A
Project Report
On

Flex Movie Streaming Database Management System

Developed by

Jayswal Tirth – Department of IT, DD University

Guided By
Internal Guide:
Ms. Roshni M. Raval
Department of Information Technology
Faculty of Technology
DD University

Department of Information Technology
Faculty of Technology, Dharmsinh Desai University
College Road, Nadiad-387001
October-2019

DHARMSINH DESAI UNIVERSITY
NADIAD-387001, GUJARAT

CERTIFICATE

This is to certify that the project entitled "**Flex Movie Streaming Database Management System"** is a bona fide report of the work carried out by **Mr. Jayswal Tirth, Student ID No: 17ITUBS051** of Department of Information Technology, semester V, under the guidance and supervision for the subject Database Management System. They were involved in Project training during academic year 2019-2020.

Prof. Roshni M. Raval
(Project Guide)
Department of Information Technology,
Faculty of Technology,
Dharmsinh Desai University, Nadiad
Date:

Prof. Vipul Dabhi
Head , Department of Information Technology
Faculty of Technology,
Dharmsinh Desai University, Nadiad
Date:

ACKNOWLEDGEMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and we are extremely privileged to have got this all along the completion of this project.

We owe our deep gratitude to our project guide Prof. Roshni M Raval, who took keen interest on our project work and guided us all along, till the completion of our project work by providing all the necessary information for developing a good Database System.

We are thankful to and fortunate enough to get constant encouragement, support and guidance from all Teaching staffs of IT Department which helped us in successfully completing our project work.

Thanking You,
Jayswal Tirth B. (IT 032)

TABLE OF CONTENTS

1. SYSTEM OVERVIEW

1.1 CURRENT SYSYTEM:

- The Movie Database project is to categorize and catalog every single piece of movie from the movie information from IMDB
- The idea of movie database arose out of common interest of movie. Today movie does a great influence on us. It not only freshens our mood but also inspires us.
- Movie Database was developed by taking the ideas from movie applications like Google Play Movies, Amazon Prime Videos.
- The Movie Details are the Most Important unit of our database. They are categorized in title, release year, budget, gross earnings, IMDB ratings; etc .You can also find your favorite director or artist's information and Movies associated with them.
- In Current System, anyone can Surf through the details of movies, directors, artists and movie purchase details.
- If user want to watch movie he/she have to create an account and then they can watch movie by paying specific amount for that movie .User can Watch that movie only for given period of time which is provided in purchase details.
- There are some movies that are FREE to watch

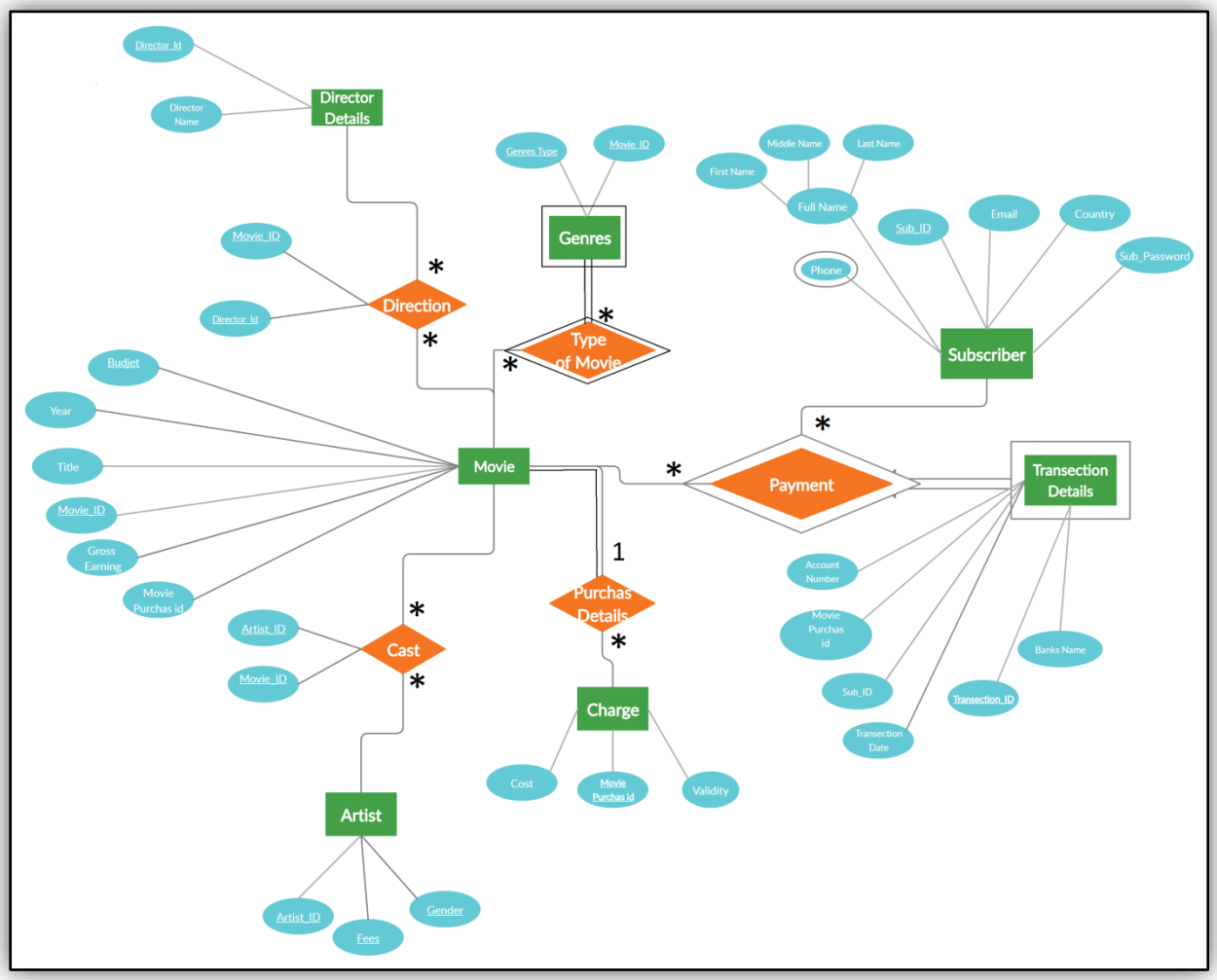
1.2 OBJECTIVES OF PROPOSED SYSTEM:

- Users can watch the latest and trending movies.
- Grouping movies into genres so that similar movie can be watched.
- User can watch any number of movies for given time period.
- Movie, Director, Artist details can be update.
- Only account holding users can watch movies by logging into their account.
- User can login with user id and password.

1.3 ADVANTAGES OF PROPOSED SYSTEM:

- Movie Subscription is automatically removed from user account, after the validity of that movie is over. User can't watch that movie anymore.
- Before the validity is over, users get the Notification email that your movie subscription will over soon.
- Whenever a new movie is available in our database system account holding user get the notification email.

2. ER Diagram


Director_Id
Director Name
Director Details
Genres_Type
Movie_ID
Genres
Middle Name
Last Name
First Name
Full Name
Sub_ID
Email
Country
Phone
Sub_Password
Subscriber
Movie_ID
Director_Id
Direction
Type of Movie
Budjet
Year
Title
Movie_ID
Gross Earning
Movie Purchas id
Movie
Payment
Transection Details
Account Number
Movie Purchas id
Sub_ID
Transection_ID
Transection Date
Banks Name
1
Purchas Details
Artist_ID
Movie_ID
Cast
Charge
Cost
Movie Purchas Id
Validity
Artist
Artist_ID
Fees
Gender

2.1 Entities:

1. Artist
2. Cast
3. Movie
4. Director Details
5. Subscriber
6. Transaction
7. Charge
8. Genres

2.2 Relationships & Mapping Constraints:

1. Direction
    - Movie Entity Have many to many relationship with Director details Entity
2. Cast
    - Artist Entity have many to many relationship with Movie Entity

3. Purchase Details
    - Movie Entity have one to one relationship with Charge Entity
    - Movie Entity have total Participation in this relationship
    - Charge Entity have one to many relationship with Movie Entity

4. Type of Movie
    - Genres is weak Entity set and have total participation in this relationship
    - Identifying Relationship

5. Payment
    - Movie Entity Have one to many relationship with Transaction Entity
    - Subscriber Entity have one to many relationship with Transaction Entity
    - Transaction is weak entity set and have total participation in Payment relationship
    - Identifying Relationship

3. DATA DICTIONARY

```
mysql> desc Transaction
+------------------+------------+------+-----+---------+-------+
| Field            | Type       | Null | Key | Default | Extra |
+------------------+------------+------+-----+---------+-------+
| Transaction_id   | varchar(5) | NO   | PRI | NULL    |       |
| Account_Number   | varchar(50)| YES  |     | NULL    |       |
| Banks_Name       | varchar(30)| YES  |     | NULL    |       |
| Sub_ID           | varchar(5) | YES  | MUL | NULL    |       |
| Movie_id         | varchar(5) | YES  | MUL | NULL    |       |
| Transection_Date | date       | YES  |     | NULL    |       |
+------------------+------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

```
mysql> desc Subscriber;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| Sub_ID       | varchar(5)   | NO   | PRI | NULL    |       |
| Sub_Password | varchar(15)  | YES  |     | NULL    |       |
| Phone        | varchar(50)  | YES  |     | NULL    |       |
| Email        | varchar(40)  | YES  |     | NULL    |       |
| Full_Name    | varchar(30)  | YES  |     | NULL    |       |
| Country      | varchar(110) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

```
mysql> desc Genres;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| Geners_Type | varchar(55) | NO   | PRI | NULL    |       |
| Movie_ID    | varchar(5)  | NO   | PRI | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> desc direction;
+----------+------------+------+-----+---------+-------+
| Field    | Type       | Null | Key | Default | Extra |
+----------+------------+------+-----+---------+-------+
| Dir_ID   | varchar(5) | NO   | PRI | NULL    |       |
| Movie_ID | varchar(5) | NO   | PRI | NULL    |       |
+----------+------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> desc Movie;
+-----------------+------------+------+-----+---------+-------+
| Field           | Type       | Null | Key | Default | Extra |
+-----------------+------------+------+-----+---------+-------+
| Movie_ID        | varchar(5)  | NO   | PRI | NULL    |       |
| Title           | varchar(60) | YES  |     | NULL    |       |
| Year            | year(4)     | YES  |     | NULL    |       |
| IMDB_Score      | double      | YES  |     | NULL    |       |
| Budget          | bigint(20)  | YES  |     | 0       |       |
| Gross_Earnings  | bigint(20)  | YES  |     | 0       |       |
| Movie_Purchas_id | varchar(5)  | YES  | MUL | NULL    |       |
+-----------------+------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

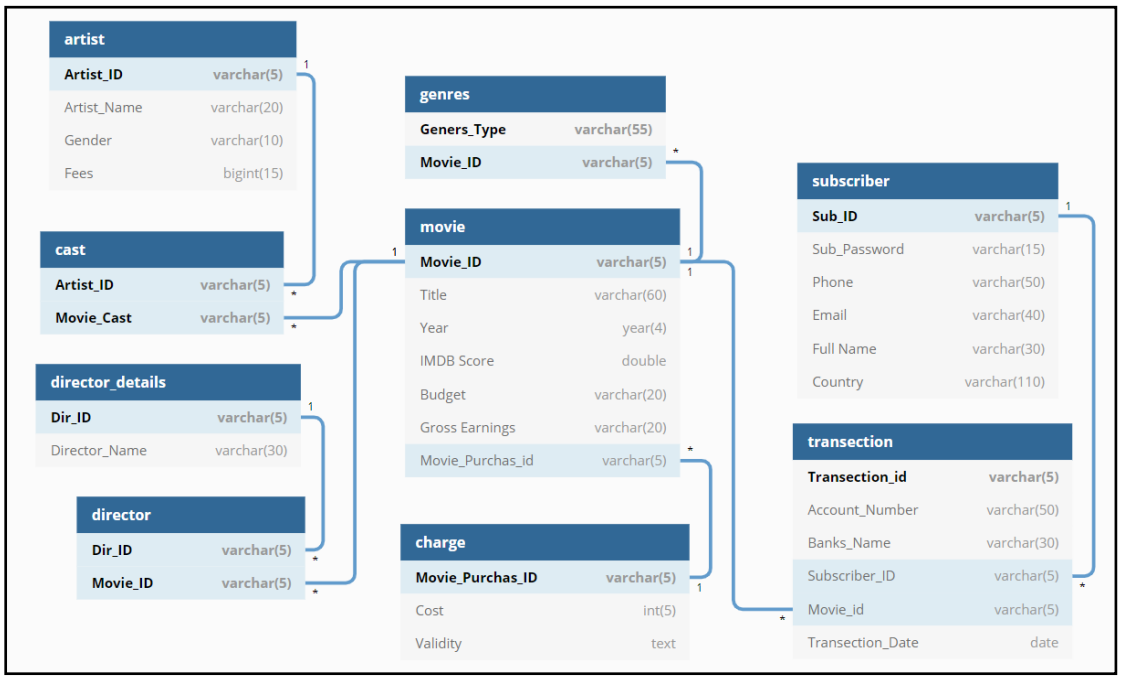
```
mysql> desc Charge;
+-----------------+-----------+------+-----+---------+-------+
| Field           | Type      | Null | Key | Default | Extra |
+-----------------+-----------+------+-----+---------+-------+
| Movie_Purchas_ID | varchar(5) | NO   | PRI | NULL    |       |
| Cost            | int(5)     | YES  |     | NULL    |       |
| Validity        | text       | YES  |     | NULL    |       |
+-----------------+-----------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

```
mysql> desc Cast;
+------------+-----------+------+-----+---------+-------+
| Field      | Type      | Null | Key | Default | Extra |
+------------+-----------+------+-----+---------+-------+
| Artist_ID  | varchar(5) | NO   | PRI | NULL    |       |
| Movie_Cast | varchar(5) | NO   | PRI | NULL    |       |
+------------+-----------+------+-----+---------+-------+
2 rows in set (0.01 sec)
```

```
mysql> desc Artist;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| Artist_ID   | varchar(5)  | NO   | PRI | NULL    |       |
| Artist_Name | varchar(20) | YES  |     | NULL    |       |
| Gender      | varchar(10) | YES  |     | NULL    |       |
| Fees        | bigint(15)  | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

```
mysql> desc Director_Details;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| Dir_ID        | varchar(5)  | NO   | PRI | NULL    |       |
| Director_Name | varchar(30) | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

4. SCHEMA DIAGRAM



5. DATABASE IMPLEMENTATION

5.1 CREATE TABLES:

1. Artist

```
CREATE TABLE `artist` (
`Artist_ID` varchar(5) NOT NULL,
`Artist_Name` varchar(20) DEFAULT NULL,
`Gender` varchar(10) DEFAULT NULL,
`Fees` bigint(15) DEFAULT NULL,
PRIMARY KEY (`Artist_ID`)
) ;
```

2. Cast

```
CREATE TABLE `cast` (
`Artist_ID` varchar(5) NOT NULL,
`Movie_Cast` varchar(5) NOT NULL,
PRIMARY KEY (`Artist_ID`,`Movie_Cast`),
CONSTRAINT `FK_Artist_Table` FOREIGN KEY
   (`Artist_ID`) REFERENCES `artist` (`Artist_ID`) ON
   DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT `FK_Movie_Table` FOREIGN KEY
   (`Movie_Cast`) REFERENCES `movie` (`Movie_ID`)
   ON DELETE CASCADE ON UPDATE CASCADE
) ;
```

3. Charge

```
CREATE TABLE `charge` (
`Movie_Purchas_ID` varchar(5) NOT NULL,
`Cost` int(5) DEFAULT NULL,
`Validity` text,
PRIMARY KEY (`Movie_Purchas_ID`)
) ;
```

4. Director

```
CREATE TABLE `director` (
`Dir_ID` varchar(5) NOT NULL,
`Movie_ID` varchar(5) NOT NULL,
```

```
PRIMARY KEY (`Dir_ID`,`Movie_ID`),
CONSTRAINT `FK_Director_Table` FOREIGN KEY (`Dir_ID`) REFERENCES `director_details` (`Dir_ID`) ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT `FK_Movie_t` FOREIGN KEY (`Movie_ID`) REFERENCES `movie` (`Movie_ID`) ON DELETE CASCADE ON UPDATE CASCADE
) ;
```

5. Director_Details

```
CREATE TABLE `director_details` (
`Dir_ID` varchar(5) NOT NULL,
`Director_Name` varchar(30) DEFAULT NULL,
PRIMARY KEY (`Dir_ID`)
) ;
```

6. Genres

```
CREATE TABLE `genres` (
`Geners_Type` varchar(55) NOT NULL,
`Movie_ID` varchar(5) NOT NULL,
PRIMARY KEY (`Geners_Type`,`Movie_ID`),
CONSTRAINT `FK_Geners_Movie` FOREIGN KEY (`Movie_ID`) REFERENCES `movie` (`Movie_ID`) ON DELETE CASCADE ON UPDATE CASCADE
) ;
```

7. Movie

```
CREATE TABLE `movie` (
`Movie_ID` varchar(5) NOT NULL,
`Title` varchar(60) DEFAULT NULL,
`Year` year(4) DEFAULT NULL,
`IMDB_Score` double DEFAULT NULL,
`Budget` bigint(20) DEFAULT '0',
`Gross_Earnings` bigint(20) DEFAULT '0',
`Movie_Purchas_id` varchar(5) DEFAULT NULL,
PRIMARY KEY (`Movie_ID`),
CONSTRAINT `FK_Charge` FOREIGN KEY (`Movie_Purchas_id`) REFERENCES `charge` (`Movie_Purchas_ID`) ON DELETE CASCADE ON UPDATE CASCADE
```

```
) ;
```

8. Subscriber

```
CREATE TABLE `subscriber` (
`Sub_ID` varchar(5) NOT NULL,
`Sub_Password` varchar(15) DEFAULT NULL,
`Phone` varchar(50) DEFAULT NULL,
`Email` varchar(40) DEFAULT NULL,
`Full_Name` varchar(30) DEFAULT NULL,
`Country` varchar(110) DEFAULT NULL,
PRIMARY KEY (`Sub_ID`)
) ;
```

9. Transaction

```
CREATE TABLE `transaction` (
`Transaction_id` varchar(5) NOT NULL,
`Account_Number` varchar(50) DEFAULT NULL,
`Banks_Name` varchar(30) DEFAULT NULL,
`Sub_ID` varchar(5) DEFAULT NULL,
`Movie_id` varchar(5) DEFAULT NULL,
`Transection_Date` date DEFAULT NULL,
PRIMARY KEY (`Transection_id`),
CONSTRAINT `FK_Sub` FOREIGN KEY (`Sub_ID`)
   REFERENCES `subscriber` (`Sub_ID`) ON DELETE
   CASCADE ON UPDATE CASCADE,
CONSTRAINT `FK_Tran_Movie` FOREIGN KEY
   (`Movie_id`) REFERENCES `movie` (`Movie_ID`)
) ;
```

5.2 INSERT DATA:

1. **Artist**

```
INSERT INTO `finaldbms`.`artist`
(`Artist_ID`,
`Artist_Name`,
`Gender`,
`Fees`)
VALUES
(<{Artist_ID: }>,<{Artist_Name: }>,<{Gender: }>,<{Fees: }>);
```

```
mysql> select * from artist;
+--------------------+----------+---------+
| Artist_Name        | Gender   | Fees    |
+--------------------+----------+---------+
| Alan D. Purwin     | "Male"   | 1012893 |
| Alan Rickman       | "Male"   | 3932702 |
| Anne Hathaway      | "Female" | 5374896 |
| Ayelet Zurer       | "Female" | 4393870 |
| Barry Bostwick     | "Male"   | 4908056 |
| Bradley Cooper     | "Male"   | 7690028 |
| Bree Williamson    | "Male"   | 4101664 |
| Chris Evans        | "Male"   | 8115610 |
| David Costabile    | "Male"   | 2744753 |
| Dustin Ingram      | "Male"   | 2573193 |
| Emmanuel Kabongo   | "Female" | 8504173 |
```

2. **Cast**

```
INSERT INTO `finaldbms`.`cast`
(`Artist_ID`,
`Movie_Cast`)
VALUES
(<{Artist_ID: }>,
<{Movie_Cast: }>);
```

```
mysql> select * from Cast;
+--------------------+------------+
| Artist_Name        | Movie_Cast |
+--------------------+------------+
| Bradley Cooper     | M001       |
| Jada Pinkett Smith | M001       |
| John Gallagher Jr. | M001       |
| Sumalee Montano    | M001       |
| Alan D. Purwin     | M002       |
| David Costabile    | M002       |
| James Badge Dale   | M002       |
| Toby Stephens      | M002       |
| Bree Williamson    | M003       |
```

3. Direction

```
INSERT INTO `finaldbms`.`direction`
(`Dir_ID`,
`Movie_ID`)
VALUES
(<{Dir_ID: }>,
<{Movie_ID: }>);
```

```
mysql> select * from direction;
+--------+----------+
| Dir_ID | Movie_ID |
+--------+----------+
| D001   | M001     |
| D002   | M002     |
| D003   | M003     |
| D004   | M004     |
| D005   | M005     |
| D006   | M006     |
| D007   | M007     |
| D008   | M008     |
| D009   | M009     |
| D010   | M010     |
```

4. Director_Details

```
INSERT INTO `finaldbms`.`director_details`
(`Dir_ID`,
`Director_Name`)
VALUES
(<{Dir_ID: }>,
<{Director_Name: }>);
```

```
mysql> select * from Director_Details;
+--------+------------------------+
| Dir_ID | Director_Name          |
+--------+------------------------+
| D001   | Dan Trachtenberg       |
| D002   | Michael Bay            |
| D003   | Mitchell Altieri       |
| D004   | Raja Menon             |
| D005   | James Bobin            |
| D006   | Robert Schwentke       |
| D007   | Darren Lynn Bousman    |
| D008   | Danny Perez            |
| D009   | Jon Lucas              |
| D010   | Zack Snyder            |
```

5. Genres

```
INSERT INTO `finaldbms`.`genres`
(`Geners_Type`,
`Movie_ID`)
VALUES
(<{Geners_Type: }>,
<{Movie_ID: }>);
```

```
mysql> select * from genres;
+-------------+----------+
| Geners_Type | Movie_ID |
+-------------+----------+
| Biography   | M001     |
| Drama       | M001     |
| Horror      | M001     |
| Sci-Fi      | M001     |
| Action      | M002     |
| Comedy      | M002     |
| Drama       | M002     |
| Sport       | M002     |
| Action      | M003     |
| Adventure   | M003     |
```

6. Transection

```
INSERT INTO `finaldbms`.`transection`
(`Transection_id`,
`Account_Number`,
`Banks_Name`,
`Sub_ID`,
`Movie_id`,
`Transection_Date`)
VALUES
(<{Transection_id:}>,<{Account_Number:
}>,<{Banks_Name:}>,
<{Sub_ID:}>,<{Movie_id:}>,<{Transection_Date: }>);
```

```
mysql> select * from Transection;
+---------------+-------------------------+-------------------------+--------+----------+------------------+
| Transection_id | Account_Number          | Banks_Name              | Sub_ID | Movie_id | Transection_Date |
+---------------+-------------------------+-------------------------+--------+----------+------------------+
| T001          | 5,010,000,000,000,000   | Bank of Baroda          | S024   | M001     | 2019-10-11       |
| T002          | 3,550,000,000,000,000   | Bank of India           | S025   | M002     | 2019-09-15       |
| T003          | 372,000,000,000,000     | Bank of Maharashtra     | S026   | M003     | 2019-10-14       |
| T004          | 202,000,000,000,000     | Canara Bank             | S027   | M004     | 2019-10-04       |
| T005          | 3,560,000,000,000,000   | Central Bank of India   | S028   | M005     | 2019-09-30       |
| T006          | 5,100,000,000,000,000   | Corporation Bank        | S029   | M006     | 2019-09-27       |
| T007          | 4,910,000,000,000,000   | Dena Bank               | S030   | M007     | 2019-10-06       |
| T008          | 5,050,000,000,000,000   | Indian Bank             | S031   | M008     | 2019-10-09       |
| T009          | 633,000,000,000,000,000 | Indian Overseas Bank    | S032   | M009     | 2019-10-02       |
| T010          | 3,560,000,000,000,000   | IDBI Bank               | S033   | M010     | 2019-10-04       |
```

7. Movie

```
INSERT INTO `finaldbms`.`movie`
(`Movie_ID`,
`Title`,`Year`,`IMDB_Score`,`Budget`,
`Gross_Earnings`,`Movie_Purchas_id`)
VALUES(<{Movie_ID:        }>,<{Title:        }>,<{Year:
}>,<{IMDB_Score: }>,<{Budget: 0}>,
<{Gross_Earnings: 0}>,<{Movie_Purchas_id: }>);
```

```
mysql> select * from Movie;
+----------+-------------------------------------------------+------+------------+----------+---------------+-----------------+
| Movie_ID | Title                                           | Year | IMDB_Score | Budget   | Gross_Earnings | Movie_Purchas_id |
+----------+-------------------------------------------------+------+------------+----------+---------------+-----------------+
| M001     | 10 Cloverfield Lane                             | 2016 |        7.3 | 15000000 |      71897215 | MP001           |
| M002     | 13 Hours                                        | 2016 |        7.4 | 50000000 |      52822418 | MP002           |
| M003     | A Beginner's Guide to Snuff                     | 2016 |        8.7 |        0 |             0 | MP003           |
| M004     | Airlift                                         | 2016 |        8.5 |        0 |             0 | MP004           |
| M005     | Alice Through the Looking Glass                 | 2016 |        6.4 |        0 |             0 | MP005           |
| M006     | Allegiant                                       | 2016 |        5.8 |        0 |             0 | MP006           |
| M007     | Alleluia! The Devil's Carnival                  | 2016 |        7.4 |        0 |             0 | MP007           |
| M008     | Antibirth                                       | 2016 |        6.3 |        0 |             0 | MP007           |
| M009     | Bad Moms                                        | 2016 |        6.7 | 20000000 |      55461307 | MP007           |
| M010     | Batman v Superman: Dawn of Justice              | 2016 |        6.9 |        0 |             0 | MP007           |
```

8. Subscriber

```
INSERT INTO `finaldbms`.`subscriber`
(`Sub_ID`,
`Sub_Password`,
`Phone`,
`Email`,
`Full_Name`,
`Country`)
VALUES
(<{Sub_ID: }>,
<{Sub_Password: }>,
<{Phone: }>,
<{Email: }>,
<{Full_Name: }>,
<{Country: }>);
```

```
mysql> select * from $ubscriber;
+--------+----------------------+----------------------------------+---------------------+-------------------------------+-------------+
| Sub_Id | Sub_Name             | Sub_Country                      | Sub_Phone           | Sub_Email                     | Sub_Password |
+--------+----------------------+----------------------------------+---------------------+-------------------------------+-------------+
| S001   | Miss Al Hills        | French Polynesia                 | (947)556-4565 x429  | Aida_OConner@bernard.ca       | afdU4t95nQ   |
| S002   | Claudine Renner      | Chile                            | 1-529-776-1430      | Archibald_Carroll@santiago.me | vmiJoWE8Qz   |
| S003   | Mrs. Arnold Fadel    | Panama                           | 1-012-234-1176 x30526 | Danika@bart.net             | KZ22Io2K     |
| S004   | Clement Fisher MD    | Denmark                          | (270)162-5452 x202  | Eleanora@josephine.us         | OdNyzY       |
| S005   | Justyn Gleichner V   | Lesotho                          | 1-042-962-5709      | Eric@tate.biz                 | 0qdSmc6M9w$$ |
| S006   | Joesph Emmerich      | Bulgaria                         | (734)544-6259 x6710 | Gwendolyn.Ratke@cheyanne.org  | JNkv6OEKJWfJ |
| S007   | Lucious Jast         | Portugal                         | 1-964-146-5737 x212 | Joanny@nayeli.info            | GazyeLwB0    |
| S008   | Corene Reichert      | Suriname                         | 829-083-9396        | Stacy@virginia.biz            | qFMBzW       |
| S009   | Elian Murray         | Guinea-Bissau                    | 388-982-8981 x6574  | Abigayle_Kuhic@maddison.io    | fb4chBnj     |
| S010   | Adolf Anderson       | Djibouti                         | 1-394-923-9212 x3666 | Brooke@river.me              | 8CbuQv7ZY    |
```

9. Charge

```
INSERT INTO `finaldbms`.`charge`
(`Movie_Purchas_ID`,
`Cost`,
`Validity`)
VALUES
(<{Movie_Purchas_ID: }>,
<{Cost: }>,
<{Validity: }>);
```

```
mysql> select * from charge;
+------------------+------+----------+
| Movie_Purchas_ID | Cost | Validity |
+------------------+------+----------+
| MP001            |  250 | 90       |
| MP002            |  100 | 60       |
| MP003            |  150 | 75       |
| MP004            |   50 | 30       |
| MP005            |   25 | 10       |
| MP006            |  500 | Lifetime |
| MP007            |    0 | Lifetime |
+------------------+------+----------+
7 rows in set (0.00 sec)
```

5.3 Queries

1. List out all the movie title whose genre is “Action”.

```
mysql> select m.Title as "Action Movie" from movie m inner join genres g on m.movie_id=g.movie_id where g.geners_type="Action";
+-----------------------------------------------+
| Action Movie                                  |
+-----------------------------------------------+
| 13 Hours                                      |
| A Beginner's Guide to Snuff                   |
| Batman v Superman: Dawn of Justice            |
| Cabin Fever                                   |
| Fight Valley                                  |
| Keanu                                         |
| Operation Chromite                            |
| Teenage Mutant Ninja Turtles: Out of the Shadows  |
| X-Men: Apocalypse                             |
+-----------------------------------------------+
9 rows in set (0.00 sec)
```

2. List Out all the movie which is directed by “Anthony Russo”.

```
mysql> select m.title,m.movie_id from director d inner join movie m on m.movie_id=d.movie_id where d.Dir_ID=(select d.dir_id from director_Det
irector_Name="Anthony Russo");
+---------------------------+----------+
| title                     | movie_id |
+---------------------------+----------+
| Captain America: Civil War  | M013     |
| Nerve                     | M054     |
+---------------------------+----------+
2 rows in set (0.00 sec)
```

3. List out Name of the Artists who was part of Movie titled “Cabin Fever”.

```
mysql> select a.Artist_Name from artist a inner join cast c on a.Artist_id=c.Artist_id where c.movie_cast=(select m.movie_id from movie m where m
.Title="Cabin Fever ");
+----------------+
| Artist_Name    |
+----------------+
| Johnny Depp    |
| Mila Kunis     |
| Dustin Ingram  |
| Sameer Ali Khan |
| Gage Golightly |
| Samuel Davis   |
+----------------+
6 rows in set (0.03 sec)
```

4. List out title and genres of movies which is directed by “Travis Zariwny”

```
mysql> select g.Geners_Type,m.title from genres g inner join  movie m on g.movie_id=m.movie_id inner join director d on d.movie_id=m.movie_id whe
re d.dir_id=(select dir_id from director_details where director_name="Travis Zariwny");
+------------+----------------------------+
| Geners_Type | title                      |
+------------+----------------------------+
| Action     | Cabin Fever                |
| War        | Cabin Fever                |
| Crime      | Neighbors 2: Sorority Rising |
+------------+----------------------------+
3 rows in set (0.00 sec)
```

5. List out the movie details based on ascending order of gross earnings

```
mysql> select * from movie m where m.Gross_Earnings <>0 order by m.Gross_Earnings;
+----------+---------------------------------+------+------------+----------+---------------+-----------------+
| Movie_ID | Title                           | Year | IMDB_Score | Budget   | Gross_Earnings | Movie_Purchas_id |
+----------+---------------------------------+------+------------+----------+---------------+-----------------+
| M056     | Operation Chromite              | 2016 |        6.8 | 12620000 |         31662 | MP007           |
| M084     | The Masked Saint                | 2016 |        4.7 |  3500000 |        123777 | MP001           |
| M085     | The Neon Demon                  | 2016 |          7 |  7000000 |       1330827 | MP002           |
| M038     | Jane Got a Gun                  | 2016 |        5.8 | 25000000 |       1512815 | MP007           |
| M016     | Compadres                       | 2016 |          5 |  3000000 |       3105269 | MP006           |
| M047     | Midnight Special                | 2016 |        6.7 | 18000000 |       3707794 | MP007           |
| M092     | The Young Messiah               | 2016 |        5.4 | 18500000 |       6462576 | MP002           |
| M086     | The Perfect Match               | 2016 |        4.5 |  5000000 |       9658370 | MP003           |
| M059     | Pride and Prejudice and Zombies | 2016 |        5.8 | 28000000 |      10907291 | MP002           |
| M022     | Fifty Shades of Black           | 2016 |        3.5 |  5000000 |      11675178 | MP001           |
| M093     | Triple 9                        | 2016 |        6.3 | 20000000 |      12626905 | MP003           |
| M017     | Criminal                        | 2016 |        6.3 | 31500000 |      14268533 | MP007           |
| M080     | The Infiltrator                 | 2016 |        7.3 | 25000000 |      14946229 | MP005           |
| M020     | Eddie the Eagle                 | 2016 |        7.5 | 23000000 |      15785632 | MP007           |
| M021     | Eddie the Eagle                 | 2016 |        7.5 | 23000000 |      15785632 | MP007           |
| M026     | Free State of Jones             | 2016 |        6.7 | 50000000 |      20389967 | MP005           |
| M040     | Keanu                           | 2016 |        6.4 | 15000000 |      20566327 | MP001           |
| M028     | God's Not Dead 2                | 2016 |        3.4 |  5000000 |      20773070 | MP007           |
| M078     | The Forest                      | 2016 |        4.8 | 10000000 |      26583369 | MP003           |
| M098     | Zoolander 2                     | 2016 |        4.8 | 50000000 |      28837115 | MP007           |
| M054     | Nerve                           | 2016 |        7.1 | 20000000 |      28876924 | MP007           |
| M031     | Hail, Caesar!                   | 2016 |        6.4 | 22000000 |      29997095 | MP001           |
| M069     | The 5th Wave                    | 2016 |        5.2 | 38000000 |      34912982 | MP007           |
| M019     | Dirty Grandpa                   | 2016 |          6 | 11500000 |      35537564 | MP007           |
| M074     | The Boy                         | 2016 |          6 | 10000000 |      35794166 | MP007           |
| M063     | Risen                           | 2016 |        6.3 | 20000000 |      36874745 | MP006           |
| M050     | Money Monster                   | 2016 |        6.7 | 27000000 |      41008532 | MP007           |
| M033     | How to Be Single                | 2016 |        6.1 | 38000000 |      46813366 | MP003           |
| M002     | 13 Hours                        | 2016 |        7.4 | 50000000 |      52822418 | MP002           |
| M089     | The Shallows                    | 2016 |        6.8 | 17000000 |      54257433 | MP006           |
| M053     | Neighbors 2: Sorority Rising    | 2016 |          6 | 35000000 |      55291815 | MP007           |
| M009     | Bad Moms                        | 2016 |        6.7 | 20000000 |      55461307 | MP007           |
```

6. List out all the movie title of movies which is free to watch

```
mysql> select m.title from movie m inner join charge c using(Movie_Purchas_id) where c.cost=0;
+-------------------------------------------------+
| title                                           |
+-------------------------------------------------+
| Alleluia! The Devil's Carnival                  |
| Antibirth                                       |
| Bad Moms                                        |
| Batman v Superman: Dawn of Justice              |
| Criminal                                        |
| Deadpool                                        |
| Dirty Grandpa                                   |
| Eddie the Eagle                                 |
| Eddie the Eagle                                 |
| God's Not Dead 2                                |
| Gods of Egypt                                   |
| Godzilla Resurgence                             |
| Irreplaceable                                   |
| Jane Got a Gun                                  |
| Jason Bourne                                    |
| Me Before You                                   |
| Midnight Special                                |
| Miracles from Heaven                            |
| Misconduct                                      |
| Money Monster                                   |
| Mr. Church                                      |
| My Big Fat Greek Wedding 2                      |
| Neighbors 2: Sorority Rising                    |
| Nerve                                           |
| Now You See Me 2                                |
| Operation Chromite                              |
| Our Kind of Traitor                             |
| Rodeo Girl                                      |
| Sausage Party                                   |
| Star Trek Beyond                                |
| Suicide Squad                                   |
| Teenage Mutant Ninja Turtles: Out of the Shadows |
| The 5th Wave                                    |
| The Angry Birds Movie                           |
```

7. List out transaction details of customer whose name is “Pearlier Leffler”

```
mysql> select t.* from transection t inner join Subscriber s using(Sub_id) where s.Full_Name='Pearlie Leffler';
+--------------+----------------------+--------------------+--------+----------+------------------+
| Transection_id | Account_Number       | Banks_Name         | Sub_ID | Movie_id | Transection_Date |
+--------------+----------------------+--------------------+--------+----------+------------------+
| T009         | 633,000,000,000,000,000 | Indian Overseas Bank | S032 | M009     | 2019-10-02       |
| T034         | 5,610,000,000,000,000   | Bank of India        | S032 | M021     | 2019-09-29       |
| T037         | 3,540,000,000,000,000   | Central Bank of India | S032 | M024     | 2019-10-07       |
| T092         | 3,540,000,000,000,000   | UCO Bank             | S032 | M085     | 2019-10-01       |
+--------------+----------------------+--------------------+--------+----------+------------------+
4 rows in set (0.00 sec)
```

8. List Out Movie Details which is Purchased By Customer “Pearlier Leffler”

```
mysql> select m.* from movie m where m.Movie_ID in (select t.Movie_ID from transection t inner join Subscriber s using(Sub_id) where s.Full_Name=
'Pearlie Leffler');
+----------+-----------------+------+------------+----------+---------------+-----------------+
| Movie_ID | Title           | Year | IMDB_Score | Budget   | Gross_Earnings | Movie_Purchas_id |
+----------+-----------------+------+------------+----------+---------------+-----------------+
| M009     | Bad Moms        | 2016 |        6.7 | 20000000 |      55461307 | MP007           |
| M021     | Eddie the Eagle | 2016 |        7.5 | 23000000 |      15785632 | MP007           |
| M024     | Fight Valley    | 2016 |          5 |        0 |             0 | MP003           |
| M085     | The Neon Demon  | 2016 |          7 |  7000000 |       1330827 | MP002           |
+----------+-----------------+------+------------+----------+---------------+-----------------+
4 rows in set (0.00 sec)
```

9. Count how much money “Pearlier Leffler” spend on Purchasing movie and count number of Movie he had Purchased.

```
mysql> select count(m.Movie_ID), sum(c.cost) from charge c inner join movie m using(Movie_Purchas_ID) where m.Movie_ID in (select m.Movie_ID fro
 movie m where m.Movie_ID in (select t.Movie_ID from transection t inner join Subscriber s using(Sub_id) where s.Full_Name='Pearlie Leffler'));
+-----------------+------------+
| count(m.Movie_ID) | sum(c.cost) |
+-----------------+------------+
|               4 |        250 |
+-----------------+------------+
1 row in set (0.00 sec)
```

10. List out the movie details whose imdb score is greater than 8.

```
mysql> select * from movie where imdb_score>8;
+----------+-----------------------------+------+------------+--------+---------------+------------------+
| Movie_ID | Title                       | Year | IMDB_Score | Budget | Gross_Earnings | Movie_Purchas_id |
+----------+-----------------------------+------+------------+--------+---------------+------------------+
| M003     | A Beginner's Guide to Snuff | 2016 |        8.7 |      0 |             0 | MP003            |
| M004     | Airlift                     | 2016 |        8.5 |      0 |             0 | MP004            |
| M013     | Captain America: Civil War  | 2016 |        8.2 |      0 |             0 | MP003            |
| M018     | Deadpool                    | 2016 |        8.1 |      0 |             0 | MP007            |
| M030     | Godzilla Resurgence         | 2016 |        8.2 |      0 |             0 | MP007            |
| M041     | Kickboxer: Vengeance        | 2016 |        9.1 |      0 |             0 | MP002            |
+----------+-----------------------------+------+------------+--------+---------------+------------------+
```

5.4 PL / SQL Block (Exception Handling)

1. Given procedure is equivalent to insert query in transaction table.

   This Procedure is to demonstrate Inbuilt Exception with Modification of displaying appropriate message to user.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
    `System_Exception_ex`(IN `t_id` VARCHAR(7), IN `a_no`
    VARCHAR(30), IN `b_name` VARCHAR(30), IN `s_id` VARCHAR(7), IN
    `m_id` VARCHAR(7))
  NO SQL
BEGIN
  -- exit if Foreign key not Found
  DECLARE EXIT HANDLER FOR 1452
  BEGIN
   SELECT ('User does not Exist First Sign Up Then Try Again') AS "Error
     Message: Log IN";
   SELECT "" as "OR";
   SELECT ("Given Movie Is Not Available to Stream please try again
     later") as "Error Message: Movie Not Available to Stream";
  END;
  DECLARE EXIT HANDLER FOR 1062
  BEGIN
    SELECT ("Transectio ID is Already Present ") as "Error Message :\r\n
    Primary Key Constrain";
  end;

  INSERT INTO `transection`(`Transection_id`, `Account_Number`,
                    `Banks_Name`, `Sub_ID`, `Movie_id`,
    `Transection_Date`) VALUES
         (t_id,a_no,b_name,s_id,m_id,CURRENT_DATE);
  SELECT "Enjoy The High Quality Streaming" As "Success Message:
    Transection Executed";



end$$
DELIMITER ;
```

Case 1:- Transaction Successfully Executed Without any Exception.

```
SET @p0='T105'; SET @p1='4397575092702'; SET @p2='Central Bank OF india'; SET @p3='S025'; SET @p4='M099'; CALL
`System_Exception_ex`(@p0, @p1, @p2, @p3, @p4);
```

Execution results of routine `System_Exception_ex`

Message: Transection Successfully Executed
Enjoy The High Quality Streaming

Case 2:- Primary Key Constrain Violated

```
SET @p0='T100'; SET @p1='47384729749'; SET @p2='BOB'; SET @p3='S002'; SET @p4='M001'; CALL `System_Exception_ex`(@p0, @p1,
@p2, @p3, @p4);
```

Execution results of routine `System_Exception_ex`

Error Message : Primary Key Constrain
Transectio ID is Already Present

Case 3:- Foreign Key Constrain Violated

```
SET @p0='T106'; SET @p1='47274723894789'; SET @p2='BOB'; SET @p3='S999'; SET @p4='M999'; CALL `System_Exception_ex`(@p0,
@p1, @p2, @p3, @p4);
```

Execution results of routine `System_Exception_ex`

Error Message: Log IN
User does not Exist First Sign Up Then Try Again

OR

Error Message: Movie Not Available to Stream
Given Movie Is Not Available to Stream please try again later

2. User Defined Exception Demonstrate

If User wants to stream movie he/she have to provide user id, password and movie id using given procedure

"User_Defined_Exception(user_id,pass_word,movie_id)"

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `User_Defined_Exception_ex`(IN
    `User_id` VARCHAR(7), IN `Pass_word` VARCHAR(20), IN `Movie_id` VARCHAR(7))
  NO SQL
BEGIN
    DECLARE invalid_request CONDITION FOR SQLSTATE '22012';
    DECLARE invalid_login CONDITION FOR SQLSTATE '22011';

  DECLARE Exit HANDLER FOR  invalid_login

   begin
     RESIGNAL SET MESSAGE_TEXT = 'Invalid Log Info :- Check User id and  password ';
   end;

    DECLARE Exit HANDLER FOR invalid_request
  BEGIN
    RESIGNAL SET MESSAGE_TEXT = 'Invalid Movie Request :-You Have to Purchased
    First';
   end;

  if(check_log_info(User_id,Pass_word))
  THEN
    IF EXISTS (SELECT t1.Transection_id FROM transection t1 inner
        join transection t2 WHERE
                                        t1.Transection_id=t2.Transection_id and
                t1.Sub_ID=User_id and
    t2.Movie_id=Movie_id)
     THEN
          Select "Enjoy Your Movie ";

     ELSE
        SIGNAL invalid_request;
     END IF;
  ELSE
    SIGNAL invalid_login;

    END if;
END$$
DELIMITER ;
```

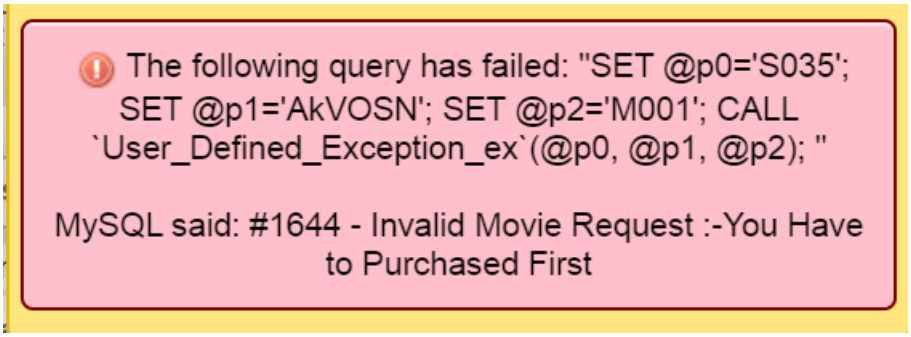
Case 1:- Log in Information is correct and user have purchased the given movie

```
SET @p0='S039'; SET @p1='M1Ac6P3'; SET @p2='M031'; CALL `User_Defined_Exception_ex`(@p0, @p1, @p2);
```

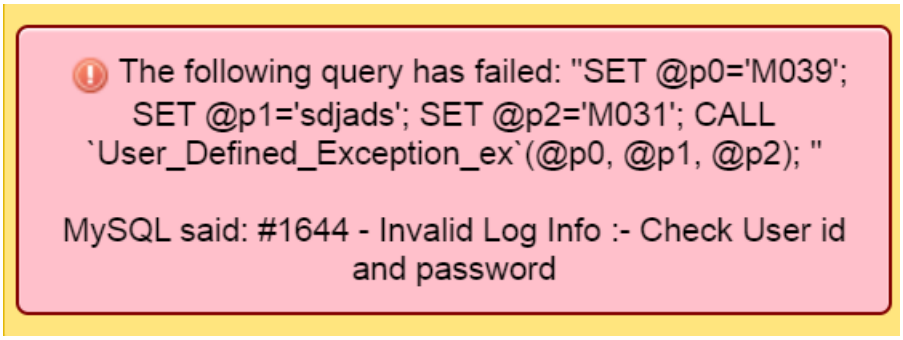
Execution results of routine `User_Defined_Exception_ex`

Enjoy Your Movie
Enjoy Your Movie

Case 2:- Log in Information is Correct and user did not purchased requested movie



Case 3:- Log in Information is Invalid



5.5 Views

1. Information OF Free Movies

```
CREATE view Free_Movies_details as SELECT
    m.Title,m.Year,m.IMDB_Score from Movie m INNER join
    charge c where
    c.Movie_Purchas_ID=m.Movie_Purchas_id and c.Cost=0


SELECT * FROM `free_movies_details`
```

Showing rows 0 - ... (Query took 0.0252 seconds.)

```
SELECT * FROM `free_movies_details`
```

	Title	Year	IMDB_Score
Edit Copy Delete	Alleluia! The Devil's Carnival?	2016	7.4
Edit Copy Delete	Antibirth?	2016	6.3
Edit Copy Delete	Bad Moms?	2016	6.7
Edit Copy Delete	Batman v Superman: Dawn of Justice?	2016	6.9
Edit Copy Delete	Criminal?	2016	6.3
Edit Copy Delete	Deadpool?	2016	8.1
Edit Copy Delete	Dirty Grandpa?	2016	6
Edit Copy Delete	Eddie the Eagle?	2016	7.5
Edit Copy Delete	Eddie the Eagle?	2016	7.5
Edit Copy Delete	God's Not Dead 2?	2016	3.4
Edit Copy Delete	Gods of Egypt?	2016	5.5

2. Information Of "Family" type Movies

```
CREATE view family_type_movies_details as SELECT
    m.Title,m.Year,m.IMDB_Score from Movie m INNER join genres
    g where g.Movie_ID=m.Movie_ID and g.Geners_Type="Family"


SELECT * FROM `family_type_movies_details`
```

Showing rows 0 - ... (Query took 0.0014 seconds.)

```
SELECT * FROM `family_type_movies_details`
```

	Title	Year	IMDB_Score
Edit Copy Delete	Alice Through the Looking Glass?	2016	6.4
Edit Copy Delete	Allegiant?	2016	5.8
Edit Copy Delete	Ben-Hur?	2016	6
Edit Copy Delete	Captain America: Civil War?	2016	8.2
Edit Copy Delete	Code of Honor?	2016	4.2
Edit Copy Delete	Ghostbusters?	2016	5.5
Edit Copy Delete	Kung Fu Panda 3?	2016	7.2
Edit Copy Delete	Pride and Prejudice and Zombies?	2016	5.8
Edit Copy Delete	The BFG?	2016	6.8
Edit Copy Delete	The Finest Hours?	2016	6.8

5.6 Functions

1. In the Procedure "Login_Flex_Stream" We have used a Function "get_User_Name(user_id)" which returns the Full name of the user according to the parameter "user_id"

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` FUNCTION
    `get_User_Name`(`user_id` VARCHAR(10)) RETURNS
    varchar(50) CHARSET utf8 COLLATE utf8_bin
  DETERMINISTIC
BEGIN
    declare user_name varchar(50);
  select s.Full_Name into user_name from Subscriber s
    where s.sub_id=user_id;
RETURN user_name;
END$$
DELIMITER ;
```

```
SET @p0='S035'; SELECT `get_User_Name`(@p0) AS `get_User_Name`;
```

Execution results of routine `get_User_Name`

get_User_Name
Frank Moen

2. In the demo of user defined exception we have used a function “check log info(user_id,pass_word)” to verify the log in details of that user.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` FUNCTION
    `check_Log_info`(`User_id` VARCHAR(7), `Pass_word`
    VARCHAR(20)) RETURNS tinyint(1)
  NO SQL
BEGIN
    declare temp bool default false;
    if exists (select s.Sub_id from subscriber s where
    s.sub_id=User_id and          s.Sub_password=Pass_word)
  THEN set temp=true;
    end if;
  RETURN temp;
END$$
DELIMITER ;
```

```
SET @p0='S035'; SET @p1='sssss'; SELECT `check_Log_info`(@p0, @p1) AS `check_Log_info`;
```

Execution results of routine `check_Log_info`

check_Log_info
0

```
SET @p0='S035'; SET @p1='AkVOSN'; SELECT `check_Log_info`(@p0, @p1) AS `check_Log_info`;
```

Execution results of routine `check_Log_info`

check_Log_info
1

5.7 PROCEDURES

1. Give Procedure “Subscriber Details” with Parameter Customers Name, Which will display all information related to that user with appropriate messages.

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
    `Subscriber_Details`(IN `CustName` VARCHAR(20))
BEGIN
    if EXISTS (select s.Sub_ID from subscriber s where
    s.Full_Name=CustName)
  then
    select CustName as "Details of Subscriber :"  ;
        select * from subscriber where FULL_Name=CustName;
  select CustName as "Transection Details of Subscriber :";
        select t.* from transection t inner join subscriber s
        where s.sub_id=t.sub_id and s.Full_Name=CustName;
  select CustName as "Movie Purchased By :";
        select * from Movie where Movie.Movie_ID in
    (select t.Movie_ID from transection t inner join subscriber s
    where s.sub_id=t.sub_id and s.Full_Name=CustName);
  ELSE
    SELECT "User Does Not Exist" as "Error";
    end if;
END$$
DELIMITER ;
```

```
SET @p0='Tirth'; CALL `Subscriber_Details`(@p0);
```

Execution results of routine `Subscriber_Details`

Error
User Does Not Exist

```
SET @p0='Mrs. Arnold Fadel'; CALL `Subscriber_Details`(@p0);
```

Execution results of routine `Subscriber_Details`

Details of Subscriber :
Mrs. Arnold Fadel

Sub_ID	Sub_Password	Phone	Email	Full_Name	Country
S003	KZ22lo2K	1-012-234-1176 x30526	Danika@bart.net	Mrs. Arnold Fadel	Panama

Transection Details of Subscriber :
Mrs. Arnold Fadel

Transection_id	Account_Number	Banks_Name	Sub_ID	Movie_id	Transection_Date
T015	4,410,000,000,000,000	Syndicate Bank	S003	M015	2019-08-24
T063	30,300,000,000,000	Jammu and Kashmir Bank	S003	M043	2019-08-15

Movie Purchased By :
Mrs. Arnold Fadel

Movie_ID	Title	Year	IMDB_Score	Budget	Gross_Earnings	Movie_Purchas_id
M015	Code of Honor?	2016	4.2	0	0	MP005
M043	Kung Fu Panda 3?	2016	7.2	0	0	MP004

2. Give the procedure “Log In” with parameter User id and Password, Which will Welcome the User With appropriate Welcome Message

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
    `Log_IN_Flex_Stream`(IN `user_id` VARCHAR(5), IN `pass_word`
    VARCHAR(20))
BEGIN
    declare temp varchar(100) default "False";
  declare name_cust varchar(50) ;
    if exists (select s.Sub_id from subscriber s where s.sub_id=user_id
    and s.Sub_password=pass_word)
        then set name_cust=get_User_Name(user_id);
     set temp=concat(" Welcome Back ",name_cust," To Online Movie
    Streaming");
    else
        set temp="Invalid User ID or Password";
    end if;
  select temp as "Accesss";
END$$
DELIMITER ;
```

```
SET @p0='S035'; SET @p1='AkVOSN'; CALL `Log_IN_Flex_Stream`(@p0, @p1);
```

Execution results of routine `Log_IN_Flex_Stream`

Accesss
Welcome Back Frank Moen To Online Movie Streaming

```
SET @p0='S002'; SET @p1='sss'; CALL `Log_IN_Flex_Stream`(@p0, @p1);
```

Execution results of routine `Log_IN_Flex_Stream`

Accesss
Invalid User ID or Password

5.8 Trigger

1. After Inserting the movie data in Movie Entity ,The Entry in Genre table will automatically executed with given Movie id with genre type =”Entertainment”

```
CREATE TRIGGER `after_movie_insert` AFTER INSERT ON `movie`
 FOR EACH ROW BEGIN
     DECLARE new_movie_id varchar(10);

    select Max(movie.Movie_ID) into new_movie_id from movie ;
   if not EXISTS (SELECT g.Movie_ID from genres g where
                                                g.Movie_ID=new_movie_id)
     THEN insert into genres(genres.Geners_Type,genres.Movie_ID)
                                VALUES("Entertenment",new_movie_id);
     end if;
END
```

//Befor_Trigger

```
SELECT * FROM `genres` ORDER BY `Movie_ID` DESC
```

	Geners_Type	Movie_ID
Edit Copy Delete	Entertenment	M100
Edit Copy Delete	War	M098
Edit Copy Delete	Comedy	M097
Edit Copy Delete	Action	M096
Edit Copy Delete	Drama	M095
Edit Copy Delete	Drama	M094
Edit Copy Delete	Biography	M093
Edit Copy Delete	Crime	M092
Edit Copy Delete	Mystery	M091
Edit Copy Delete	Sport	M090

```
INSERT INTO `movie`(`Movie_ID`, `Title`, `Year`, `Movie_Purchas_id`) VALUES
    ("M101","Toy Story 4",2019,"MP001");
```

//After Trigger

```
SELECT * FROM `genres` ORDER BY `Movie_ID` DESC
```

	Geners_Type	Movie_ID 1
Edit Copy Delete	Entertenment	M101
Edit Copy Delete	Entertenment	M100
Edit Copy Delete	War	M098
Edit Copy Delete	Comedy	M097
Edit Copy Delete	Action	M096
Edit Copy Delete	Drama	M095
Edit Copy Delete	Drama	M094
Edit Copy Delete	Biography	M093
Edit Copy Delete	Crime	M092

5.9. Cursor

1. Generate the list of email of that user whose validity to watch their subscripted movie will be over in 5 days (Reminder)

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
      `Create_Email_List_Reminder`()
BEGIN
   DECLARE finished INTEGER DEFAULT 0;
   DECLARE emailAddress varchar(100) DEFAULT "";
      DECLARE emailList varchar(40000) DEFAULT "";
   -- declare cursor for Subscriber email
   DEClARE curEmail
      CURSOR FOR

            SELECT s.Email from Transection t inner join movie m inner
                              join charge c INNER join subscriber s
                     where t.Movie_id=m.Movie_ID and

            m.Movie_Purchas_id=c.Movie_Purchas_ID and t.Sub_ID
      =s.Sub_ID                          and
      DATEDIFF(CURDATE(),t.Transection_Date) < c.Validity and
            c.Validity-DATEDIFF(CURDATE(),t.Transection_Date)<5;
   -- declare NOT FOUND handler
   DECLARE CONTINUE HANDLER
      FOR NOT FOUND SET finished = 1;

   OPEN curEmail;

   getEmail: LOOP
      FETCH curEmail INTO emailAddress;
      IF finished = 1 THEN
         LEAVE getEmail;
      END IF;
      -- build email list
      SET emailList = CONCAT(emailAddress,";",emailList);
   END LOOP getEmail;
   CLOSE curEmail;
      select emailList as "Send Email To Given Users";
   select "Your Validity will over soon, Dont Miss The High Quality
      Streaming" as "Reminder";
END$$
DELIMITER ;
```

```
CALL `Create_Email_List_Reminder`();
```

Execution results of routine `Create_Email_List_Reminder`

Send Email To Given Users

Adrien.Kemmer@kelton.com;

Reminder

Your Validity will over soon, Dont Miss The High Quality Streaming

2. Generate list of existing user email to notify them that a new movie is added in our Streaming Service .

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `new_movie_added`()
BEGIN
  DECLARE finished INTEGER DEFAULT 0;
  DECLARE emailAddress varchar(100) DEFAULT "";
    DECLARE emailList varchar(40000) DEFAULT "";
  DEClARE curEmail
    CURSOR FOR
        SELECT s.Email from subscriber s;
  DECLARE CONTINUE HANDLER
    FOR NOT FOUND SET finished = 1;
  OPEN curEmail;
  getEmail: LOOP
    FETCH curEmail INTO emailAddress;
    IF finished = 1 THEN
      LEAVE getEmail;
    END IF;
    SET emailList = CONCAT(emailAddress,"; \n",emailList);
  END LOOP getEmail;
  CLOSE curEmail;
    select emailList as "Send Email To Given Users";
  select "New High Quality Movie is here, Check Out Now!!!!" as
    "Attention!!";
END$$
DELIMITER ;
```

5.10 Event

1. At every day predefined event named “Expired” will be executed which will check the validity of movie based on the day difference of transaction date and validity of movie, according to that entry from transaction(Dummy) table will be removed which will indicate that the user can not watch that movie anymore whose validity is over.

```
CREATE DEFINER=`root`@`localhost` EVENT `Expired` ON SCHEDULE
EVERY 1 DAY STARTS '2019-09-26 14:05:00' ENDS '2019-11-30 12:00:00'
ON COMPLETION PRESERVE ENABLE DO DELETE t
from transection t inner join
movie m inner join charge c
where t.Movie_id=m.Movie_ID and
 m.Movie_Purchas_id=c.Movie_Purchas_ID and
DATEDIFF(CURDATE(),t.Transection_Date) > c.Validity
```

6. FUTURE ENHANCEMENTS OF THE SYSTEM

- Full Front-end Design.
- Prime Membership with Special offers and Reduction in Movie purchase prize.
- Emails are automatically sent to user's registered email id when there are new movies available in System
- Emails are automatically sent to user's registered email id when user's validity to stream movie will over soon
- Notifications and email about offers and new releases are sent automatically.
- Movies are sorted automatically according to IMDB ratings.
- I will make database more consistent.

7. BIBLIOGRAPHY

For the successful implementation of this project I referred to many websites and books. The schema was designed by taking ideas from movie streaming applications like Google Play Movies, Amazon Prime Video, and Hotstar etc. I created the ER Diagram on "Creately.com" website and Schema Diagram on "dbdiagram.io". I also used online data generator like "Mockaroo.com" and many more that's why my system may not be that much consistent .Mostly I referred the online material for syntax of procedures, triggers, Exception and cursors.

Reference book:
Data Base System Concepts
-Henry F. Korth & A. Silberschatz 2nd Ed. McGraw-Hill 1991

Reference Websites:

- **https://www.w3schools.com/**
- **https://stackoverflow.com/**
- **http://www.mysqltutorial.org/**