CS 353 Project

“DMGTV”

Project Design Report

Gökberk Beydemir - 21902638

Melis Atun - 21901865

Mert Barkın Er - 21901645

Doruk Kantarcıoğlu - 21902319

TA: Zülal Bingöl

https://dorukkantarcioglu.github.io/cs-353-website/

Table of Contents

E/R Diagram Revisions

According to the feedback taken from our TA, Zülal Bingöl, the ER diagram of our project has been revised.

Below are the points where the revisions are made on the ER diagram:

- Unnecessary total participation cases are deleted from the ER diagram.
- Redundant registration_date attribute is deleted from the table ‘customer’.
- Redundant reg_customer_date attribute is deleted from the relation ‘register_customer’.
- Straight lines in the table ‘credit_card’ are converted to dashed lines due to being a weak entity.
- ‘rating’ attribute of the table ‘movie’ is renamed as ‘movie_rating’ to prevent any confusion related to duplicate attribute names.
- Table name ‘user’ is renamed as ‘member’ as SQL does not allow the name ‘user’ to be used as a table name.
- Attribute ‘like_count’ added to table ‘movie’ as suggested.

Revised E/R Diagram

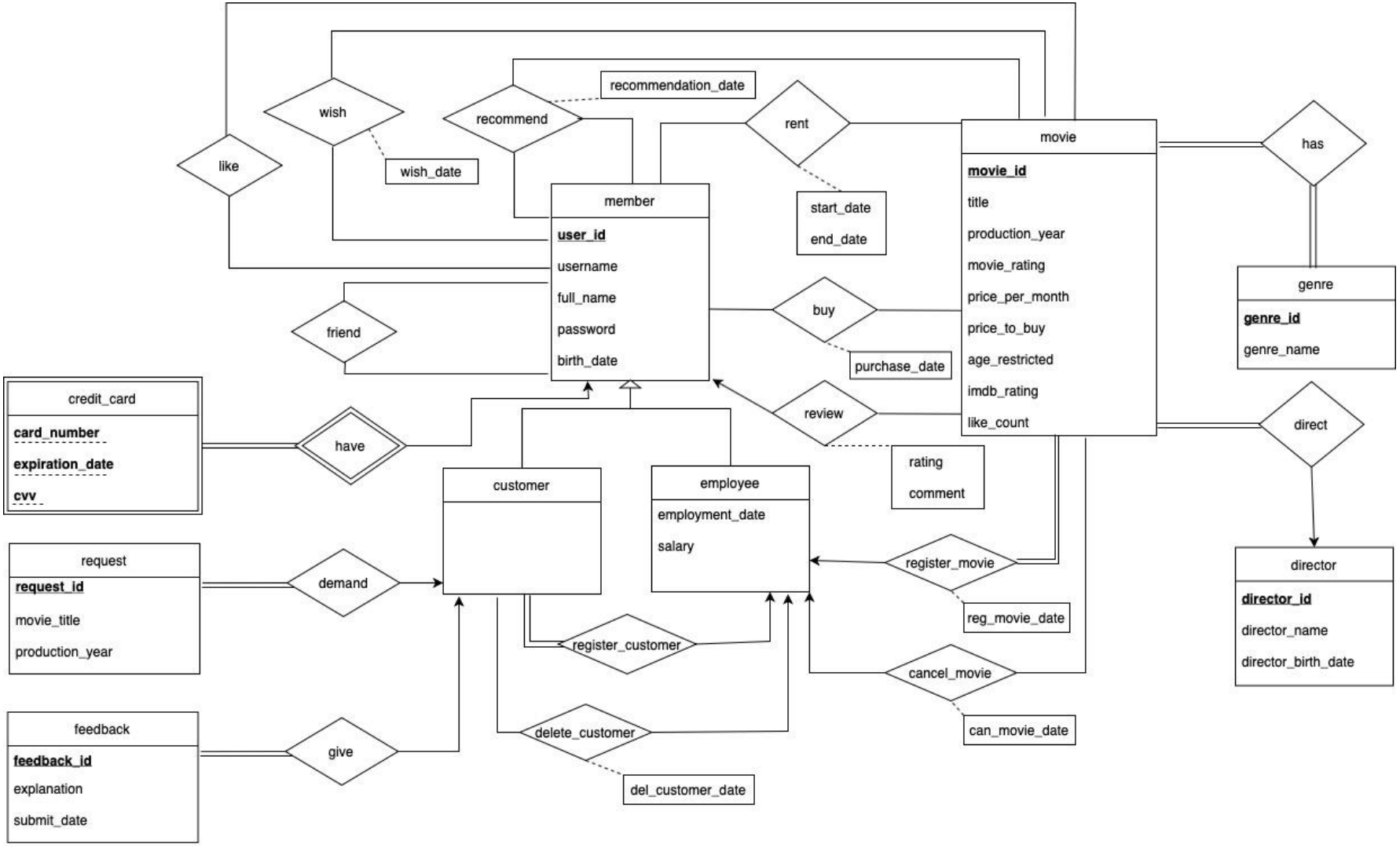


Table Schemas

Member

a. Relational Model

member(<u>user_id</u>, username, full_name, password, birth_date)

b. Functional Dependencies

user_id -> username, full_name, password, birth_date

username -> user_id, full_name, password, birth_date

c. Candidate Keys

{user_id}

{username}

d. Normal Form

This table is in BCNF (and 3NF) form, since both of the functional dependencies have superkeys on the left-hand side (user_id and username).

e. Table Definition

```
create table member(
    user_id int unique not null,
    username varchar(20) unique not null,
    full_name varchar(50) not null,
    password varchar(16) not null,
    birth_date date,
    primary key (user_id));
```

Customer

a. Relational Model

customer(<u>user_id</u>)

b. Functional Dependencies

There are only trivial functional dependencies in this table.

c. Candidate Keys

{user_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there are only trivial functional dependencies in this table.

e. Table Definition

```
create table customer(
    user_id int unique not null,
    foreign key user_id references member(user_id) on update cascade on delete restrict,
    primary key (user_id));
```

Employee

a. Relational Model

employee(<u>user_id</u>, employment_date, salary)

b. Functional Dependencies

user_id -> employment_date, salary

c. Candidate Keys

{user_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table employee(
    user_id int unique not null,
    employment_date date,
    salary int not null,
    foreign key user_id references member(user_id) on update cascade on delete restrict,
    primary key (user_id));
```

Genre

a. Relational Model

genre(genre_id, genre_name)

b. Functional Dependencies

genre_id -> genre_name

c. Candidate Keys

{genre _id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table genre(
    genre_id int unique not null,
    genre_name varchar(64) unique not null,
    primary key (genre_id));
```

Movie

a. Relational Model

movie(<u>movie_id</u>, title, production_year, rating, price_per_month, price_to_buy, age_restricted, imdb_rating, like_count)

b. Functional Dependencies

movie_id -> title, production_year, rating, price_per_month, price_to_buy,age_restricted, imdb_rating, like_count

c. Candidate Keys

{movie_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table movie(
  movie_id int unique not null,
  title varchar(64) not null,
  production_year int,
  rating numeric(2,1),
  price_per_month int not null,
  price_to_buy int,
  age_restricted boolean,
  imdb_rating numeric(2,1),
  like_count int,
  primary key (movie_id));
```

Credit Card

a. Relational Model

credit_card(<u>card_number, expiration_date, cvv</u>)

b. Functional Dependencies

There are only trivial functional dependencies in this table.

c. Candidate Keys

{card_number, expiration_date, cvv}

d. Normal Form

This table is in BCNF (and 3NF) form, since there are only trivial functional dependencies in this table.

e. Table Definition

```
create table credit_card(
    card_number numeric(16,0) unique not null,
    expiration_date date not null,
    cvv numeric(3,0) not null,
    primary key (card_number, expiration_date, cvv));
```

Director

a. Relational Model

director(<u>director_id</u>, director_name, director_birth_date)

b. Functional Dependencies

director_id -> director_name, director_birth_date

c. Candidate Keys

{director_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table director(
    director_id int unique not null,
    director_name varchar(64) not null,
    director_birth_date date not null,
    primary key (director_id));
```

Request

a. Relational Model

request(<u>request_id</u>, movie_title, production_year)

b. Functional Dependencies

request_id -> movie_title, production_year

c. Candidate Keys

{request_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table request(
  request_id int unique not null,
  movie_title varchar(64) not null,
  production_year numeric(4,0) not null,
  primary key (request_id));
```

Feedback

a. Relational Model

feedback(<u>feedback_id</u>, explanation, submit_date)

b. Functional Dependencies

feedback_id -> explanation, submit_date

c. Candidate Keys

{feedback_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table feedback(
    feedback_id int unique not null,
    explanation text not null,
    submit_date date not null,
    primary key (feedback_id));
```

Like Relation

a. Relational Model

like(user_id,movie_id)

b. Functional Dependencies

There are only trivial functional dependencies in this table.

c. Candidate Keys

{user_id, movie_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there are only trivial functional dependencies in this table.

e. Table Definition

```
create table like(
    user_id int not null,
    movie_id int not null,
    foreign key user_id references member(user_id) on update cascade on delete restrict,
    foreign key movie_id references movie(movie_id) on update cascade on delete restrict,
    primary key (user_id, movie_id));
```

Wish Relation

a. Relational Model

wish(<u>user_id,movie_id</u>, wish_date)

b. Functional Dependencies

user_id, movie_id -> wish_date

c. Candidate Keys

{user_id, movie_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table wish(
  user_id int not null,
  movie_id int not null,
  wish_date date not null,
  foreign key user_id references member(user_id) on update cascade on delete restrict,
  foreign key movie_id references movie(movie_id) on update cascade on delete restrict,
  primary key (user_id, movie_id));
```

Friend Relation

a. Relational Model

friend(first_user_id,second_user_id)

b. Functional Dependencies

There are only trivial functional dependencies in this table.

c. Candidate Keys

{first_user_id,second_user_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there are only trivial functional dependencies in this table.

e. Table Definition

```
create table friend(
  first_user_id int not null,
  second_user_id int not null,
  foreign key first_user_id references member(user_id) on update cascade on delete restrict,
  foreign key second_user_id references member(user_id) on update cascade on delete restrict,
  primary key (first_user_id, second_user_id));
```

User - Credit Card Relation

a. Relational Model

user_credit_card(<u>card_number, expiration_date, cvv</u>, user_id)

b. Functional Dependencies

card_number, expiration_date, cvv -> user_id

c. Candidate Keys

{card_number, expiration_date, cvv}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table user_credit_card(
  card_number numeric(16,0) unique not null,
  expiration_date date not null,
  cvv numeric(3,0) not null,
  user_id int not null,
  foreign key card_number references credit_card(card_number) on update cascade on delete restrict,
  foreign key expiration_date references credit_card(expiration_date) on update cascade on delete restrict,
  foreign key cvv references credit_card(cvv) on update cascade on delete restrict,
  foreign key user_id references member(user_id) on update cascade on delete restrict,
  primary key (card_number,expiration_date, cvv));
```

Demand Relation

a. Relational Model

demand(<u>request_id</u>, user_id)

b. Functional Dependencies

request_id -> user_id

c. Candidate Keys

{request_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table demand(
  request_id int unique not null,
  user_id int not null,
  foreign key request_id references request(request_id) on update cascade on delete restrict,
  foreign key user_id references member(user_id) on update cascade on delete restrict,
  primary key (request_id));
```

Give Relation

a. Relational Model

give(feedback_id, user_id)

b. Functional Dependencies

feedback_id -> user_id

c. Candidate Keys

{feedback_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table give(
  feedback_id int unique not null,
  user_id int not null,
  foreign key feedback_id references feedback(feedback_id) on update cascade on delete restrict,
  foreign key user_id references member(user_id) on update cascade on delete restrict,
  primary key (feedback_id));
```

Recommend Relation

a. Relational Model

recommendation(<u>user_id,movie_id</u>, recommendation_date)

b. Functional Dependencies

user_id, movie_id-> recommendation_date

c. Candidate Keys

{ user_id, movie_id }

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table recommendation(
   user_id int not null,
   movie_id int not null,
   recommendation_date date not null,
   foreign key user_id references member(user_id) on update cascade on delete restrict,
   foreign key movie_id references movie(movie_id) on update cascade on delete
restrict,
   primary key (user_id,movie_id));
```

Register Customer Relation

a. Relational Model

register_customer(<u>customer_id,</u> employee_id)

b. Functional Dependencies

customer_id -> employee_id

c. Candidate Keys

{customer_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table register_customer(
  customer_id int unique not null,
  employee_id int not null,
  foreign key customer_id references member(user_id) on update cascade on delete restrict,
  foreign key employee_id references member(user_id) on update cascade on delete restrict,
  primary key (customer_id));
```

Delete Customer Relation

a. Relational Model

delete_customer(<u>customer_id,</u> employee_id, del_customer_date)

b. Functional Dependencies

customer_id -> employee_id, del_customer_date

c. Candidate Keys

{customer_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table delete_customer(
  customer_id int unique not null,
  employee_id int not null,
  del_customer_date date not null,
  foreign key customer_id references member(user_id) on update cascade on delete restrict,
  foreign key employee_id references member(user_id) on update cascade on delete restrict,
  primary key (customer_id));
```

Rent Relation

a. Relational Model

rent(<u>user_id, movie_id, start_date</u>, end_date)

b. Functional Dependencies

user_id, movie_id, start_date -> end_date

c. Candidate Keys

{ user_id, movie_id, start_date}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table rent(
  user_id int not null,
  movie_id int not null,
  start_date date not null,
  end_date date not null,
  foreign key user_id references member(user_id) on update cascade on delete restrict,
  foreign key movie_id references movie(movie_id) on update cascade on delete restrict,
  primary key (user_id, movie_id, start_date));
```

Buy Relation

a. Relational Model

buy(<u>user_id, movie_id,</u> purchase_date)

b. Functional Dependencies

user_id, movie_id -> purchase_date

c. Candidate Keys

{ user_id, movie_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table buy(
   user_id int not null,
   movie_id int not null,
   purchase_date date not null,
   foreign key user_id references member(user_id) on update cascade on delete restrict,
   foreign key movie_id references movie(movie_id) on update cascade on delete restrict,
   primary key (user_id, movie_id));
```

Review

a. Relational Model

review(<u>user_id, movie_id,</u> rating, comment)

b. Functional Dependencies

user_id, movie_id -> rating, comment

c. Candidate Keys

{user_id, movie_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table review(
  user_id int not null,
  movie_id int not null,
  rating numeric(2,1) not null,
  comment text,
  foreign key user_id references member(user_id) on update cascade on delete restrict,
  foreign key movie_id references movie(movie_id) on update cascade on delete restrict,
  primary key (user_id, movie_id));
```

Register Movie Relation

a. Relational Model

register_movie(<u>movie_id,</u> user_id, reg_movie_date)

b. Functional Dependencies

movie_id -> user_id, reg_movie_date

c. Candidate Keys

{movie_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table register_movie(
    movie_id int unique not null,
    user_id int not null,
    reg_movie_date date not null,
    foreign key movie_id references movie(movie_id) on update cascade on delete
restrict,
    foreign key user_id references member(user_id) on update cascade on delete restrict,
    primary key (movie_id));
```

Cancel Movie Relation

a. Relational Model

cancel_movie(movie_id, user_id, can_movie_date)

b. Functional Dependencies

movie_id -> user_id, can_movie_date

c. Candidate Keys

{movie_id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there is only one non-trivial functional dependency and the left-hand side of this dependency is a superkey.

e. Table Definition

```
create table cancel_movie(
  movie_id int unique not null,
  user_id int not null,
  can_movie_date date not null,
  foreign key movie_id references movie(movie_id) on update cascade on delete restrict,
  foreign key user_id references member(user_id) on update cascade on delete restrict,
  primary key (movie_id));
```

Movie - Genre Relation

a. Relational Model

movie_genre(movie_id, genre_id)

b. Functional Dependencies

There are only trivial functional dependencies in this table.

c. Candidate Keys

{movie_id,genre _id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there are only trivial functional dependencies in this table.

e. Table Definition

```
create table movie_genre(
  movie_id int not null,
  genre_id int not null,
  foreign key movie_id references movie(movie_id) on update cascade on delete restrict,
  foreign key genre_id references genre(genre_id) on update cascade on delete restrict,
  primary key (movie_id, genre_id));
```

Director - Movie Relation

a. Relational Model

movie_director(movie_id, director_id)

b. Functional Dependencies

There are only trivial functional dependencies in this table.

c. Candidate Keys

{movie_id,director _id}

d. Normal Form

This table is in BCNF (and 3NF) form, since there are only trivial functional dependencies in this table.

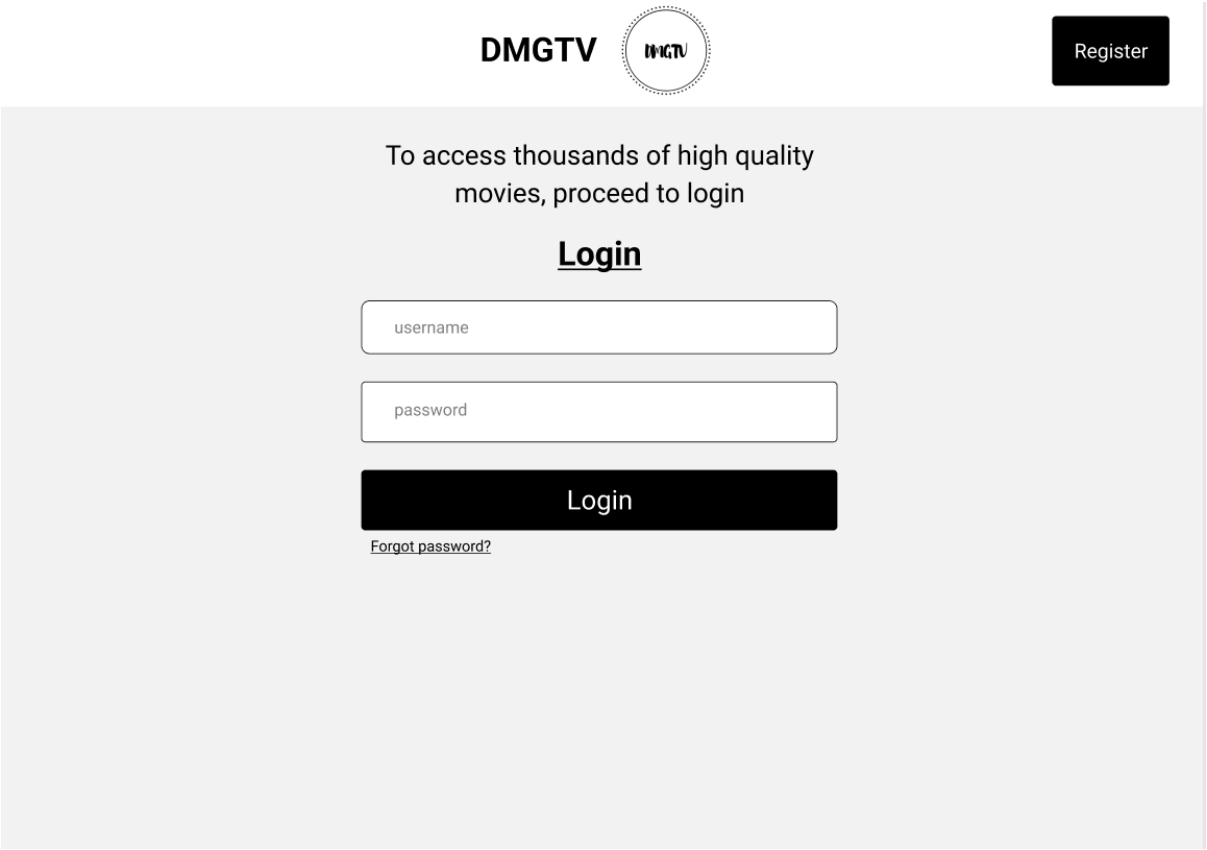
e. Table Definition

```
create table movie_director(
  movie_id int not null,
  director_id int not null,
  foreign key movie_id references movie(movie_id) on update cascade on delete restrict,
  foreign key director_id references director(director_id) on update cascade on delete restrict,
  primary key (movie_id, director_id));
```

User Interface Design and Corresponding SQL Statements

Note: symbol '@' represents a sample input.

Login

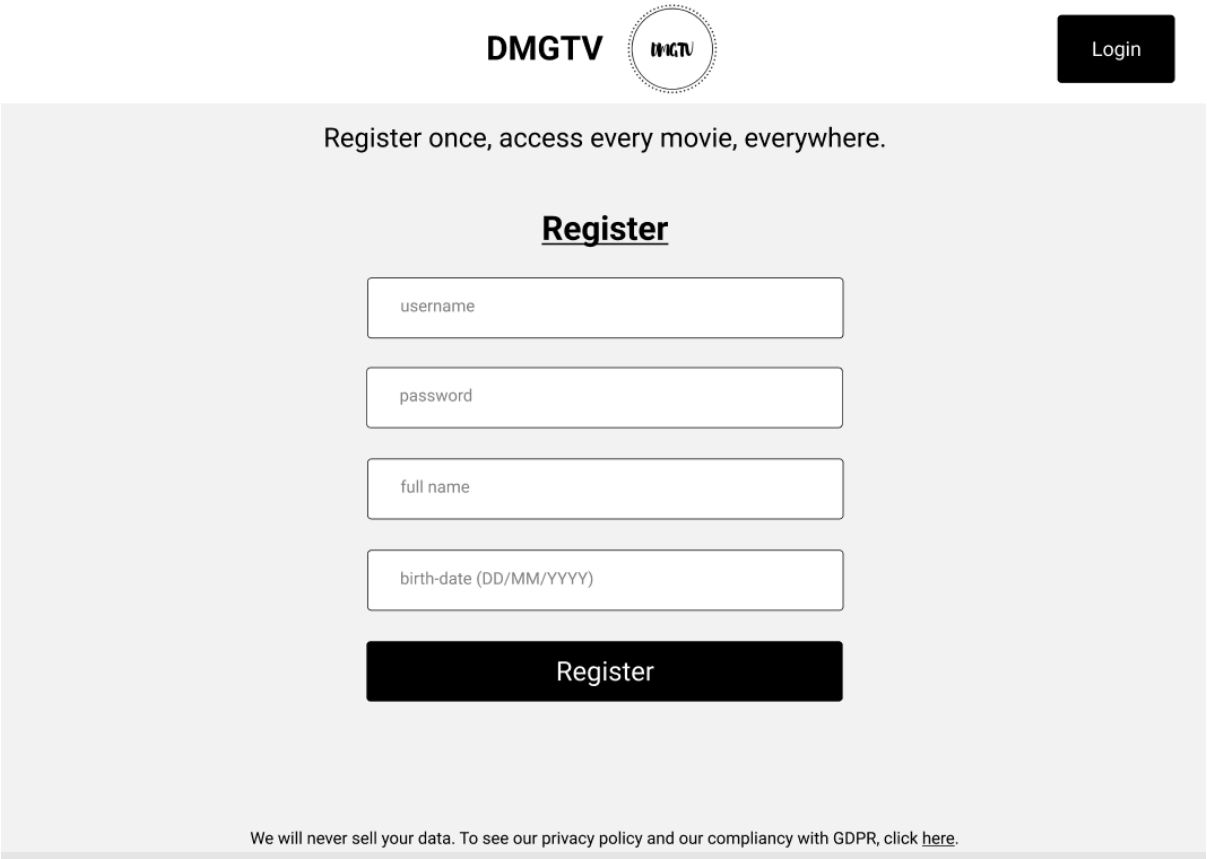


Users can login to the system from this page using their username and password. If they forgot their password, they can click "Forgot password?" and create a new one.

SQL Queries:

SELECT * FROM member WHERE username = @username AND password = @password;

Register



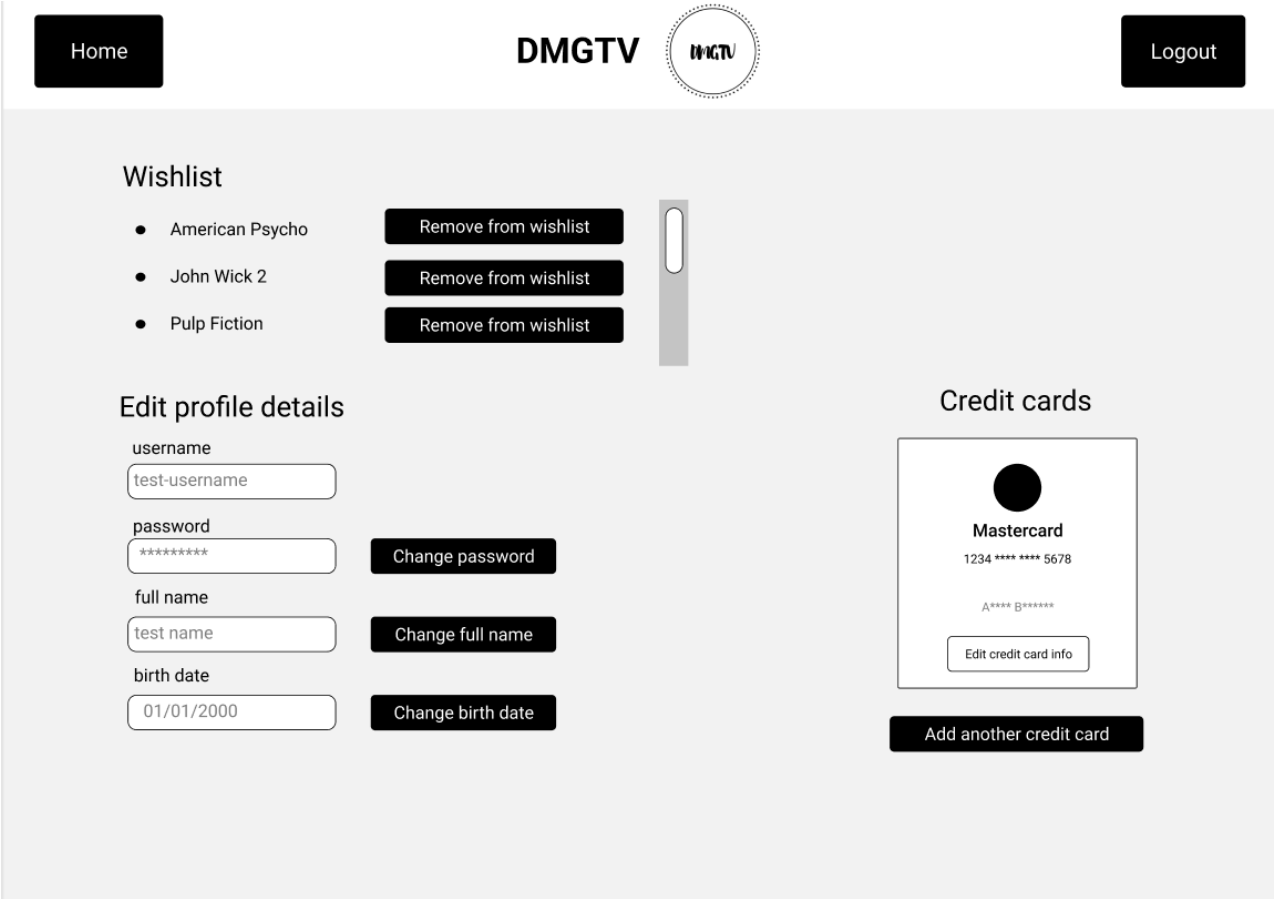
If users have not registered to the system before, they can register from this page by creating a username, password and entering their personal information which are their full name and birth date. After registering to the system, they can login.

SQL Queries:

INSERT INTO member (user_id, username, full_name, password, birth_date) VALUES (0, @username, @full_name, @password, @birth-date);

INSERT INTO customer (user_id) VALUES (0);

View Profile



Users can view their profile from this page. They can see their wishlist, edit their profile details, see their credit cards and add a new credit card or delete an existing credit card.

SQL Queries:

SELECT title FROM wish NATURAL JOIN movie WHERE user_id = @user_id;

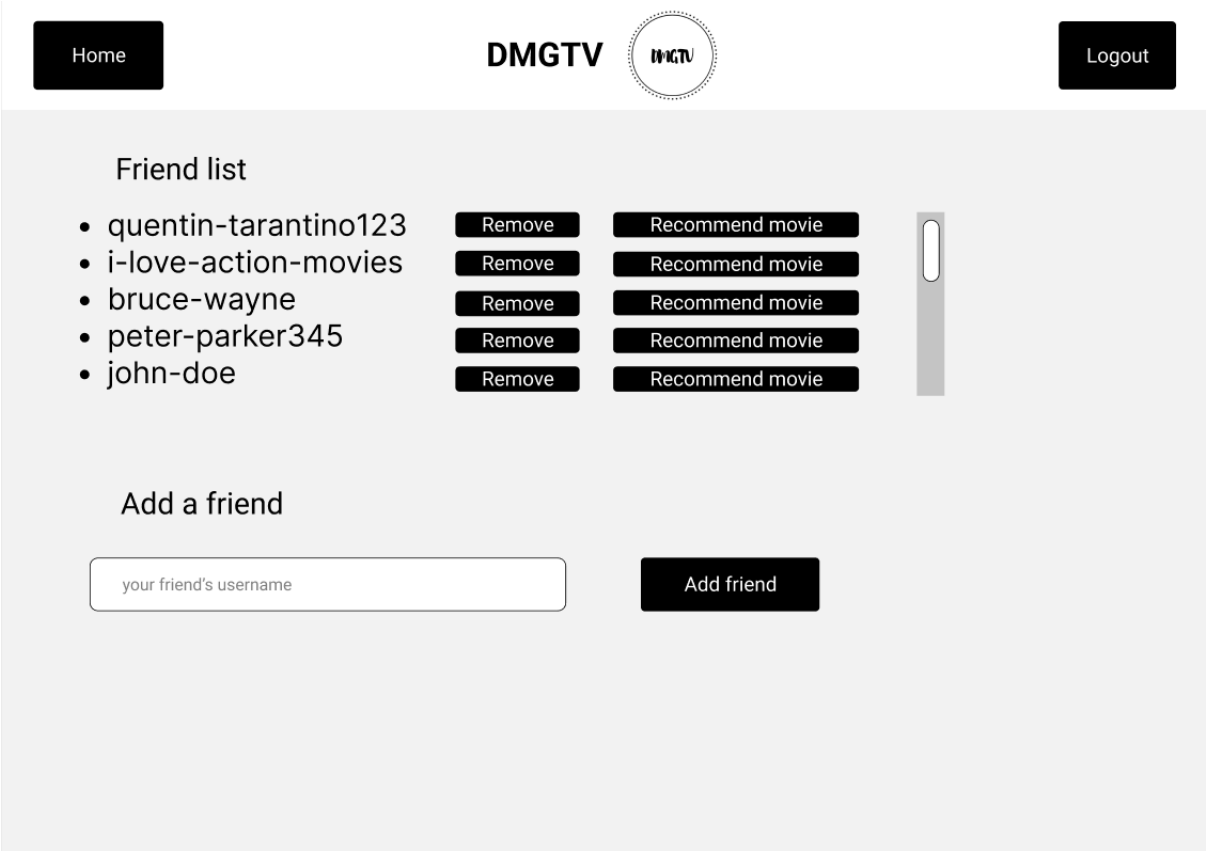
UPDATE member SET password = @password WHERE username = @username;

UPDATE member SET full_name = @name WHERE username = @username;

UPDATE member SET birth_date = @birth_date WHERE username = @username;

INSERT INTO credit_card(card_number, expiration_date, cvv) VALUES (@card_number, @expiration_date, @cvv);

Friends



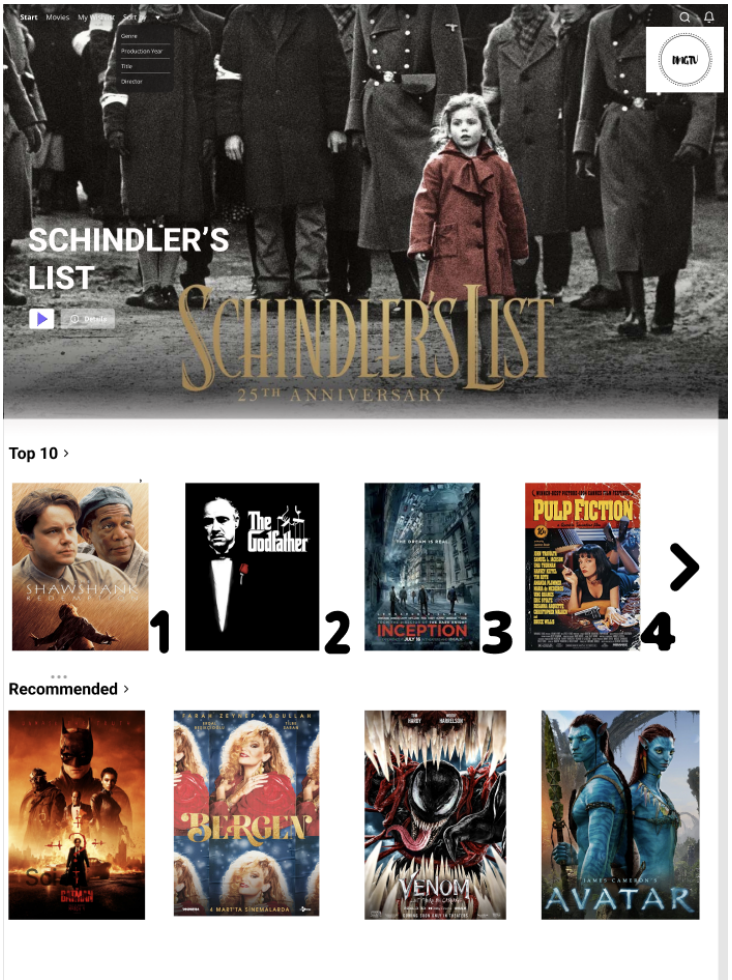
Users can see their added friends from this page and they can add a new friend. They can also remove an existing friend if they would like to and recommend movies to their added friends.

SQL Queries:

SELECT user_id as friend_user_id FROM member WHERE username = @friend_username;

INSERT INTO friends(<u>first_user_id, second_user_id</u>) VALUES (@user_id, @friend_user_id);

List Movies

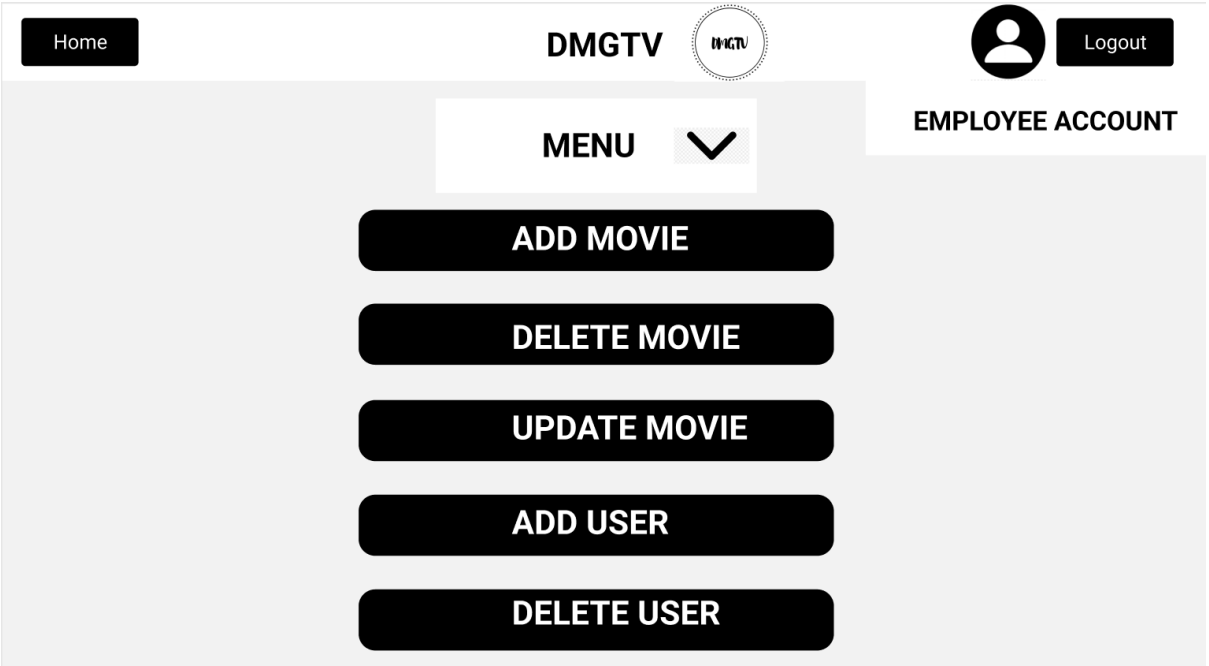


Users can see all of the movies in the system from this page. They can sort the movies by genre, production year, title, and director while searching for a movie. They can see the Top 10 movie list and recommended movies from this page as well.

SQL Queries:

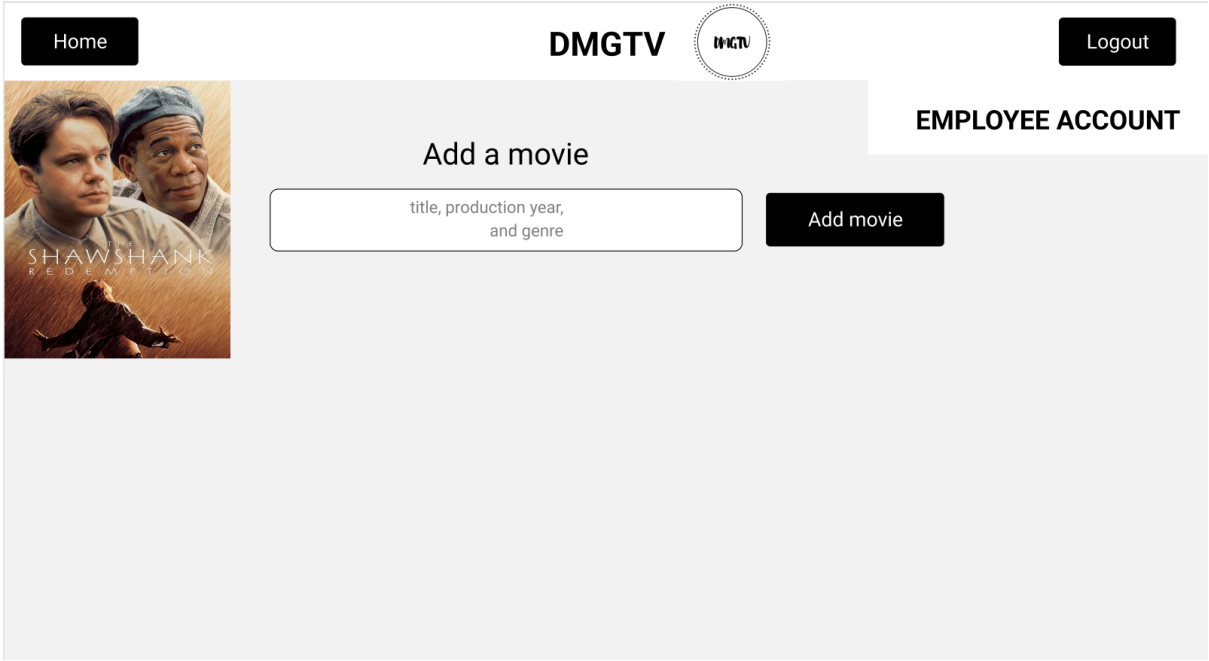
SELECT * FROM movie;

Employee Main Page

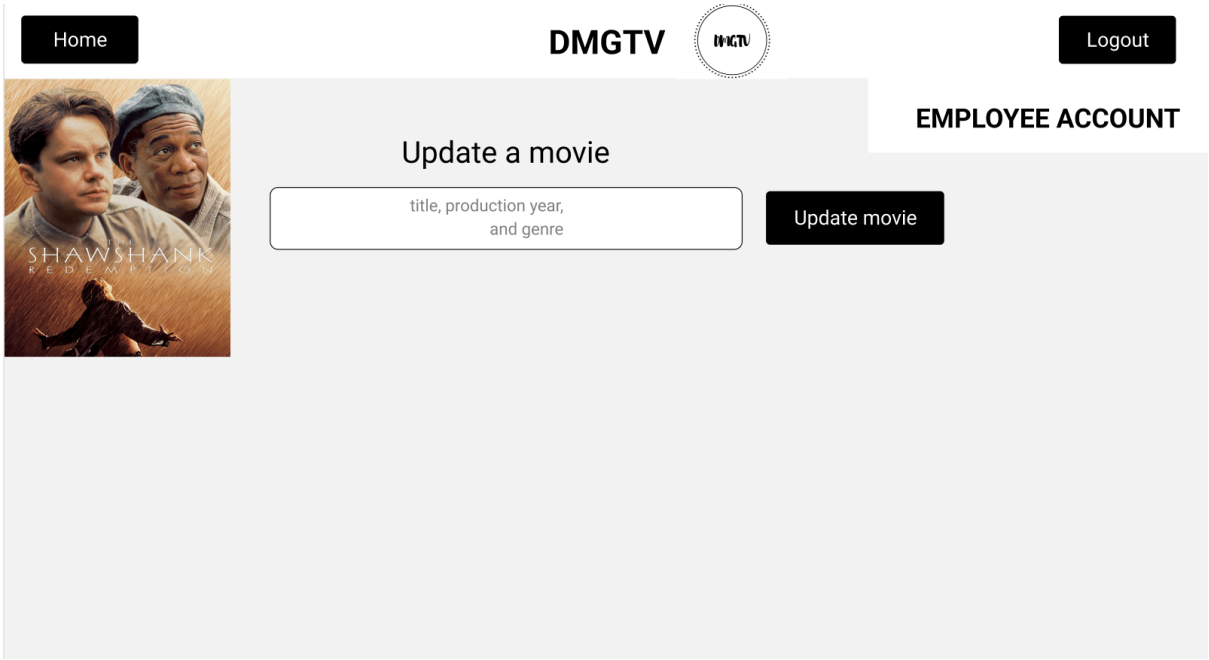


Employees can add movies to the system, update a movie or delete a movie from the system from this page. Additionally, this is the page where employees can manage users.

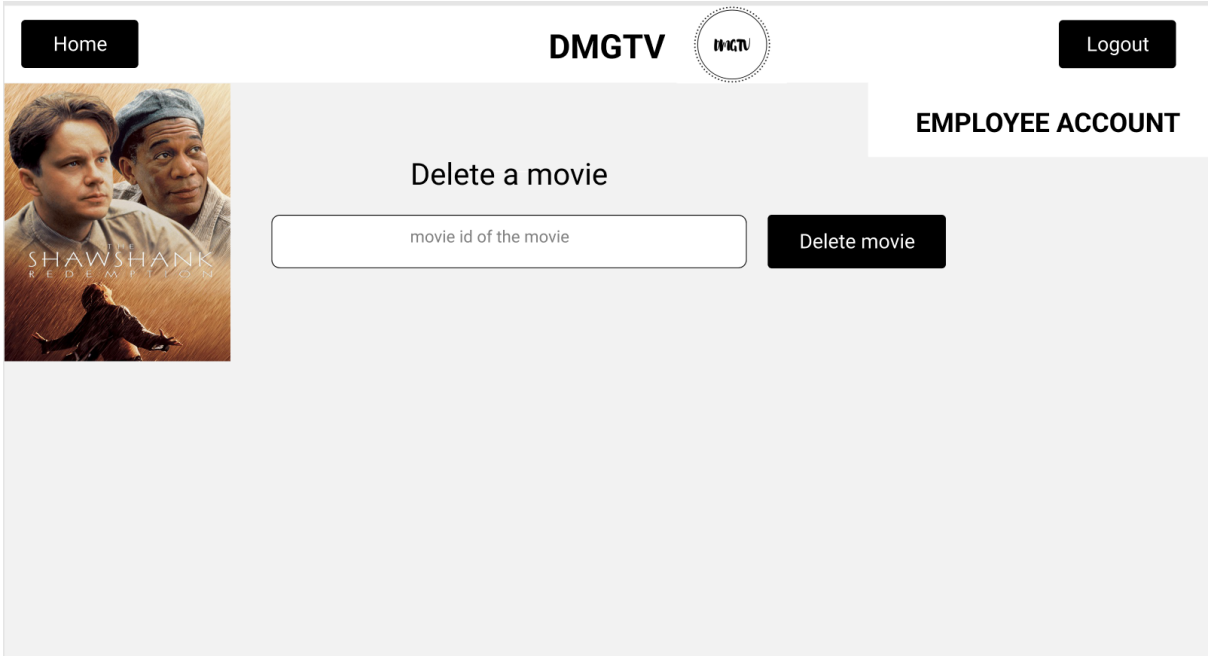
SQL Queries (following page):



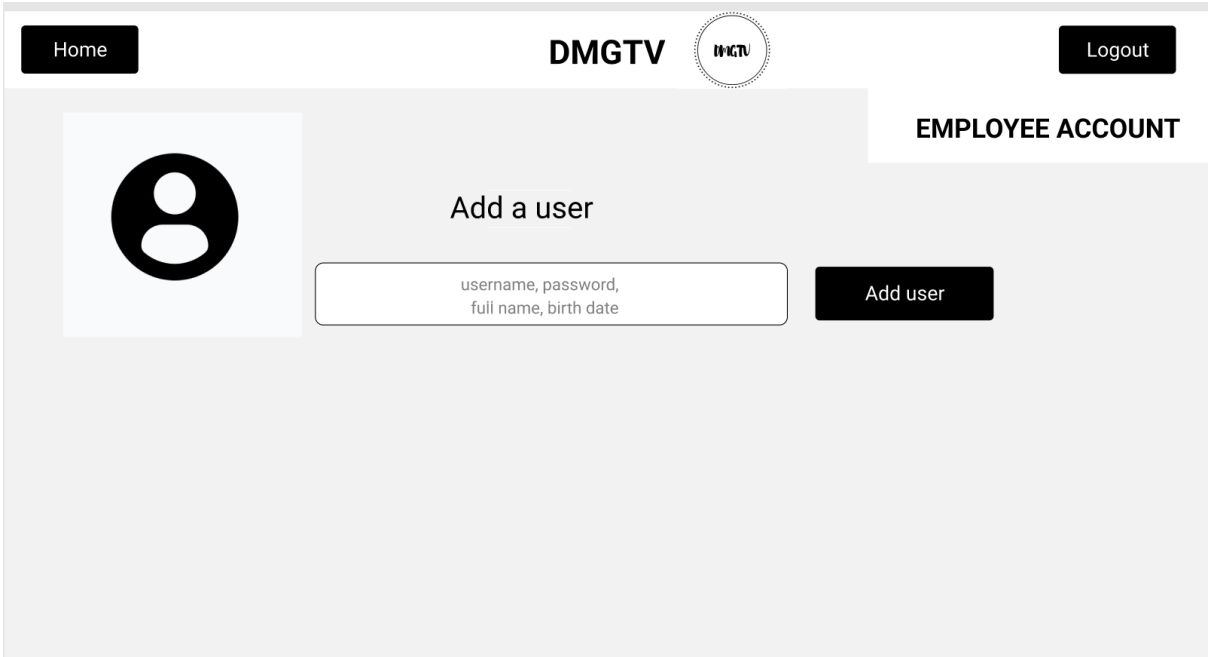
INSERT INTO movie(movie_id, title, production_year, rating, price_per_month, price_to_buy, age_restricted, imdb_rating, like_count) VALUES (0, @title, @production_year, @rating, @price_per_month, @price_to_buy, @age_restricted, @imdb_rating, @like_count);



UPDATE movie SET price_per_month = @price_per_month, price_to_buy = @price_to_buy, imdb_rating = @imdb_rating, age_restricted = @age_restricted where movie_id = @movie_id;



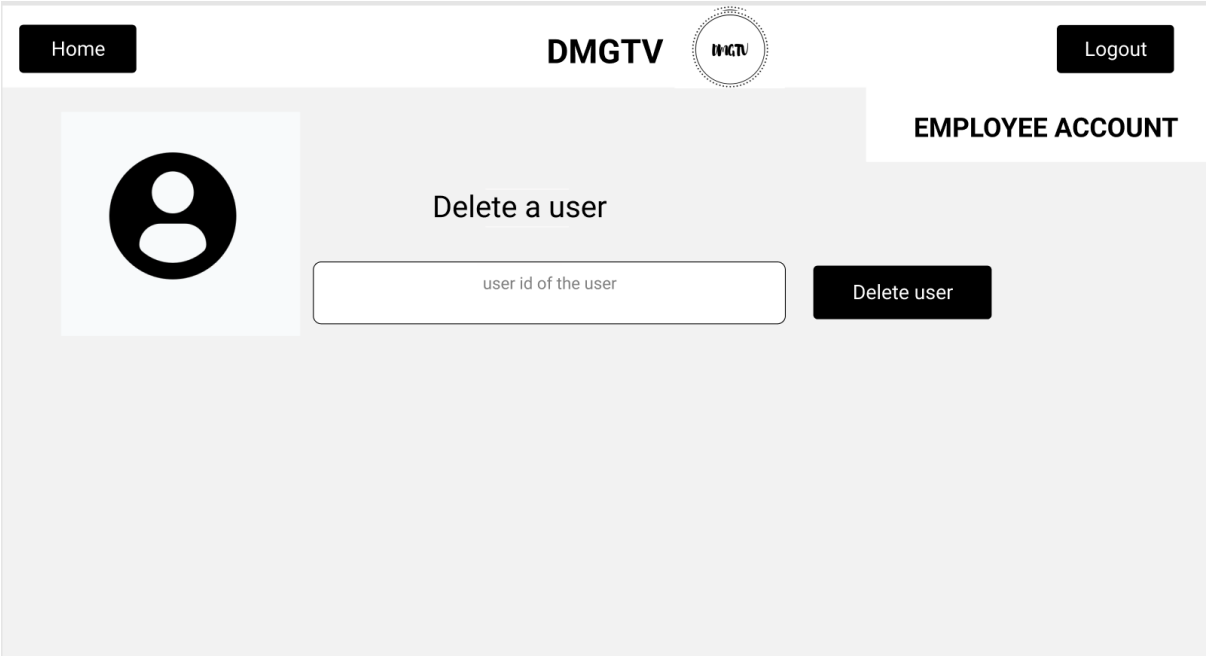
DELETE FROM movie WHERE movie_id = @movie_id;



INSERT INTO member (user_id, username, full_name, password, birth_date) VALUES (0, @username, @full_name, @password, @birth-date);

INSERT INTO customer (user_id) VALUES (0);

INSERT INTO registered_customer(customer_id, employee_id) VALUES(0, @employee_id);

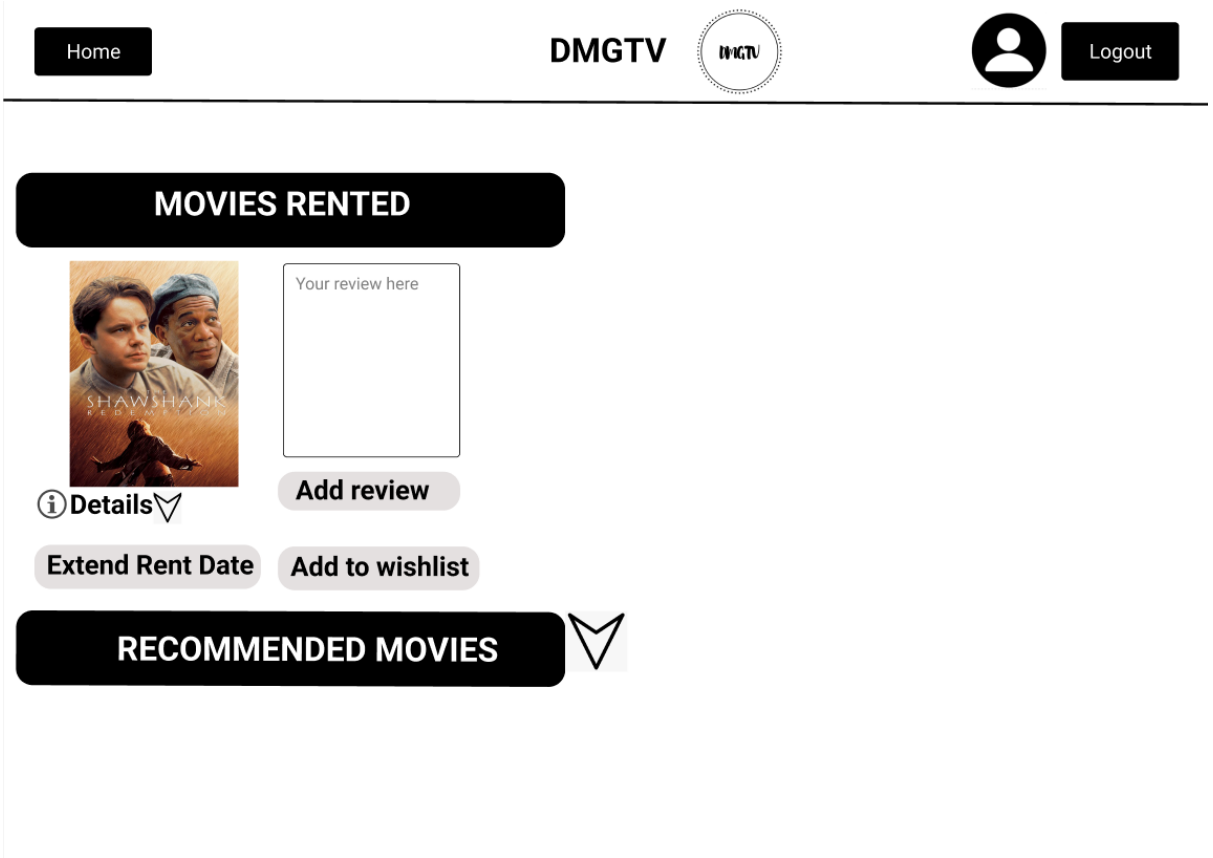


DELETE FROM customer WHERE user_id = @user_id;

DELETE FROM member WHERE user_id = @user_id;

INSERT INTO delete_customer(customer_id, employee_id, del_customer_date)
VALUES (@customer_id, @employee_id, @del_customer_date);

My Movies



Users can see the movies that they have rented from this page. They can extend the rent date of the movies if they would like to and they can see recommended movies by their friends from this page as well.

SQL Queries:

SELECT title FROM rent NATURAL JOIN movie WHERE user_id = @user_id;

Triggers

Update Movie Rating in Insertion

```
create trigger update_movie_rating_insertion
after insert on review
referencing new row as nrow
begin atomic
	update movie
		set movie.movie_rating =	(select sum(rating)
						from review
						where review.movie_id = nrow.movie_id
						)
						/
						(select count(*)
						from review
						where review.movie_id = nrow.movie_id
						)
	where movie.movie_id = nrow.movie_id
end;
```

Update Movie Rating in Deletion

```
create trigger update_movie_rating_deletion
after delete on review
referencing old row as orow
begin atomic
	update movie
		set movie.movie_rating =	(select sum(rating)
						from review
						where review.movie_id = orow.movie_id
						)
						/
						(select count(*)
						from review
						where review.movie_id = orow.movie_id
						)
	where movie.movie_id = orow.movie_id
end;
```

Update Like Count in Insertion

```
create trigger update_like_count_insertion
after insert on like
referencing new row as nrow
begin atomic
	update movie
		set movie.like_count =		(select count(*)
						from like
						where like.movie_id = nrow.movie_id
						)
	where movie.movie_id = nrow.movie_id
end;
```

Update Like Count in Deletion

```
create trigger update_like_count_deletion
after delete on like
referencing old row as orow
begin atomic
	update movie
		set movie.like_count =		(select count(*)
						from like
						where like.movie_id = orow.movie_id
						)
	where movie.movie_id = orow.movie_id
end;
```

Views

View All Movies Ordered By IMDB Rating

```
create view view_movies_imdb as
select movie.title, movie.imdb_rating
from movie
order by movie.imdb_rating desc
```

View All Movies Ordered By User Rating

```
create view view_movies_user as
select movie.title, movie.movie_rating
from movie
order by movie.movie_rating desc
```

View All Movies Ordered By Like Count

```
create view view_movies_like as
select movie.title, movie.movie_rating
from movie
order by movie.like_count desc
```

View Reviews for a Movie Ordered By User Rating

```
create view view_reviews as
select review.comment, movie.title, member.full_name
from review, movie, member
where review.movie_id = movie.movie_id and review.user_id = member.user_id
order by review.rating desc
```