'Abrahamson', 'Male', '1960-11-30'),
    (19, 'Tom', 'McCarthy', 'Male', '1966-06-07'),
    (20, 'Steven', 'Spielberg', 'Male', '1940-12-18'),
    (21, 'Quentin', 'Tarantino', 'Male', '1963-03-27'),
    (22, 'Martin', 'Scorsese', 'Male', '1942-11-17'),
    (23, 'David', 'Fincher', 'Male', '1962-08-28'),
    (24, 'Wes', 'Anderson', 'Male', '1969-05-01'),
    (25, 'David O.', 'Russell', 'Male', '1958-08-20'),
    (26, 'Darren', 'Aronofsky', 'Male', '1969-02-12'),
    (27, 'Ridley', 'Scott', 'Male', '1937-11-30'),
    (28, 'Joel', 'Coen', 'Male', '1954-11-29'),
    (29, 'Ethan', 'Coen', 'Male', '1957-09-21'),
    (30, 'James', 'Cameron', 'Male', '1954-08-16'),
    (31, 'Ang', 'Lee', 'Male', '1954-10-23'),
    (32, 'Gus', 'Van Sant', 'Male', '1952-07-24'),
    (33, 'Peter', 'Jackson', 'Male', '1961-10-31'),
```

```sql
/* Actor in a Leading Role */
INSERT INTO Actor_Nominated
VALUES
    (27, 49, 1, 'The Theory of Everything', 1),
    (27, 49, 2, 'Interstellar', 0),
    (27, 50, 3, 'American Sniper', 0),
    (27, 51, 3, 'The Imitation Game', 0),
    (27, 52, 3, 'Birdman or (The Unexpected Virtue of Ignorance)', 0);

/* Actor in a Supporting Role*/
INSERT INTO Actor_Nominated
VALUES
    (27, 53, 3, 'Whiplash', 0),
    (27, 54, 3, 'The Judge', 0),
    (27, 55, 3, 'Boyhood', 0),
    (27, 56, 3, 'Birdman or (The Unexpected Virtue of Ignorance)', 0),
    (27, 57, 3, 'Interstellar', 0);

/* Actress in a Leading Roles*/
INSERT INTO Actor_Nominated
VALUES
    (27, 58, 4, 'Still Alice', 0),
    (27, 59, 4, 'Two Days, One Night', 0),
    (27, 60, 4, 'The Theory of Everything', 0),
    (27, 61, 4, 'Gone Girl', 0),
    (27, 62, 4, 'Wild', 0);

/* Actress in a Supporting Roles*/
INSERT INTO Actor_Nominated
VALUES
    (27, 63, 4, 'Boyhood', 1),
    (27, 64, 4, 'Wild', 0),
    (27, 65, 4, 'The Imitation Game', 0),
    (27, 66, 4, 'Birdman or (The Unexpected Virtue of Ignorance)', 0),
    (27, 67, 4, 'Into the Woods', 0);

/* Actor in a Leading Roles*/
INSERT INTO Actor_Nominated
VALUES
    (28, 27, 1, 'The Pianist', 1),
    (28, 28, 4, 'Frida', 0),
    (28, 29, 1, 'The Martian', 0),
    (28, 30, 1, 'Steve Jobs', 0),
    (28, 31, 1, 'The Danish Girl', 0);

/* Actor in a Supporting Roles*/
INSERT INTO Actor_Nominated
VALUES
    (28, 32, 1, 'Bridge of Spies', 1),
    (28, 33, 1, 'The Big Short', 0),
    (28, 34, 1, 'Bridge of Spies', 0),
    (28, 35, 1, 'Creed', 0),
    (28, 36, 1, 'Creed', 0);
```

```sql
#movies


INSERT INTO Movies
VALUES
    (1, 'Birdman or (The Unexpected Virtue of Ignorance)', 2014, 'Comedy', 10),
    (2, 'American Sniper', 2014, 'Action', 30),
    (3, 'Boyhood', 2014, 'Drama', 42),
    (4, 'The Grand Budapest Hotel', 2014, 'Comedy', 24),
    (5, 'The Imitation Game', 2014, 'Biography', 43),
    (6, 'Selma', 2014, 'Biography', 44),
    (7, 'The Theory of Everything', 2014, 'Biography', 45),
    (8, 'Whiplash', 2014, 'Drama', 12),
    (9, 'Spotlight', 2015, 'Drama', 19),
    (10, 'The Big Short', 2015, 'Biography', 3),
    (11, 'Bridge of Spies', 2015, 'Drama', 20),
    (12, 'Brooklyn', 2015, 'Drama', 40),
    (13, 'Mad Max: Fury Road', 2015, 'Action', 17),
    (14, 'The Martian', 2015, 'Adventure', 27),
    (15, 'The Revenant', 2015, 'Adventure', 10),
    (16, 'Room', 2015, 'Drama', 18),
    (17, 'Moonlight', 2016, 'Drama', 11),
    (18, 'Arrival', 2016, 'Sci-Fi', 13),
    (19, 'Fences', 2016, 'Drama', 38),
    (20, 'Hacksaw Ridge', 2016, 'Biography', 15),
    (21, 'Hell or High Water', 2016, 'Crime', 48),
    (22, 'Hidden Figures', 2016, 'Biography', 49),
    (23, 'La La Land', 2016, 'Comedy', 12),
    (24, 'Lion', 2016, 'Biography', 33),
    (25, 'Manchester by the Sea', 2016, 'Drama', 14),
    (26, 'The Shape of Water', 2017, 'Adventure', 7),
    (27, 'Call Me by Your Name', 2017, 'Drama', 51),
    (28, 'Darkest Hour', 2017, 'Biography', 52),
```

```
mysql> source values.sql
Query OK, 11 rows affected (0.05 sec)
Records: 11  Duplicates: 0  Warnings: 0

Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0

Query OK, 24 rows affected (0.01 sec)
Records: 24  Duplicates: 0  Warnings: 0

Query OK, 173 rows affected (0.01 sec)
Records: 173  Duplicates: 0  Warnings: 0

Query OK, 75 rows affected (0.01 sec)
Records: 75  Duplicates: 0  Warnings: 0

Query OK, 59 rows affected (0.01 sec)
Records: 59  Duplicates: 0  Warnings: 0

Query OK, 169 rows affected (0.02 sec)
Records: 169  Duplicates: 0  Warnings: 0

Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0

Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Use of the database:

Queries

1) The first query should select the first name and last name of all the actors and actresses whose first name starts with an “E”, like this I would know all the actors whose name starts with an “E” in my database.

```sql
/* 1 Select the actor ID and names of the actors that their name starts with an E*/
SELECT actorID, CONCAT(firstName, ' ', lastName) AS Full_Name
FROM Actors
WHERE firstName LIKE 'E%';
```

```
+---------+----------------+
| actorID | Full_Name      |
+---------+----------------+
|       2 | Emma Watson    |
|      10 | Emily Blunt    |
|      48 | Eddie Redmayne |
|      55 | Ethan Hawke    |
|      56 | Edward Norton  |
|      66 | Emma Stone     |
|      87 | Ellar Coltrane |
|     103 | Emory Cohen    |
+---------+----------------+
8 rows in set (0.00 sec)
```

2) The second query should select all the movies that were released from 2015 to 2017.

```
/*  2 select all the movie that got released between 2016 and 2018*/

SELECT *
FROM Movies
WHERE year BETWEEN 2015 AND 2017;
```

movieID	movieName	year	genre	directorID
9	Spotlight	2015	Drama	19
10	The Big Short	2015	Biography	3
11	Bridge of Spies	2015	Drama	20
12	Brooklyn	2015	Drama	46
13	Mad Max: Fury Road	2015	Action	17
14	The Martian	2015	Adventure	27
15	The Revenant	2015	Adventure	16
16	Room	2015	Drama	18
17	Moonlight	2016	Drama	11
18	Arrival	2016	Sci-Fi	13
19	Fences	2016	Drama	30
20	Hacksaw Ridge	2016	Biography	15
21	Hell or High Water	2016	Crime	48
22	Hidden Figures	2016	Biography	49
23	La La Land	2016	Comedy	12
24	Lion	2016	Biography	33
25	Manchester by the Sea	2016	Drama	14
26	The Shape of Water	2017	Adventure	7
27	Call Me by Your Name	2017	Drama	51
28	Darkest Hour	2017	Biography	52
29	Dunkirk	2017	Action	10
30	Get Out	2017	Horror	8
31	Lady Bird	2017	Comedy	6
32	Phantom Thread	2017	Drama	9
33	The Post	2017	Biography	20
34	Three Billboards Outside Ebbing, Missouri	2017	Crime	53
35	The Hateful Eight	2015	Crime	21
36	The Jungle Book	2016	Adventure	57
37	Sicario	2015	Action	13
38	Creed	2015	Drama	55
39	Ex Machina	2015	Sci-Fi	56

3) The third query should select the actor or actress with the most nominees in these four years.

```
/*  3 select the actor with the most nominees  */

SELECT CONCAT(firstName, ' ', lastName) AS Full_Name,
    (SELECT COUNT(*)
     FROM Actor_Nominated b
     WHERE b.actorID = a.actorID
    ) AS num_nominations
FROM
    Actors a
WHERE
    (SELECT COUNT(*)
     FROM Actor_Nominated c
     WHERE c.actorID = a.actorID
    ) >= ALL (
        SELECT COUNT(*)
        FROM Actor_Nominated
        GROUP BY actorID
    );
```

Full_Name	num_nominations
Meryl Streep	3

1 row in set (0.01 sec)

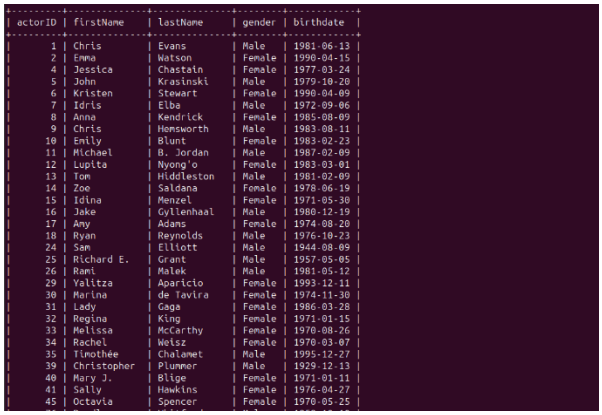
4) The fourth query should select all the directors that are below the age of 50 years old, a lot of the famous directors are a bit on the older side, so with this query I expect that we will not have many names.

```sql
/* 4 select all the directors below the age of 50*/

SELECT
    DirectorID,
    CONCAT(firstName, ' ', lastName) AS Full_Name,
    BirthDate,
    TRUNCATE(DATEDIFF(CURDATE(), BirthDate) / 365, 0) AS Age
FROM
    Directors
WHERE
    DATEDIFF(CURDATE(), BirthDate) / 365 < 50
ORDER BY
    Age;
```

```
+------------+-----------------+------------+------+
| DirectorID | Full_Name       | BirthDate  | Age  |
+------------+-----------------+------------+------+
|         55 | Ryan Coogler    | 1986-05-23 |   37 |
|         12 | Damien Chazelle | 1985-01-19 |   39 |
|          6 | Greta Gerwig    | 1983-08-04 |   40 |
|         11 | Barry Jenkins   | 1979-11-19 |   44 |
|          8 | Jordan Peele    | 1979-02-21 |   45 |
|         70 | Jeff Nichols    | 1978-12-07 |   45 |
|         71 | Pablo Larrain   | 1976-08-19 |   47 |
|         50 | Garth Davis     | 1974-07-26 |   49 |
+------------+-----------------+------------+------+
8 rows in set (0.00 sec)
```

5) The fifth query should select all the actors and actresses that have never been nominated for an Oscars from 2015 to 2018 and they are present in the database.

```sql
/* 5 select all the actors that have never been nominated for an Oscar*/

SELECT *
FROM Actors
WHERE actorID NOT IN (
    SELECT DISTINCT actorID
    FROM Actor_Nominated
);
```



actorID	firstName	lastName	gender	birthdate
1	Chris	Evans	Male	1981-06-13
2	Emma	Watson	Female	1990-04-15
4	Jessica	Chastain	Female	1977-03-24
5	John	Krasinski	Male	1979-10-20
6	Kristen	Stewart	Female	1990-04-09
7	Idris	Elba	Male	1972-09-06
8	Anna	Kendrick	Female	1985-08-09
9	Chris	Hemsworth	Male	1983-08-11
10	Emily	Blunt	Female	1983-02-23
11	Michael	B. Jordan	Male	1987-02-09
12	Lupita	Nyong'o	Female	1983-03-01
13	Tom	Hiddleston	Male	1981-02-09
14	Zoe	Saldana	Female	1978-06-19
15	Idina	Menzel	Female	1971-05-30
16	Jake	Gyllenhaal	Male	1980-12-19
17	Amy	Adams	Female	1974-08-20
18	Ryan	Reynolds	Male	1976-10-23
24	Sam	Elliott	Male	1944-08-09
25	Richard E.	Grant	Male	1957-05-05
26	Rami	Malek	Male	1981-05-12
29	Yalitza	Aparicio	Female	1993-12-11
30	Marina	de Tavira	Female	1974-11-30
31	Lady	Gaga	Female	1986-03-28
32	Regina	King	Female	1971-01-15
33	Melissa	McCarthy	Female	1970-08-26
34	Rachel	Weisz	Female	1970-03-07
35	Timothée	Chalamet	Male	1995-12-27
39	Christopher	Plummer	Male	1929-12-13
40	Mary J.	Blige	Female	1971-01-11
41	Sally	Hawkins	Female	1976-04-27
45	Octavia	Spencer	Female	1970-05-25

6) The sixth query should select the number of male actors and the number of female actresses that are present in the database, with this query we will be able to compare the difference.

```sql
/* 6 select the number of male actors and the number of female actresses*/

SELECT 'Male Actors' AS gender, COUNT(*) AS count_actors
FROM Actors
WHERE Gender = 'Male'
UNION
SELECT 'Female Actresses' AS gender, COUNT(*) AS count_actors
FROM Actors
WHERE Gender = 'Female';
```

```
+------------------+--------------+
| gender           | count_actors |
+------------------+--------------+
| Male Actors      |           98 |
| Female Actresses |           75 |
+------------------+--------------+
2 rows in set (0.00 sec)
```

7) The seventh query should select all the Oscars winners for a leading role in a movie from 2015 to 2018.

```sql
/*  7 select all the Oscars winners for leading role and for which movie */
SELECT CONCAT(a.FirstName, ' ', a.LastName) AS Full_Name, an.movieName
FROM Actor_Nominated AS an
NATURAL JOIN Actors AS a
WHERE an.won = 1
AND (an.categoryID = 3 OR an.categoryID = 4);
```

```
+-------------------+-----------------------------------------+
| Full_Name         | movieName                               |
+-------------------+-----------------------------------------+
| Eddie Redmayne    | The Theory of Everything                |
| Julianne Moore    | Still Alice                             |
| Brie Larson       | Room                                    |
| Leonardo DiCaprio | The Revenant                            |
| Casey Affleck     | Manchester by the Sea                   |
| Emma Stone        | La La Land                              |
| Gary Oldman       | Darkest Hour                            |
| Frances McDormand | Three Billboards outside Ebbing, Missouri |
+-------------------+-----------------------------------------+
```

8) The eighth query should select all the actors that have on the movies that have won the Oscars from 2015 to 2018.

```sql
/*  8 select all the actors that have worked on best picture movies */

SELECT a.firstName, a.lastName, m.MovieName
FROM Actors AS a
INNER JOIN Acts ON a.actorID = Acts.actorID
INNER JOIN Movie_Nominated AS mn ON Acts.movieID = mn.movieID
INNER JOIN Movies AS m ON mn.movieID = m.movieID
WHERE mn.won = 1;
```

```
+-----------+----------+---------------------------------------------+
| firstName | lastName | MovieName                                   |
+-----------+----------+---------------------------------------------+
| Michael   | Keaton   | Birdman or (The Unexpected Virtue of Ignorance) |
| Edward    | Norton   | Birdman or (The Unexpected Virtue of Ignorance) |
| Emma      | Stone    | Birdman or (The Unexpected Virtue of Ignorance) |
| Michael   | Keaton   | Spotlight                                   |
| Mark      | Ruffalo  | Spotlight                                   |
| Rachel    | McAdams  | Spotlight                                   |
| Mahershala | Ali     | Moonlight                                   |
| Naomie    | Harris   | Moonlight                                   |
| Richard   | Jenkins  | The Shape of Water                          |
| Sally     | Hawkins  | The Shape of Water                          |
| Octavia   | Spencer  | The Shape of Water                          |
| Michael   | Shannon  | The Shape of Water                          |
+-----------+----------+---------------------------------------------+
12 rows in set (0.00 sec)
```

9) The ninth query should select all the hosts that have hosted the Oscars more than once from 2015 to 2018

```sql
/* 9 select the hosts that have hosted the Oscars more than once */

SELECT CONCAT(h.firstName, ' ', h.lastName) AS Full_Name
FROM Host h
INNER JOIN Oscars o ON h.hostID = o.hostID
GROUP BY h.firstName, h.lastName
HAVING COUNT(*) > 1;
```

```
+--------------+
| Full_Name    |
+--------------+
| Jimmy Kimmel |
+--------------+
1 row in set (0.00 sec)
```

10) The tenth query should select all the directors and the movies that they have directed and if a director did not direct any movie from the movies in the database, then it would be padded with NULL.

```sql
/*  10 select all the directors, the movies they have directed, and put NULL if they have not directed any movies */

SELECT d.directorID, CONCAT(d.firstName, ' ', d.lastName) AS Full_Name, m.movieID, m.movieName
FROM Directors d
LEFT OUTER JOIN Movies m ON m.directorID = d.directorID;
```

directorID	Full_Name	movieID	movieName
1	Alfonso Cuarón	NULL	NULL
2	Spike Lee	NULL	NULL
3	Adam McKay	10	The Big Short
4	Yorgos Lanthimos	NULL	NULL
5	Paweł Pawlikowski	NULL	NULL
6	Greta Gerwig	31	Lady Bird
7	Guillermo del Toro	26	The Shape of Water
8	Jordan Peele	30	Get Out
9	Paul Thomas Anderson	32	Phantom Thread
10	Christopher Nolan	29	Dunkirk
11	Barry Jenkins	17	Moonlight
12	Damien Chazelle	8	Whiplash
12	Damien Chazelle	23	La La Land
13	Denis Villeneuve	18	Arrival
13	Denis Villeneuve	37	Sicario
14	Kenneth Lonergan	25	Manchester by the Sea
15	Mel Gibson	20	Hacksaw Ridge
16	Alejandro G. Iñárritu	1	Birdman or (The Unexpected Virtue of Ignorance)
16	Alejandro G. Iñárritu	15	The Revenant
17	George Miller	13	Mad Max: Fury Road
18	Lenny Abrahamson	16	Room
19	Tom McCarthy	9	Spotlight
20	Steven Spielberg	11	Bridge of Spies
20	Steven Spielberg	33	The Post
21	Quentin Tarantino	35	The Hateful Eight
22	Martin Scorsese	NULL	NULL
23	David Fincher	43	Gone Girl
24	Wes Anderson	4	The Grand Budapest Hotel
25	David O. Russell	50	Joy
26	Darren Aronofsky	NULL	NULL
27	Ridley Scott	14	The Martian

Advanced concepts:

The first procedure called AddDirector will take the first name, last name, gender, and date of birth of the director that the user wants to add to the database and to the director's table. To call the procedure you type
CALL AddDirector(firstName, lastName, gender, birthDate);

```sql
/*  procedure that inserts a new director,
to call it, type CALL AddActor("firstName", "lastName", "gender", "YYYY-MM-DD"); */

DROP PROCEDURE IF EXISTS AddDirector;
DELIMITER //

CREATE PROCEDURE AddDirector(
    IN p_firstName VARCHAR(50),
    IN p_lastName VARCHAR(50),
    IN p_gender VARCHAR(10),
    IN p_birthDate DATE
)
BEGIN
        DECLARE nextID INT;
        SELECT MAX(directorID) + 1 FROM Directors INTO nextID;
    INSERT INTO Directors (directorID,firstName, lastName, gender, birthDate)
    VALUES (nextID,p_firstName, p_lastName, p_gender, p_birthDate);
END //

DELIMITER ;
```

```
mysql> CALL AddDirector("Mohamed","Ahmed","Male","2005-08-05");
Query OK, 1 row affected (0.02 sec)

mysql> SELECT * FROM Directors WHERE firstName = "Mohamed";
+------------+-----------+----------+--------+------------+
| directorID | firstName | lastName | gender | birthdate  |
+------------+-----------+----------+--------+------------+
|         76 | Mohamed   | Ahmed    | Male   | 2005-08-05 |
+------------+-----------+----------+--------+------------+
1 row in set (0.00 sec)
```

The second procedure called GetActorNominationCount will take the actorID and a global variable that will save the result of the number of total nominations of the specific actor that the user has given his or her actorID. To call the procedure you type
CALL GetActorNominationCount (actorID, @Result);

```
/* This procedure retrieves the total number of nominations for an actor or an actress
to call it, type CALL GetActorNominationCount(actorID, @Result);
SELCET @Result; */

DROP PROCEDURE IF EXISTS GetActorNominationCount;

DELIMITER //

CREATE PROCEDURE GetActorNominationCount(
    IN p_actorID INT,
    OUT p_nominationCount INT
)
BEGIN
    SELECT COUNT(*) INTO p_nominationCount
    FROM Actor_Nominated
    WHERE actorID = p_actorID;
END //

DELIMITER ;
```

```
mysql> SET @Result=0
    -> ;
Query OK, 0 rows affected (0.00 sec)

mysql> CALL GetActorNominationCount(66, @Result);
Query OK, 1 row affected (0.02 sec)

mysql> SELECT @Result;
+---------+
| @Result |
+---------+
|       2 |
+---------+
1 row in set (0.00 sec)
```

The third procedure called GetNomineesByYear, this will take the Oscars year and produce all the nominees for that year for Best Leading Actor, Best Supporting Actor, Best Leading Actress, and Best Supporting Actress. To call the procedure you type

CALL GetNomineesByYear (year);

```
/*procedure that retrieves all the actors nominated for an Oscar in a specific year
to call it, type CALL GetNomineesByYear(year); */

DROP PROCEDURE IF EXISTS GetNomineesByYear;
DELIMITER //
CREATE PROCEDURE GetNomineesByYear(
    IN p_year INT
)
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE actorID INT;
    DECLARE actorName VARCHAR(255);


    DECLARE actorCursor CURSOR FOR
        SELECT a.actorID, CONCAT(a.firstName, ' ', a.lastName) AS actorName
        FROM Actors a
        INNER JOIN Actor_Nominated an ON a.actorID = an.actorID
        INNER JOIN Oscars o ON an.oscarsID = o.oscarsID
        WHERE o.year = p_year;


    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;


    OPEN actorCursor;


    actor_loop: LOOP
        FETCH actorCursor INTO actorID, actorName;
        IF done THEN
            LEAVE actor_loop;
        END IF;


        SELECT actorID, actorName;
    END LOOP;


    CLOSE actorCursor;
END //

DELIMITER ;
```

```
mysql> CALL GetNomineesByYear(2015);
+---------+-------------+
| actorID | actorName   |
+---------+-------------+
|      46 | Meryl Streep |
+---------+-------------+
1 row in set (0.00 sec)

+---------+---------------+
| actorID | actorName     |
+---------+---------------+
|      48 | Eddie Redmayne |
+---------+---------------+
1 row in set (0.00 sec)

+---------+-------------+
| actorID | actorName   |
+---------+-------------+
|      49 | Steve Carell |
+---------+-------------+
1 row in set (0.00 sec)

+---------+---------------+
| actorID | actorName     |
+---------+---------------+
|      50 | Bradley Cooper |
+---------+---------------+
1 row in set (0.00 sec)

+---------+---------------------+
| actorID | actorName           |
+---------+---------------------+
```

The first view called ActorWinners, will show all the Oscars winners from 2015 to 2018

```sql
/* view that retrieves all the actors Oscars winners,
to call it, example: SELECT * FROM ActorWinners; */
DROP VIEW IF EXISTS ActorWinners;
CREATE VIEW ActorWinners AS
SELECT
    an.oscarsID AS OscarsID,
    an.actorID AS ActorID,
    CONCAT(a.firstName, ' ', a.lastName) AS ActorName,
    an.movieName AS MovieName
FROM
    Actor_Nominated an
    INNER JOIN Actors a ON an.actorID = a.actorID
WHERE
    an.won = 1;
```

```
mysql> SELECT * FROM ActorWinners;
+----------+---------+------------------+--------------------------------------+
| OscarsID | ActorID | ActorName        | MovieName                            |
+----------+---------+------------------+--------------------------------------+
|       87 |      48 | Eddie Redmayne   | The Theory of Everything             |
|       87 |      53 | J.K. Simmons     | Whiplash                             |
|       87 |      58 | Julianne Moore   | Still Alice                          |
|       87 |      63 | Patricia Arquette | Boyhood                             |
|       88 |      19 | Brie Larson      | Room                                 |
|       88 |      67 | Leonardo DiCaprio | The Revenant                        |
|       88 |     100 | Mark Rylance     | Bridge of Spies                      |
|       88 |     137 | Alicia Vikander  | The Danish Girl                      |
|       89 |      21 | Mahershala Ali   | Moonlight                            |
|       89 |      47 | Casey Affleck    | Manchester by the Sea                |
|       89 |      66 | Emma Stone       | La La Land                           |
|       89 |      71 | Viola Davis      | Fences                               |
|       90 |      28 | Sam Rockwell     | Three Billboards outside Ebbing, Missouri |
|       90 |      42 | Allison Janney   | I, Tonya                             |
|       90 |      74 | Gary Oldman      | Darkest Hour                         |
|       90 |      78 | Frances McDormand | Three Billboards outside Ebbing, Missouri |
+----------+---------+------------------+--------------------------------------+
16 rows in set (0.00 sec)
```

The second view called HostsOfOscar, will show all the hosts that have hosted the Oscars from 2015 to 2018

```sql
/* view to display the hosts who hosted the Oscars,
to call it, example: SELECT * FROM HostsOfOscars; */
DROP VIEW IF EXISTS HostsOfOscars;
CREATE VIEW HostsOfOscars AS
SELECT
    o.oscarsID AS OscarsID,
    h.hostID AS HostID,
    CONCAT(h.firstName, ' ', h.lastName) AS HostName,
    o.year AS OscarsYear
FROM
    Oscars o
    INNER JOIN Host h ON o.hostID = h.hostID;
```

```
mysql> SELECT * FROM HostsOfOscars;
+----------+--------+--------------------+------------+
| OscarsID | HostID | HostName           | OscarsYear |
+----------+--------+--------------------+------------+
|       87 |      1 | Neil Patrick Harris |       2015 |
|       88 |      2 | Chris Rock         |       2016 |
|       89 |      3 | Jimmy Kimmel       |       2017 |
|       90 |      3 | Jimmy Kimmel       |       2018 |
+----------+--------+--------------------+------------+
4 rows in set (0.01 sec)
```

The first trigger called ActorNominatedCheckWonTrigger, is trigged if one of the tuples in the Actor-Nominated table is updated, and the won attribute is changed to a number other than 0 or 1, then this trigger is triggered, and it produces an error message and sets the won attribute back to the Old Won.

```
/* Trigger that makes sure that won for actors is either 0 or 1 */

DELIMITER //
DROP TRIGGER IF EXISTS ActorNominatedCheckWonTrigger;
CREATE TRIGGER ActorNominatedCheckWonTrigger
BEFORE UPDATE ON Actor_Nominated
FOR EACH ROW
BEGIN
    IF NEW.won NOT IN (0, 1) THEN
        SET NEW.won = OLD.won;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: The "won" column must be either 0 or 1.';
    END IF;
END //

DELIMITER ;
```

The second trigger called DirectorNominatedCheckWonTrigger, is trigged if one of the tuples in the Director-Nominated table is updated, and the won attribute is changed to a number other than 0 or 1, then this trigger is triggered, and it produces an error message and sets the won attribute back to the OLD. Won.

```
/* Trigger that makes sure that won for directors is either 0 or 1 */

DELIMITER //
DROP TRIGGER IF EXISTS DirectorNominatedCheckWonTrigger;
CREATE TRIGGER DirectorNominatedCheckWonTrigger
BEFORE UPDATE ON Director_Nominated
FOR EACH ROW
BEGIN
    IF NEW.won NOT IN (0, 1) THEN
        SET NEW.won = OLD.won;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: The "won" column must be either 0 or 1.';
    END IF;
END //

DELIMITER ;
```

The third trigger called MovieNominatedCheckWonTrigger, is trigged if one of the tuples in the Movie-Nominated table is updated, and the won attribute is changed to a number other than 0 or 1, then this trigger is triggered, and it produces an error message and sets the won attribute back to the OLD. Won.

```sql
/* Trigger that makes sure that won for movies is either 0 or 1 */

DELIMITER //
DROP TRIGGER IF EXISTS MoviesNominatedCheckWonTrigger;
CREATE TRIGGER MoviesNominatedCheckWonTrigger
BEFORE UPDATE ON Movie_Nominated
FOR EACH ROW
BEGIN
    IF NEW.won NOT IN (0, 1) THEN
        SET NEW.won = OLD.won;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: The "won" column must be either 0 or 1.';
    END IF;
END //

DELIMITER ;
```

```
mysql> UPDATE Actor_Nominated SET won = 2 WHERE actorID = 66;
ERROR 1644 (45000): Error: The "won" column must be either 0 or 1.
mysql> UPDATE Director_Nominated SET won = -4 WHERE directorID = 3;
ERROR 1644 (45000): Error: The "won" column must be either 0 or 1.
mysql> UPDATE Movie_Nominated SET won = 10 WHERE movieID = 5;
ERROR 1644 (45000): Error: The "won" column must be either 0 or 1.
mysql>
```

Python Connector:

There are three files connecting to Python one for running all the defined queries, one for adding an actor and one for updating the birthdate of a director.

The first file called PythonQueries.py, will run all queries that were defined before.

```
cursor4 = conn.cursor()

# Define the fourth query and data
print()
print('Fourth Query')
query4 = ("""SELECT
    DirectorID,
    CONCAT(FirstName, ' ', LastName) AS Full_Name,
    BirthDate,
    TRUNCATE(DATEDIFF(CURDATE(), BirthDate) / %s,%s) AS Age
    FROM
    Directors
    WHERE
    DATEDIFF(CURDATE(), BirthDate) / %s < %s
    ORDER BY
    Age""")

data4 = (365, 0, 365, 50)
# Execute the fourth query
cursor4.execute(query4, data4)
# Fetch the results
result4 = cursor4.fetchall()
for row in result4:
    print(row)

# Close the fourth cursor and connection
cursor4.close()


cursor5 = conn.cursor()

# Define the fifth query and data
print()
print('Fifth Query')
query5 = ("""SELECT *
FROM Actors
    WHERE actorID NOT IN (
        SELECT DISTINCT actorID
        FROM Actor_Nominated
)""")
# Execute the fifth query
cursor5.execute(query5)
# Fetch the results
result5 = cursor5.fetchall()
for row in result5:
    print(row)

# Close the fifth cursor and connection
cursor5.close()
```

```
cursor6 = conn.cursor()
# Define the sixth query and data
print()
print('Sixth Query')
query6 = ("""SELECT %s AS gender, COUNT(*) AS count_actors
    FROM Actors
    WHERE Gender = %s
    UNION
    SELECT %s AS gender, COUNT(*) AS count_actors
    FROM Actors
    WHERE Gender = %s""")

data6 = ('Male Actors','Male', 'Female Actresses','Female')
# Execute the sixth query
cursor6.execute(query6, data6)
# Fetch the results
result6 = cursor6.fetchall()
for row in result6:
    print(row)

# Close the sixth cursor and connection
cursor6.close()



cursor7 = conn.cursor()
# Define the seventh query and data
print()
print('Seventh Query')
query7 = ("""SELECT CONCAT(a.FirstName, ' ', a.LastName) AS Full_Name, an.movieName
    FROM Actor_Nominated AS an
    NATURAL JOIN Actors AS a
    WHERE an.won = %s
    AND (an.categoryID = %s OR an.categoryID = %s)""")

data7 = (1,1, 4)
# Execute the seventh query
cursor7.execute(query7, data7)
# Fetch the results
result7 = cursor7.fetchall()
for row in result7:
    print(row)

# Close the seventh cursor and connection
cursor7.close()



cursor8 = conn.cursor()
# Define the eighth query and data
print()
print('Eighth Query')
query8 = ("""SELECT a.FirstName, a.LastName, m.MovieName
    FROM Actors AS a
    INNER JOIN Acts ON a.actorID = Acts.actorID
    INNER JOIN Movies AS m ON Acts.movieID = m.movieID
    WHERE m.won = %s """)

data8 = (1,)
# Execute the eighth query
cursor8.execute(query8, data8)
# Fetch the results
result8 = cursor8.fetchall()
for row in result8:
    print(row)

# Close the eighth cursor and connection
cursor8.close()
```

```
cursor9 = conn.cursor()
# Define the ninth query and data
print()
print('Ninth Query')
query9 = ("""SELECT CONCAT(h.FirstName, ' ', h.LastName) AS Full_Name
    FROM Host h
    INNER JOIN Oscars o ON h.hostID = o.hostID
    GROUP BY h.FirstName, h.LastName
    HAVING COUNT(*) > %s """)

data9 = (1,)
# Execute the ninth query
cursor9.execute(query9, data9)
# Fetch the results
result9 = cursor9.fetchall()
for row in result9:
    print(row)

# Close the ninth cursor and connection
cursor9.close()



cursor10 = conn.cursor()
# Define the tenth query and data
print()
print('Tenth Query')
query10 = ("""SELECT d.directorID, CONCAT(d.FirstName, ' ', d.LastName) AS Full_Name, m.movieID, m.movieName
    FROM Directors d
    LEFT OUTER JOIN Movies m ON m.directorID = d.directorID;""")

# Execute the tenth query
cursor10.execute(query10)
# Fetch the results
result10 = cursor10.fetchall()
for row in result10:
    print(row)

# Close the tenth cursor and connection
cursor10.close()



conn.close()
```

```
mohamed@mohamed-VirtualBox:~/Desktop/DBS/project$ python3 PythonQueries.py

First Query
(2, 'Emma Watson')
(10, 'Emily Blunt')
(48, 'Eddie Redmayne')
(55, 'Ethan Hawke')
(56, 'Edward Norton')
(66, 'Emma Stone')
(87, 'Ellar Coltrane')
(103, 'Emory Cohen')

Second Query
(9, 'Spotlight', 2015, 'Drama', 19)
(10, 'The Big Short', 2015, 'Biography', 3)
(11, 'Bridge of Spies', 2015, 'Drama', 20)
(12, 'Brooklyn', 2015, 'Drama', 46)
(13, 'Mad Max: Fury Road', 2015, 'Action', 17)
(14, 'The Martian', 2015, 'Adventure', 27)
(15, 'The Revenant', 2015, 'Adventure', 16)
(16, 'Room', 2015, 'Drama', 18)
(17, 'Moonlight', 2016, 'Drama', 11)
(18, 'Arrival', 2016, 'Sci-Fi', 13)
(19, 'Fences', 2016, 'Drama', 30)
(20, 'Hacksaw Ridge', 2016, 'Biography', 15)
(21, 'Hell or High Water', 2016, 'Crime', 48)
(22, 'Hidden Figures', 2016, 'Biography', 49)
```

The second file called PythonAdding where it asks the user to input the information for a new actor, and then it adds it to the actors' table.

```python
import mysql.connector

# Establish connection
conn = mysql.connector.connect(user='me', password='myUserPassword', database='Oscars_21445116')

cursorTemp = conn.cursor()
tempQuery = "SELECT MAX(actorID) + 1 FROM Actors"
cursorTemp.execute(tempQuery)
result = cursorTemp.fetchone()[0]  # Fetch the result
cursorTemp.close()

cursor = conn.cursor()

# Prompt user for actor details
print("Adding Query")
print("Enter the first name of the actor:")
firstName = input()
print("Enter the last name of the actor:")
lastName = input()
print("Enter the gender of the actor:")
gender = input()
print("Enter the birth date of the actor (YYYY-MM-DD):")
birthDate = input()

# Define the query to insert actor
query = "INSERT INTO Actors (actorID, firstName, lastName, gender, birthDate) VALUES (%s, %s, %s, %s, %s)"
data = (result, firstName, lastName, gender, birthDate)

# Execute the insert query
cursor.execute(query, data)
conn.commit()

# Fetch the newly inserted actor's information
query_result = "SELECT * FROM Actors WHERE actorID = %s"
data_result = (result,)
cursor.execute(query_result, data_result)
actor_info = cursor.fetchone()
print("Newly added actor details:")
print(actor_info)

# Close the cursor and connection
cursor.close()
conn.close()
```

```
mohamed@mohamed-VirtualBox:~/Desktop/DBS/project$ python3 PythonAdding.py
Adding Query
Enter the first name of the actor:
Mohamed
Enter the last name of the actor:
Harby
Enter the gender of the actor:
Male
Enter the birth date of the actor (YYYY-MM-DD):
2005-08-05
Newly added actor details:
(174, 'Mohamed', 'Harby', 'Male', datetime.date(2005, 8, 5))
mohamed@mohamed-VirtualBox:~/Desktop/DBS/project$
```

The third file called PythonUpdating where it asks the user to input a director's ID and a new birthdate, and the program will update that director's birthdate to the new one given.

```python
import mysql.connector

# Establish connection
conn = mysql.connector.connect(user='me', password='myUserPassword', database='Oscars_21445116')

cursor = conn.cursor()

print("Enter the Director ID you want to update their birth Date:")
DirectorID = input()
print("Enter the updated birth date (YYYY-MM-DD):")
BirthDate = input()

# Define the UPDATE query
query = "UPDATE Directors SET BirthDate = %s WHERE DirectorID = %s"
data = (BirthDate, DirectorID)

# Execute the UPDATE query
cursor.execute(query, data)
conn.commit()

# Fetch and print the newly updated director information
query_result = "SELECT * FROM Directors WHERE DirectorID = %s"
data_result = (DirectorID,)
cursor.execute(query_result, data_result)
director_info = cursor.fetchone()
print("Newly updated director birth date:")
print(director_info)

# Close cursor and connection
cursor.close()
conn.close()
```

```
mohamed@mohamed-VirtualBox:~/Desktop/DBS/project$ python3 PythonUpdating.py
Enter the Director ID you want to update their birth Date:
23
Enter the updated birth date (YYYY-MM-DD):
1965-06-19
Newly updated director birth date:
(23, 'David', 'Fincher', 'Male', datetime.date(1965, 6, 19))
mohamed@mohamed-VirtualBox:~/Desktop/DBS/project$
```

Business Rules:

Business rule	Description
1) business rule	A movie should only have one director.
2) business rule	A movie should have at least one actor.
3) business rule	Every Oscars should only have one host.
4) business rule	There can only be one winner for every category

Discussion:

Since the entire assignment depends on the creation of the entities, this step took the most thought. I had to ensure that all the relationships were accurate and made sense. After that, creating the cardinality and participation tables was simple because I had a lot of practice due to the midterm. Next, creating the data description tables and the relational scheme was also easy.

Using the data description as a guide, it was then simple to create the tables in MySQL. The most time-consuming part of the process was gathering the data, as I had to double-check that all the primary and foreign keys were where they belonged and that the data was correct. This was necessary to ensure that the queries that I would write should produce the desired actual results.

Finding genuine primary keys required a lot of time as well. However, after researching, I discovered that the information was confidential and that I would not be able to access it.

The process of creating the queries was also straightforward. However, I had intended to write procedures and triggers exclusively for the advanced functions, but when I was writing the triggers, I was unable to locate more than one scenario based on my database, so in addition to writing code for that scenario, I also created two views.

Finally, since I was accustomed to writing in Java, connecting with Python took some time, but it was not too tough.

The things I could do better include adding more categories and tables, such as those for producers, animated films, and editors. I also believe there is a faster method for inserting data from a CSV file into the table, which could be useful.

References:

Kaggle. (n.d.). The Oscar Award. Retrieved from https://www.kaggle.com/datasets/unanimad/the-oscar-award

The Academy of Motion Picture Arts and Sciences. (n.d.). Every Oscar Host in History: See the Full List. Retrieved from https://aframe.oscars.org/news/post/every-oscar-host-in-history-see-the-full-list

Wikipedia. (n.d.). Academy Awards. In Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Academy_Awards

Wikipedia. (n.d.). List of actors with Academy Award nominations. In Wikipedia. Retrieved from https://en.wikipedia.org/wiki/List_of_actors_with_Academy_Award_nominations

Wikipedia. (n.d.). Academy Award for Best Director. In Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Academy_Award_for_Best_Director

IMDb. (n.d.). Academy Award Winning Actors. Retrieved from https://www.imdb.com/list/ls066864774/

The Academy of Motion Picture Arts and Sciences. (2015). Oscars Ceremonies - 2015. Retrieved from https://www.oscars.org/oscars/ceremonies/2015

The Academy of Motion Picture Arts and Sciences. (2016). Oscars Ceremonies - 2016. Retrieved from https://www.oscars.org/oscars/ceremonies/2016

The Academy of Motion Picture Arts and Sciences. (2017). Oscars Ceremonies - 2017. Retrieved from https://www.oscars.org/oscars/ceremonies/2017

The Academy of Motion Picture Arts and Sciences. (2018). Oscars Ceremonies - 2018. Retrieved from https://www.oscars.org/oscars/ceremonies/2018