Tablas

User (id, username, name, lastname, email, phone, password)

- id es Int es la clave primaria, no puede ser nula y es autoincremental.
- username es Varchar, contiene el nombre del Usuario, no puede ser nulo y debe ser único.
- name es Varchar, contiene el nombre verdadero del Usuario, no puede ser nulo.
- lastname es Varchar, contiene el apellido verdadero del Usuario, no puede ser nulo.
- email es Varchar, contiene el correo del Usuario, no puede ser nulo y debe ser único.
- phone es Varchar, contiene el teléfono del Usuario, no puede ser nulo y es único.
- password es Varchar, contiene la contraseña del Usuario, no puede ser nulo.

Admin (id,username,name, lastname, email, phone, password)

- id es la clave primaria, no puede ser nula y es autoincremental.
- username es Varchar, contiene el nombre del Administrador, no puede ser nulo y es único.
- name es Varchar, contiene el nombre verdadero del Administrador, no puede ser nulo.
- lastname es Varchar, contiene el apellido verdadero del Administrador, no puede ser nulo.
- email es Varchar, contiene el correo del Administrador, no puede ser nulo y es único.
- phone es Varchar, contiene el teléfono del Administrador, no puede ser nulo y es único.
- password es Varchar, contiene la contraseña del Administrador, no puede ser nulo.

Card (id, cardNumber, cvc, expirationDate, userId)

- id Int es la clave primaria, no puede ser nula y es autoincremental.
- cardNumber es Varchar, contiene el número de cuenta, no puede ser nulo.
- fullname es Varchar, contiene el nombre del nombre del dueño de la tarjeta, no puede ser nulo.
- cvc es Varchar, contiene el código de verificación del usuario, no puede ser nulo.
- expirationDate es Date, contiene el tiempo de expiración de la tarjeta, no puede ser nulo.
- userId es una llave foranea User.
    - no puede ser nulo
    - Actualizar: Cascada
    - Eliminar: Restringir

Movie (id, categoryId, category, photo, title, premiereDate, classification, duration, buyPrice,rentPrice, directorId,actor)

- id es Int es la clave primaria, no puede ser nula y es autoincremental
- categoryId es una clave foranea de Category.
    - no puede ser nulo
    - Actualizar: Cascada
    - Eliminar: Restringir
- category es una llave foránea y relaciona el id de la categoría con el id de la película, no puede ser nulo.
- photo es Varchar, contiene la liga de la foto de la película, no puede ser nulo
- title es Varchar, contiene el título de la película, no puede ser nulo
- premiereDate es Date, contiene la fecha de estreno de la película, no puede ser nulo.
- classification es Varchar, contiene la clasificación de la película, no puede ser nulo.
- duration es varchar, contiene la duración de la película, no puede ser nulo
- buyPrice es Double, contiene el precio de compra de la película, no puede ser nulo
- rentPrice es Double, contiene el precio de renta de la película, no puede ser nulo
- directorId es una clave foránea de Director
    - no puede ser nulo
    - Actualizar: Cascada
    - Eliminar: Restringir
- El director es una llave foránea y relaciona el id del director con la película, no puede ser nulo.
- synopsis es longtext, contiene la sinopsis de la película, no puede ser nulo

Category (id, name)

- id se almacena el id de cada fila, es de tipo int, es auto incrementable y no es nulo.
- name es de tipo Varchar y es único

Director (id, fullname, photo, birthdate, birthplace, biography)

- id almacena el id de cada fila, es de tipo int, es auto incrementable y no es nulo.
- fullname es tipo varchar, y no es nulo.
- photo es tipo Varchar y no es nulo.
- birthdate es tipo Date y no puede ser nulo.
- birthplace es de tipo Varchar y no puede ser nulo.
- biography es de tipo Varchar y no puede ser nulo.

Actor (id, fullname, photo, birthdate, birthplace, biography)

- id almacena el id de cada fila, es de tipo int, es auto incrementable y no es nulo.
- fullname es tipo varchar, y no es nulo.
- photo es tipo Varchar y no es nulo.
- birthdate es tipo Date y no puede ser nulo.
- birthplace es de tipo Varchar y no puede ser nulo.
- biography es de tipo Varchar y no puede ser nulo.

Ticket (id, type, movieId, userId, createdAt, expirationDate)

- id se almacena el id de cada fila, es de tipo int, es auto incrementable y no es nulo.
- type es de tipo VarChar contiene el tipo de venta, y no es nulo.
- movieId es clave foránea de Movie.
    - no puede ser nulo
    - Actualizar: Cascada
    - Eliminar: Hacer nulo
- userId es clave foránea de User.
    - no puede ser nulo
    - Actualizar: Cascada
    - Eliminar: Restringir
- createdAt es tipo Date con default (fecha actual)
- expirationDate es de tipo Date y no puede ser nulo.

Review(id, grade, movieId, userId)

- id se almacena el id de cada fila, es de tipo int, es auto incrementable y no es nulo.
- grade es de tipo entero contiene la calificación, no es nulo.
- movieId es clave foránea de Movie.
    - no puede ser nulo
    - Actualizar: Cascada
    - Eliminar: Restringir
- userId es clave foránea de User.
    - no puede ser nulo
    - Actualizar: Cascada
    - Eliminar: Restringir

ActorToMovie (a, b)

- a es clave foránea de Actor.
    - no puede ser nulo
    - Actualizar: Cascada
    - Eliminar: Restringir
- b es clave foránea de Movie.
    - no puede ser nulo
    - Actualizar: Cascada
    - Eliminar: Restringir

AdminLog(id, createdAt, subject, change, subjectId)

- id se almacena el id de cada fila, es de tipo int, es auto incrementable y no es nulo.
- createdAt representa la fecha de creación del elemento es tipo Date con default (fecha actual)
- subject almacena lo que fue cambiado es tipo string y no puede ser nulo.

- change almacena más información acerca del cambio, es tipo string y no puede ser nulo.
- subjectId almacena el id de lo que fue cambiado.

Earning (id, quantity, createdAt, ticketId)

- id se almacena el id de cada fila, es de tipo int, es auto incrementable y no es nulo.
- quantity es tipo Int y no puede ser nulo
- createdAt es tipo Date con default (fecha actual)
- ticketId es clave foránea de Ticket.
    - no puede ser nulo
    - Actualizar: Cascada
    - Eliminar: Hacer nulo

Bookmark (id, movieId, userId)

- id se almacena el id de cada fila, es de tipo int, es auto incrementable y no es nulo.
- movieId es clave foránea de Movie.
    - no puede ser nulo
    - Actualizar: Cascada
    - Eliminar: Hacer nulo
- userId es clave foránea de User.
    - no puede ser nulo
    - Actualizar: Cascada
    - Eliminar: Restringir

Triggers

–Este trigger se utiliza para que se realice un registro cuando un administrador agregue a un actor.

```
CREATE TRIGGER adminLogAddedActor
BEFORE INSERT
ON _ActorToMovie FOR EACH ROW
BEGIN
INSERT INTO AdminLog (subject, `change`,  subjectId) VALUES ('Movie', 'Actor added to Movie.', new.b);
INSERT INTO AdminLog (subject, `change`, subjectId) VALUES ('Actor', 'Actor added to Movie.', new.a);
END;
```

–Este trigger se utiliza para que se realice un registro cuando un administrador elimine a un actor.

```
CREATE TRIGGER adminLogRemovedActor
BEFORE DELETE
ON _ActorToMovie FOR EACH ROW
BEGIN
INSERT INTO AdminLog (subject, `change`,  subjectId) VALUES ('Movie', 'Actor removed from Movie.', old.b);
```

```
INSERT INTO AdminLog (subject, `change`, subjectId) VALUES ('Actor', 'Actor removed from Movie.', old.a);
END;
```

–Este trigger se utiliza para que se realice un registro cuando un administrador agregue a una nueva película.

```
CREATE TRIGGER adminLogCreatedMovie
BEFORE INSERT
ON Movie FOR EACH ROW
BEGIN
INSERT INTO AdminLog (subject, `change`,  subjectId) VALUES ('Movie', 'Movie created.', new.id);
INSERT INTO AdminLog (subject, `change`, subjectId) VALUES ('Director', 'Director added to movie.', new.directorId);
END;
```

–Este trigger se utiliza para que se realice un registro cuando un administrador actualice una nueva película.

```
CREATE TRIGGER adminLogUpdatedMovie
BEFORE UPDATE
ON Movie FOR EACH ROW
BEGIN
INSERT INTO AdminLog (subject, `change`,  subjectId) VALUES ('Movie', 'Movie updated.', new.id);
IF old.directorId != new.directorId THEN
   INSERT INTO AdminLog (subject, `change`,  subjectId) VALUES ('Director', 'Director removed from movie.', old.directorId);
   INSERT INTO AdminLog (subject, `change`,  subjectId) VALUES ('Director', 'Director added from movie.', new.directorId);
END IF;
END;
```

–Este trigger se utiliza para que se realice un registro cuando un administrador crea un nuevo actor.

```
CREATE TRIGGER adminLogCreatedActor
BEFORE INSERT
ON Actor FOR EACH ROW
BEGIN
INSERT INTO AdminLog (subject, `change`,  subjectId) VALUES ('Actor', 'Actor created.', new.id);
END;
```

–Este trigger se utiliza para que se realice un registro cuando un administrador actualice un nuevo actor.

```
CREATE TRIGGER adminLogUpdatedActor
BEFORE UPDATE
```

```
ON Actor FOR EACH ROW
BEGIN
INSERT INTO AdminLog (subject, `change`,  subjectId) VALUES ('Actor', 'Actor updated.',
new.id);
END;
```

–Este trigger se utiliza para que se realice un registro cuando un administrador crea un nuevo director.

```
CREATE TRIGGER adminLogCreatedDirector
BEFORE INSERT
ON Director FOR EACH ROW
BEGIN
INSERT INTO AdminLog (subject, `change`,  subjectId) VALUES ('Director', 'Director
created.', new.id);
END;
```

–Este trigger se utiliza para que se realice un registro cuando un administrador actualice un
Director.

```
CREATE TRIGGER adminLogUpdatedDirector
BEFORE UPDATE
ON Director FOR EACH ROW
BEGIN
INSERT INTO AdminLog (subject, `change`,  subjectId) VALUES ('Director', 'Director
updated.', new.id);
END;
```

–Este trigger se utiliza para registrar las nuevas ganancias.

```
CREATE TRIGGER setNewEarning
AFTER INSERT
ON Ticket FOR EACH ROW
BEGIN
IF new.type LIKE 'buy' THEN
   SET @Earned = (SELECT buyPrice FROM Movie WHERE new.movieId = Movie.id);
   INSERT INTO Earnings (quantity, ticketId) VALUES (@Earned, new.id);
END IF;
IF new.type LIKE 'rent' THEN
   SET @Earned = (SELECT rentPrice FROM Movie WHERE new.movieId = Movie.id);
   INSERT INTO Earnings (quantity, ticketId) VALUES (@Earned, new.id);
END IF;
END;
```

Vistas.-

-- Vista para mostrar las mejores películas

```
CREATE VIEW BestMovies AS
SELECT M.*, AVG(grade) FROM Review
JOIN Movie M on M.id = Review.movieId
GROUP BY movieId
HAVING AVG(grade) > 4.5;


-- Vista para mostrar las películas de la categoría acción
CREATE VIEW ActionMovies AS
SELECT * FROM Movie
WHERE categoryId = 1;


-- Vista para mostrar las películas de la categoría ciencia ficción
CREATE VIEW ScienceFictionMovies AS
SELECT * FROM Movie
WHERE categoryId = 2;



-- Vista para mostrar las películas de la categoría comedia
CREATE VIEW ComedyMovies AS
SELECT * FROM Movie
WHERE categoryId = 3;



-- Vista para mostrar las películas de la categoría drama
CREATE VIEW DramaMovies AS
SELECT * FROM Movie
WHERE categoryId = 4;


-- Vista para mostrar  las películas de la categoría musical
CREATE VIEW MusicalMovies AS
SELECT * FROM Movie
WHERE categoryId = 5;


-- Vista para mostrar las películas de la categoría romance
CREATE VIEW RomanceMovies AS
SELECT * FROM Movie
WHERE categoryId = 6;


-- Vista para mostrar las películas de la categoría superhéroes
CREATE VIEW SupeheroMovies AS
```

```
SELECT * FROM Movie
WHERE categoryId = 7;


-- Vista para mostrar las películas de la categoría terror
CREATE VIEW HorrorMovies AS
SELECT * FROM Movie
WHERE categoryId = 8;


-- Vista para mostrar todas las categorías
CREATE VIEW AllCategories AS
SELECT * FROM Category;



-- Vista para traer todas las categorías
CREATE VIEW AllUsers AS
SELECT * FROM User;


-- Vista para traer todos los usuarios
CREATE VIEW TotalUsers AS
SELECT COUNT(*) FROM User;

-- Vista para traer todas las películas
CREATE VIEW TotalMovies AS
SELECT COUNT(*) FROM Movie;
```

Procedimientos.-

```
-- Procedimiento para traer las películas del usuario
CREATE PROCEDURE getMyMovies(IN USER_ID int)
BEGIN
SELECT * FROM Ticket
JOIN Movie M on M.id = Ticket.movieId
WHERE userId = USER_ID AND NOT(ISNULL(movieId)) AND type NOT LIKE '%saved%';
END;

DROP PROCEDURE getMyMovies;

CALL getMyMovies(1);

-- Procedimiento para traer las películas guardadas del usuario
CREATE PROCEDURE getMyBookmarkedMovies(IN USER_ID int)
BEGIN
SELECT * FROM Ticket
JOIN Movie M on M.id = Ticket.movieId
WHERE userId = USER_ID AND NOT(ISNULL(movieId)) AND type = 'saved';
```

```
END;

DROP PROCEDURE getMyBookmarkedMovies;

CALL getMyBookmarkedMovies(1);

-- Procedimiento para traer las películas con los tickets del usuario
CREATE PROCEDURE getMovieWithUserTickets(IN USER_ID int)
BEGIN
SELECT * FROM Movie
JOIN Ticket T on Movie.id = T.movieId
WHERE T.userId = USER_ID;
END;

DROP PROCEDURE getMovieWithUserTickets;

CALL getMovieWithUserTickets(1);


-- Procedimiento para traer el promedio de calificaciones de una película
CREATE PROCEDURE getMovieAverageRating(IN MOVIE_ID int)
BEGIN
SELECT M.id, AVG(grade) FROM Review
JOIN Movie M on M.id = Review.movieId
WHERE movieId = MOVIE_ID
GROUP BY movieId;
END;

DROP PROCEDURE getMovieAverageRating;

CALL getMovieAverageRating(1);


-- Procedimiento para traer las tarjetas de un usuario
CREATE PROCEDURE getUserCards(IN USER_ID int)
BEGIN
SELECT * FROM Card
JOIN User U on U.id = Card.userId
WHERE userId = USER_ID;
END;

DROP PROCEDURE getUserCards;

CALL getUserCards(1);


-- Procedimiento para crear un usuario
```

```
CREATE PROCEDURE createUser(IN $username varchar(255), IN $name varchar(255), IN $lastname varchar(255), IN $email varchar(255), IN $phone varchar(10), IN $password varchar(255))
BEGIN
INSERT INTO User (username, name, lastname, email, phone, password)
VALUES ($username, $name, $lastname, $email, $phone, $password);
END;
```

-- Procedimiento para crear una tarjeta

```
CREATE PROCEDURE createCard(IN $cardNumber varchar(255), IN $fullname varchar(255), IN $cvc varchar(3), IN $expirationTime varchar(5), IN $userId int)
BEGIN
INSERT INTO Card (cardNumber, fullname, cvc, expirationTime, userId)
VALUES ($cardNumber, $fullname, $cvc, $expirationTime, $userId);
END;
```

-- Procedimiento para crear un admin

```
CREATE PROCEDURE createAdmin(IN $username varchar(255), IN $name varchar(255), IN $lastname varchar(255), IN $email varchar(255), IN $phone varchar(10), IN $password varchar(255))
BEGIN
INSERT INTO Admin (username, name, lastname, email, phone, password)
VALUES ($username, $name, $lastname, $email, $phone, $password);
END;
```

-- Procedimiento para crear un actor

```
CREATE PROCEDURE createActor(IN $fullname varchar(255), IN $photo varchar(255), IN $birthdate varchar(255), IN $birthplace varchar(255), IN $biography varchar(255))
BEGIN
INSERT INTO Actor (fullname, photo, birthdate, birthplace, biography)
VALUES ($fullname, $photo, $birthdate, $birthplace, $biography);
END;
```

-- Procedimiento para crear un director

```
CREATE PROCEDURE createDirector(IN $fullname varchar(255), IN $photo varchar(255), IN $birthdate varchar(255), IN $birthplace varchar(255), IN $biography varchar(255))
BEGIN
INSERT INTO Director (fullname, photo, birthdate, birthplace, biography)
VALUES ($fullname, $photo, $birthdate, $birthplace, $biography);
END;
```

-- Procedimiento para crear una película

```
CREATE PROCEDURE createMovie(IN $categoryId varchar(255), IN $photo varchar(255), IN $title varchar(255), IN $premiereDate int, IN $classification varchar(255), IN $duration
```

```
varchar(255), IN $buyPrice double, IN $rentPrice double, IN $directorId int, IN $synopsis
longtext)
BEGIN
INSERT INTO Movie (categoryId, photo, title, premiereDate, classification, duration,
buyPrice, rentPrice, directorId, synopsis)
VALUES ($categoryId, $photo, $title, $premiereDate, $classification, $duration, $buyPrice,
$rentPrice, $directorId, $synopsis);
END;

-- Procedimiento para crear una reseña
CREATE PROCEDURE createReview(IN $grade int, IN $movieId int, IN $userId int)
BEGIN
INSERT INTO Review (grade, movieId, userId)
VALUES ($grade, $movieId, $userId);
END;

-- Procedimiento para crear un ticket
CREATE PROCEDURE createTicket(IN $type varchar(255), IN $movieId int, IN $userId int,
IN $createdAt date, IN $expirationDate date)
BEGIN
INSERT INTO Ticket (type, movieId, userId, createdAt, expirationDate)
VALUES ($type, $movieId, $userId, $createdAt, $expirationDate);
END;

-- Procedimiento para actualizar un usuario
CREATE PROCEDURE updateUser(IN USER_ID int, IN $username varchar(255), IN $name
varchar(255), IN $lastname varchar(255), IN $email varchar(255), IN $phone varchar(10), IN
$password varchar(255))
BEGIN
UPDATE User SET username = $username, name = $name, lastname = $lastname, email =
$email, phone = $phone, password = $password
WHERE id = USER_ID;
END;

-- Procedimiento para actualizar un admin
CREATE PROCEDURE updateAdmin(IN ADMIN_ID int, IN $username varchar(255), IN
$name varchar(255), IN $lastname varchar(255), IN $email varchar(255), IN $phone
varchar(10), IN $password varchar(255))
BEGIN
UPDATE Admin SET username = $username, name = $name, lastname = $lastname, email
= $email, phone = $phone, password = $password
WHERE id = ADMIN_ID;
END;



-- Procedimiento para actualizar un actor
```

```
CREATE PROCEDURE updateActor(IN ACTOR_ID int, IN $fullname varchar(255), IN $photo varchar(255), IN $birthdate varchar(255), IN $birthplace varchar(255), IN $biography varchar(255))
BEGIN
UPDATE Actor SET fullname = $fullname, photo = $photo, birthdate = $birthdate, birthplace = $birthplace, biography = $biography
WHERE id = ACTOR_ID;
END;
```

-- Procedimiento para actualizar un director

```
CREATE PROCEDURE updateDirector(IN DIRECTOR_ID int, IN $fullname varchar(255), IN $photo varchar(255), IN $birthdate varchar(255), IN $birthplace varchar(255), IN $biography varchar(255))
BEGIN
UPDATE Director SET fullname = $fullname, photo = $photo, birthdate = $birthdate, birthplace = $birthplace, biography = $biography
WHERE id = DIRECTOR_ID;
END;
```

-- Procedimiento para actualizar una película

```
CREATE PROCEDURE updateMovie(IN MOVIE_ID int, IN $categoryId varchar(255), IN $photo varchar(255), IN $title varchar(255), IN $premiereDate int, IN $classification varchar(255), IN $duration varchar(255), IN $buyPrice double, IN $rentPrice double, IN $directorId int, IN $synopsis longtext)
BEGIN
UPDATE Movie SET categoryId = $categoryId, photo = $photo, title = $title, premiereDate = $premiereDate, classification = $classification, duration = $duration, buyPrice = $buyPrice, rentPrice = $rentPrice, directorId = $directorId, synopsis = $synopsis
WHERE id = MOVIE_ID;
END;
```