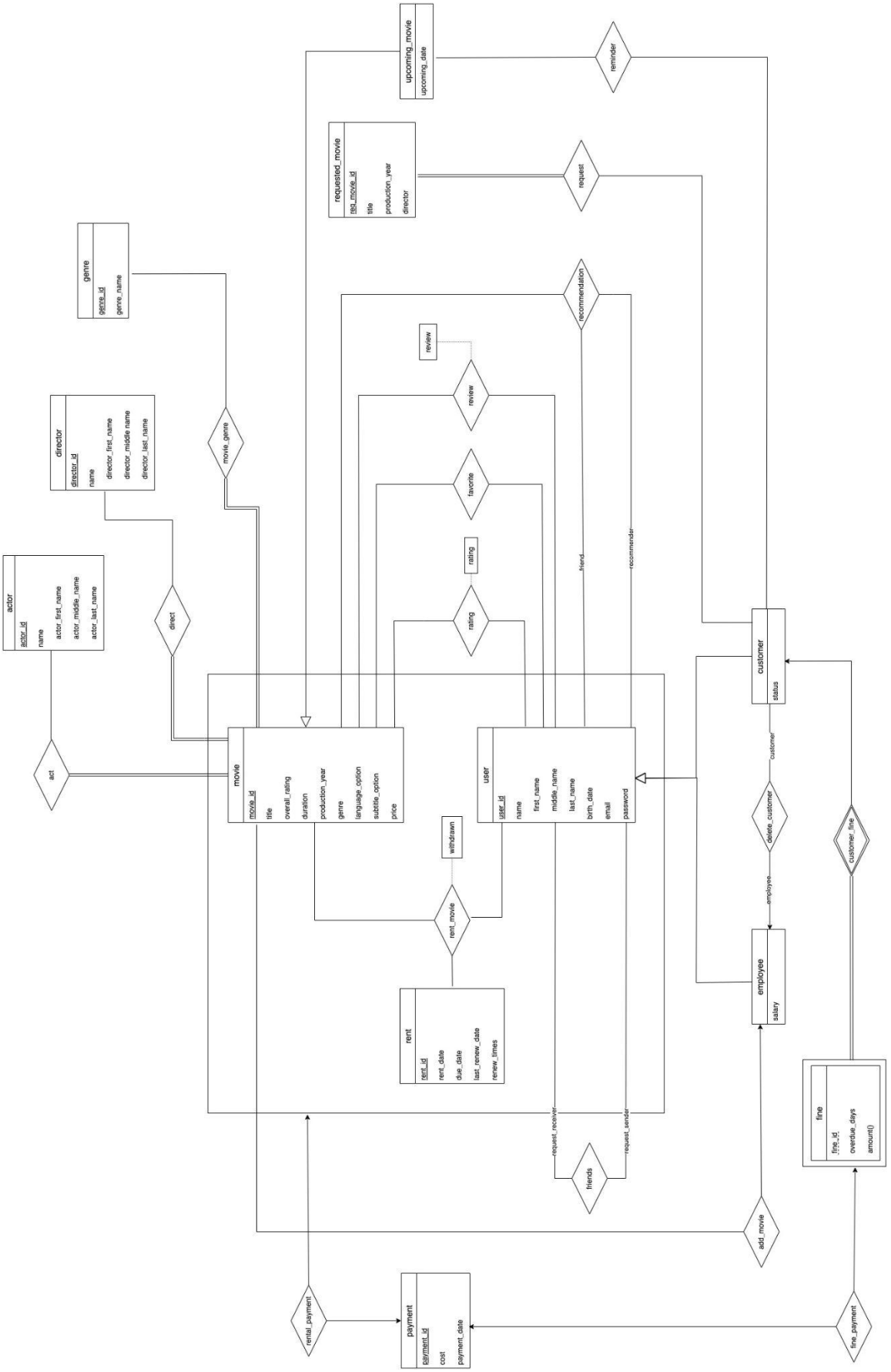
CS 353 - DATABASE SYSTEMS TERM PROJECT DESIGN REPORT

Group 22

Esra Genç	21901962, Section 3
Doğa Ece Ersoy	21902487, Section 3
Ayberk Yaşa	21801847, Section 3
Fatih Kaplama	21802755, Section 2

TA: Zülal Bingöl

1. Revised E-R Diagram



Revised Parts

- Rent relationship is switched to an entity. As a result, rent_movie relationship is created.
- Forgotten genre attribute is removed from the movie entity.
- fine_id in fine weak entity is underlined dotted.
- delete_customer relationship between customer and employee entities is created.
- add_movie relationship between employee and movie entities is created.
- Attributes email and password are added to the user entity.
- withdrawn attribute is added to rent_movie relation.

2. Table Schemas

· movie(movie_id, title, overall_rating, duration, production year, language_option, subtitle_option, price)

Primary Key:

movie_id

Candidate Keys:

movie_id

Functional Dependencies:

movie_id → title, overall_rating, duration, production year, language_option, subtitle_option, price

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE  IF NOT EXISTS movie (
     movie_id INT NOT NULL AUTO_INCREMENT,
     title VARCHAR(30) NOT NULL,
     overall_rating FLOAT,
     duration INT NOT NULL,
     production_year CHAR(4) NOT NULL,
     language_option VARCHAR(15) NOT NULL,
```

```
    subtitle_option VARCHAR(15),
    price INT NOT NULL,
    PRIMARY KEY(movie_id));
```

- rent(rent_id, rent_date, due_date, last_renew_date, renew_times)

  Primary Key:

  rent_id

  Candidate Keys:

  rent_id

  Functional Dependencies:

  rent_id → rent_date, due_date, last_renew_date, renew_times

  Normal Form:

  BCNF

  Table Declaration:

```
CREATE TABLE IF NOT EXISTS rent (
    rent_id INT NOT NULL AUTO_INCREMENT,
    rent_date DATE,
    due_date DATE,
    last_renew_date DATE,
    renew_times INT,
    PRIMARY KEY(rent_id));
```

- user(user_id, first_name, middle_name, last_name, birth_date, email, password)

  Primary Key:

  user_id

  Candidate Keys:

  user_id, email

Functional Dependencies:

user_id → first_name, middle_name, last_name, birth_date, email, password

email → user_id, first_name, middle_name, last_name, birth_date, password

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE IF NOT EXISTS user (
     user_id INT NOT NULL AUTO_INCREMENT,
    first_name VARCHAR(10) NOT NULL,
    middle_name VARCHAR(10),
    last_name VARCHAR(10) NOT NULL,
    birth_date DATE NOT NULL,
    email VARCHAR(25),
    password VARCHAR(20) NOT NULL,
    PRIMARY KEY(user_id));
```

- customer(user_id, status)

Primary Key:

user_id

Candidate Keys:

user_id

Foreign Keys:

user_id is FK to user

Functional Dependencies:

user_id → status

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE  IF NOT EXISTS customer (
     user_id INT NOT NULL,
     status VARCHAR(10) NOT NULL,
     PRIMARY KEY(user_id),
     FOREIGN KEY(user_id) REFERENCES user(user_id));
```

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

  Primary Key:

  user_id

  Candidate Keys:

  user_id

  Foreign Keys:

  user_id is FK to user

  Functional Dependencies:

  user_id → salary

  Normal Form:

  BCNF

  Table Declaration:

```
CREATE TABLE IF NOT EXISTS employee (
     user_id INT NOT NULL,
     salary INT NOT NULL,
     PRIMARY KEY(user_id),
     FOREIGN KEY(user_id) REFERENCES user(user_id));
```

- actor(actor_id, actor_first_name, actor_middle_name, actor_last_name)

  Primary Key:

  actor_id

  Candidate Keys:

  actor_id

  Functional Dependencies:

  actor_id → actor_first_name, actor_middle_name, actor_last_name

  Normal Form:

  BCNF

  Table Declaration:

```
CREATE TABLE IF NOT EXISTS actor (
     actor_id INT NOT NULL AUTO_INCREMENT,
     actor_first_name VARCHAR(10),
     actor_middle_name VARCHAR(10),
     actor_last_name VARCHAR(10) NOT NULL,
     PRIMARY KEY(actor_id));
```

- director(director_id, director_first_name, director_middle_name, director_last_name)

  Primary Key:

  director_id

  Candidate Keys:

  director_id

  Functional Dependencies:

  director_id → director_first_name, director_middle_name, director_last_name

  Normal Form:

  BCNF

Table Declaration:

```
CREATE TABLE IF NOT EXISTS director (
     director_id INT NOT NULL AUTO_INCREMENT,
     director_first_name VARCHAR(10),
     director_middle_name VARCHAR(10),
     director_last_name VARCHAR(10) NOT NULL,
     PRIMARY KEY(director_id));
```

- genre(genre_id, genre_name)

  Primary Key:

  genre_id

  Candidate Keys:

  genre_id, genre_name

  Functional Dependencies:

  genre_id → genre_name

  Normal Form:

  BCNF

  Table Declaration:

```
CREATE TABLE IF NOT EXISTS genre (
     genre_id INT NOT NULL AUTO_INCREMENT,
     genre_name VARCHAR(12) NOT NULL,
     PRIMARY KEY(genre_id));
```

- requested_movie(req_movie_id, title, production_year, director)

  Primary Key:

  req_movie_id

Candidate Keys:

req_movie_id

Functional Dependencies:

req_movie_id → title, production_year, director

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE IF NOT EXISTS requested_movie (
     req_movie_id INT NOT NULL AUTO_INCREMENT,
     title VARCHAR(30) NOT NULL,
     production_year CHAR(4) NOT NULL,
     director VARCHAR(30),
     PRIMARY KEY(req_movie_id));
```

- upcoming_movie(movie_id, upcoming_date)

Primary Key:

movie_id

Candidate Keys:

movie_id

Foreign Keys:

movie_id is FK to movie

Functional Dependencies:

movie_id → upcoming_date

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE IF NOT EXISTS upcoming_movie (
     movie_id INT NOT NULL,
     upcoming_date DATE NOT NULL,
     PRIMARY KEY(movie_id),
     FOREIGN KEY(movie_id) REFERENCES movie(movie_id));
```

- payment(payment_id, cost, payment_date)

  Primary Key:

  payment_id

  Candidate Keys:

  payment_id

  Functional Dependencies:

  payment_id → cost, payment_date

  Normal Form:

  BCNF

  Table Declaration:

```
CREATE TABLE payment (
     payment_id INT NOT NULL AUTO_INCREMENT,
     cost FLOAT,
     payment_date DATE,
     PRIMARY KEY(payment_id));
```

- rent_movie(rent_id, movie_id, user_id, withdrawn)

  Primary Key:

  {rent_id, movie_id, user_id}

  Candidate Keys:

{rent_id, movie_id, user_id}

Foreign Keys:

rent_id is FK to rent

movie_id is FK to movie

user_id is FK to user

Functional Dependencies:

rent_id, movie_id, user_id → withdrawn

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE rent_movie (
     movie_id INT,
     rent_id INT,
     user_id INT,
     withdrawn BOOLEAN,
     PRIMARY KEY(movie_id, rent_id, user_id),
     FOREIGN KEY (rent_id) REFERENCES rent(rent_id) on
UPDATE CASCADE ON DELETE RESTRICT),
     FOREIGN KEY (movie_id) REFERENCES movie(movie_id) on
UPDATE CASCADE ON DELETE RESTRICT),
     FOREIGN KEY (user_id) REFERENCES user(user_id) on
UPDATE CASCADE ON DELETE RESTRICT);
```

- act(movie_id, actor_id)

Primary Key:

{movie_id, actor_id}

Candidate Keys:

{movie_id, actor_id}

Foreign Keys:

movie_id is FK to movie

actor_id is FK to actor

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE IF NOT EXISTS act (

     movie_id INT NOT NULL,

     actor_id INT NOT NULL,

     PRIMARY KEY(movie_id, actor_id),

     FOREIGN KEY(movie_id) REFERENCES movie(movie_id) on
UPDATE CASCADE ON DELETE RESTRICT,

     FOREIGN KEY(actor_id) REFERENCES actor(actor_id) on
UPDATE CASCADE ON DELETE RESTRICT);
```

- direct(movie_id, director_id)

Primary Key:

{movie_id, director_id}

Candidate Keys:

{movie_id, director_id}

Foreign Keys:

movie_id is FK to movie

director_id is FK to director

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE IF NOT EXISTS direct (

     movie_id INT NOT NULL,

     director_id INT NOT NULL,

     PRIMARY KEY(movie_id, director_id),

     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);
```

- movie_genre(movie_id, genre_id)

  Primary Key:

  {movie_id, genre_id}

  Candidate Keys:

  {movie_id, genre_id}

  Foreign Keys:

  movie_id is FK to movie

  genre_id is FK to genre

  Normal Form:

  BCNF

  Table Declaration:

```
CREATE TABLE IF NOT EXISTS movie_genre (

     movie_id INT NOT NULL,

     genre_id INT NOT NULL,

     PRIMARY KEY(movie_id, genre_id),

     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);
```

- rating(<u>user_id</u>, <u>movie_id</u>, rating)

  Primary Key:

  {user_id, movie_id}

  Candidate Keys:

  {user_id, movie_id}

  Foreign Keys:

  user_id is FK to user

  movie_id is FK to movie

  Functional Dependencies:

  user_id, movie_id → rating

  Normal Form:

  BCNF

  Table Declaration:

```
CREATE TABLE IF NOT EXISTS rating (
     user_id INT NOT NULL,
     movie_id INT NOT NULL,
     rating FLOAT NOT NULL,
     PRIMARY KEY(user_id, movie_id),
     FOREIGN KEY(user_id) REFERENCES user(user_id) on
UPDATE CASCADE ON DELETE RESTRICT,
     FOREIGN KEY(movie_id) REFERENCES movie(movie_id) on
UPDATE CASCADE ON DELETE RESTRICT);
```

- favorite(<u>user_id</u>, <u>movie_id</u>)

  Primary Key:

  {user_id, movie_id}

Candidate Keys:

{user_id, movie_id}

Foreign Keys:

user_id is FK to user

movie_id is FK to movie

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE IF NOT EXISTS favorite (

     user_id INT NOT NULL,

     movie_id INT NOT NULL,

     PRIMARY KEY(user_id, movie_id),

     FOREIGN KEY(user_id) REFERENCES user(user_id) on
UPDATE CASCADE ON DELETE RESTRICT,

     FOREIGN KEY(movie_id) REFERENCES movie(movie_id) on
UPDATE CASCADE ON DELETE RESTRICT);
```

- review(<u>user_id</u>, <u>movie_id</u>, review)

Primary Key:

{user_id, movie_id}

Candidate Keys:

{user_id, movie_id}

Foreign Keys:

user_id is FK to user

movie_id is FK to movie

Functional Dependencies:

user_id, movie_id → review

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE IF NOT EXISTS review (

     user_id INT NOT NULL,

     movie_id INT NOT NULL,

     review VARCHAR(300) NOT NULL,

     PRIMARY KEY(user_id, movie_id),

     FOREIGN KEY(user_id) REFERENCES user(user_id) on
UPDATE CASCADE ON DELETE RESTRICT,

     FOREIGN KEY(movie_id) REFERENCES movie(movie_id) on
UPDATE CASCADE ON DELETE RESTRICT);
```

- recommendation(<u>recommender</u>, <u>friend</u>, <u>movie_id</u>)

Primary Key:

{recommender, friend, movie_id}

Candidate Key:

{recommender, friend, movie_id}

Foreign Keys:

recommender is FK to user

friend is FK to user

movie_id is FK to movie

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE IF NOT EXISTS recommendation (

     recommender INT NOT NULL,
```

```
    friend INT NOT NULL,

    movie_id INT NOT NULL,

    PRIMARY KEY(recommender, friend, movie_id),

    FOREIGN KEY(recommender) REFERENCES user(user_id) on
UPDATE CASCADE ON DELETE RESTRICT,

    FOREIGN KEY(friend) REFERENCES user(user_id) on
UPDATE CASCADE ON DELETE RESTRICT,

    FOREIGN KEY(movie_id) REFERENCES movie(movie_id) on
UPDATE CASCADE ON DELETE RESTRICT);
```

- request(user_id, req_movie_id)

  Primary Key:

  {user_id, req_movie_id}

  Candidate Keys:

  {user_id, req_movie_id}

  Foreign Keys:

  user_id is FK to customer

  req_movie_id is FK to requested_movie

  Normal Form:

  BCNF

  Table Declaration:

```
CREATE TABLE IF NOT EXISTS request (

    user_id INT NOT NULL,

    req_movie_id INT NOT NULL,

    PRIMARY KEY(user_id, req_movie_id),

    FOREIGN KEY(user_id) REFERENCES customer(user_id) on
UPDATE CASCADE ON DELETE RESTRICT,
```

```
    FOREIGN KEY(req_movie_id) REFERENCES
requested_movie(req_movie_id) on UPDATE CASCADE ON DELETE
RESTRICT);
```

- reminder(user_id, movie_id)

  Primary Key:

  {user_id, movie_id}

  Candidate Keys:

  {user_id, movie_id}

  Foreign Keys:

  user_id is FK to customer

  movie_id is FK to movie

  Normal Form:

  BCNF

  Table Declaration:

```
CREATE TABLE IF NOT EXISTS reminder (

    user_id INT NOT NULL,

    movie_id INT NOT NULL,

    PRIMARY KEY(user_id, movie_id),

    FOREIGN KEY(user_id) REFERENCES customer(user_id) on
UPDATE CASCADE ON DELETE RESTRICT,

    FOREIGN KEY(movie_id) REFERENCES
upcoming_movie(movie_id) on UPDATE CASCADE ON DELETE
RESTRICT);
```

- delete_customer(employee, customer)

  Primary Key:

  {employee, customer}

Candidate Keys:

{employee, customer}

Foreign Keys:

employee is FK to employee

customer is FK to customer

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE IF NOT EXISTS delete_customer (

     employee INT NOT NULL,

     customer INT NOT NULL,

     PRIMARY KEY(employee, customer),

     FOREIGN KEY(employee) REFERENCES employee(user_id)
on UPDATE CASCADE ON DELETE RESTRICT,

     FOREIGN KEY(customer) REFERENCES customer(user_id)
on UPDATE CASCADE ON DELETE RESTRICT);
```

- rental_payment(<u>payment_id</u>, <u>rent_id</u>)

Primary Key:

{payment_id, rent_id}

Candidate Keys:

{payment_id, rent_id}

Foreign Keys:

payment_id is FK to payment

rent_id is FK to rent

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE rental_payment (

     payment_id INT,

     rent_id INT,

     PRIMARY KEY(payment_id, rent_id),

     FOREIGN KEY (rent_id) REFERENCES rent(rent_id) on
UPDATE CASCADE ON DELETE RESTRICT,

     FOREIGN KEY (payment_id) REFERENCES
payment(payment_id) on UPDATE CASCADE ON DELETE
RESTRICT);
```

- friends(request_reciever, request_sender)

Primary Key:

{request_reciever, request_sender}

Candidate Keys:

{request_reciever, request_sender}

Foreign Keys:

request_reciever is FK to user

request_sender is FK to user

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE friends (

     request_receiver INT,

     request_sender INT,

     PRIMARY KEY(request_receiver, request_sender),

     FOREIGN KEY (request_receiver) REFERENCES
user(user_id) on UPDATE CASCADE ON DELETE RESTRICT,

     FOREIGN KEY (request_sender) REFERENCES
user(user_id) on UPDATE CASCADE ON DELETE RESTRICT);
```

- fine(user_id, fine_id, overdue_days, amount)

  Primary Key:

  {user_id, fine_id}

  Candidate Keys:

  {user_id, fine_id}

  Foreign Keys:

  user_id is FK to customer

  Normal Form:

  BCNF

  Table Declaration:

```
CREATE TABLE IF NOT EXISTS fine (
     user_id INT NOT NULL,
     fine_id INT UNIQUE NOT NULL,
     overdue_days INT NOT NULL,
     amount FLOAT,
     PRIMARY KEY(user_id, fine_id),
     FOREIGN KEY(user_id) REFERENCES customer(user_id));
```

- fine_payment(user_id, payment_id, fine_id)

  Primary Key:

  {user_id, payment_id, fine_id}

  Candidate Keys:

  {user_id, payment_id, fine_id}

  Foreign Keys:

  user_id is FK to fine

fine_id is FK to fine

payment_id is FK to payment

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE fine_payment (

     user_id INT,

     payment_id INT,

     fine_id INT,

     PRIMARY KEY(user_id, payment_id, fine_id),

     FOREIGN KEY (user_id) REFERENCES fine(user_id) on
UPDATE CASCADE ON DELETE RESTRICT,

     FOREIGN KEY (fine_id) REFERENCES fine(fine_id) on
UPDATE CASCADE ON DELETE RESTRICT,

     FOREIGN KEY (payment_id) REFERENCES
payment(payment_id) on UPDATE CASCADE ON DELETE
RESTRICT);
```

- add_movie(<u>user_id</u>, <u>movie_id</u>)

Primary Key:

{user_id, movie_id}

Candidate Keys:

{user_id, movie_id}

Foreign Keys:

user_id is FK to user

movie_id is FK to movie

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE add_movie (
     user_id INT,
     movie_id INT,
     PRIMARY KEY(user_id, movie_id),
     FOREIGN KEY (user_id) REFERENCES user(user_id) on
UPDATE CASCADE ON DELETE RESTRICT,
     FOREIGN KEY (movie_id) REFERENCES movie(movie_id) on
UPDATE CASCADE ON DELETE RESTRICT);
```

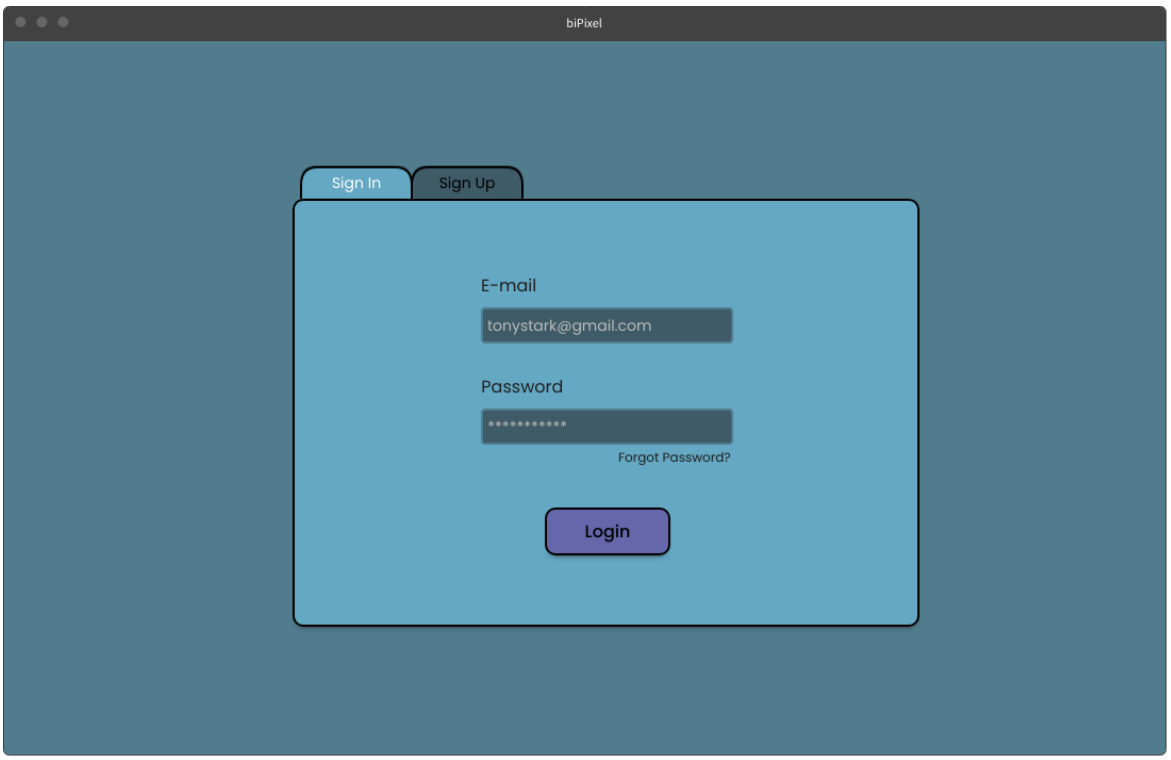
3. Scenarios

This part provides UIs and SQL queries of common functionalities that were given in the project functionality document and first topic-specific functionality which is the movie rental.

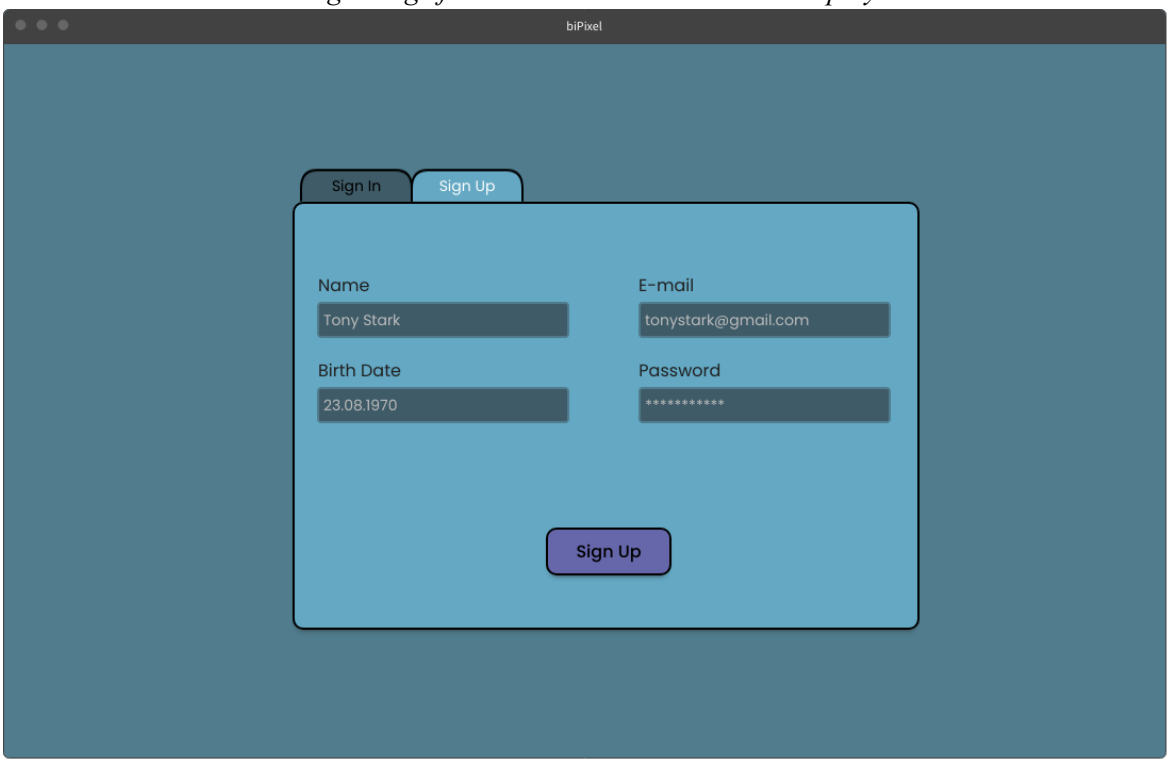
3.1. Login and Register

Login operation will be done on the same page for both the customer and the employee. Moreover, register operation will be done on the same page for both the customer and the employee. Therefore, there are two mock-ups in this section, which set light to the future frontend implementation.

3.1.1. UIs



Login Page for both the customer and the employee



Sign Up Page for both the customer and the employee

3.1.2. SQL Queries

a. Login

Inputs: @email, @password

Query:

```
SELECT *
FROM user
WHERE email = @email and password = @password
```

b. Register

Inputs: @first_name, @middle_name, @last_name, @birth_date, @email, @password

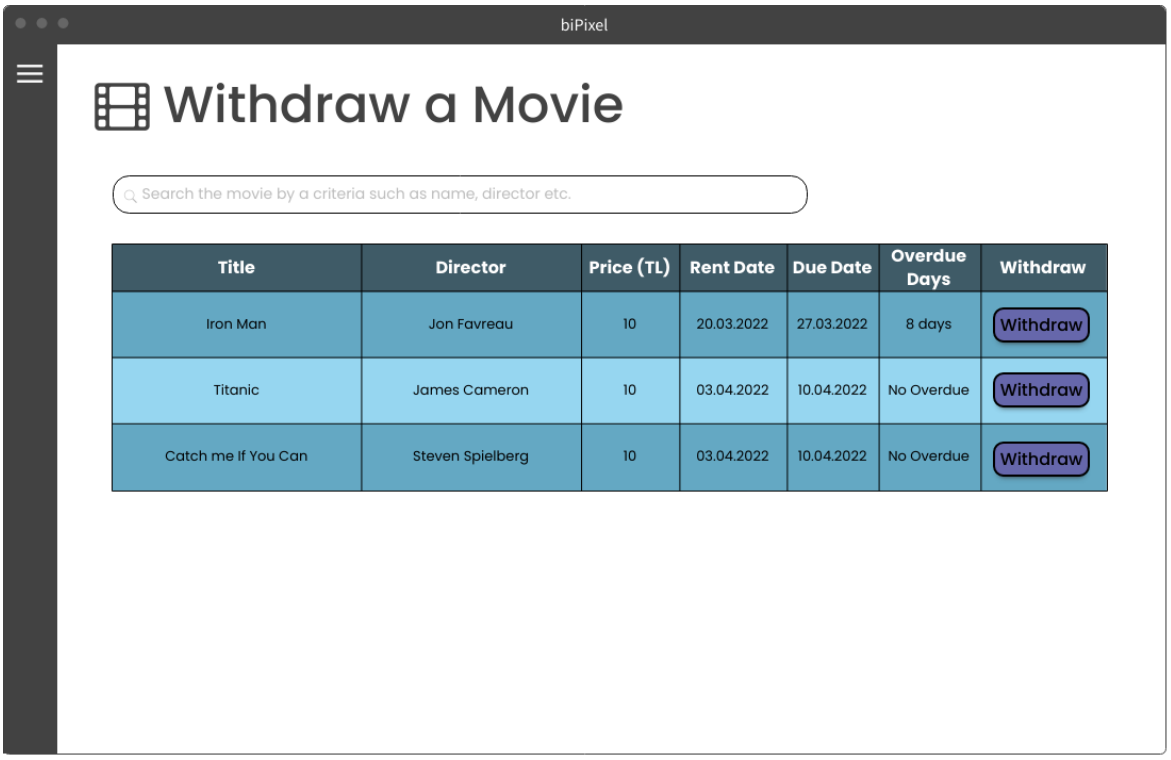
Query:

```
INSERT INTO user(first_name, middle_name, last_name,
birth_date, email, password)
VALUES(@first_name, @middle_name, @last_name,
@birth_date, @email, @password)
```

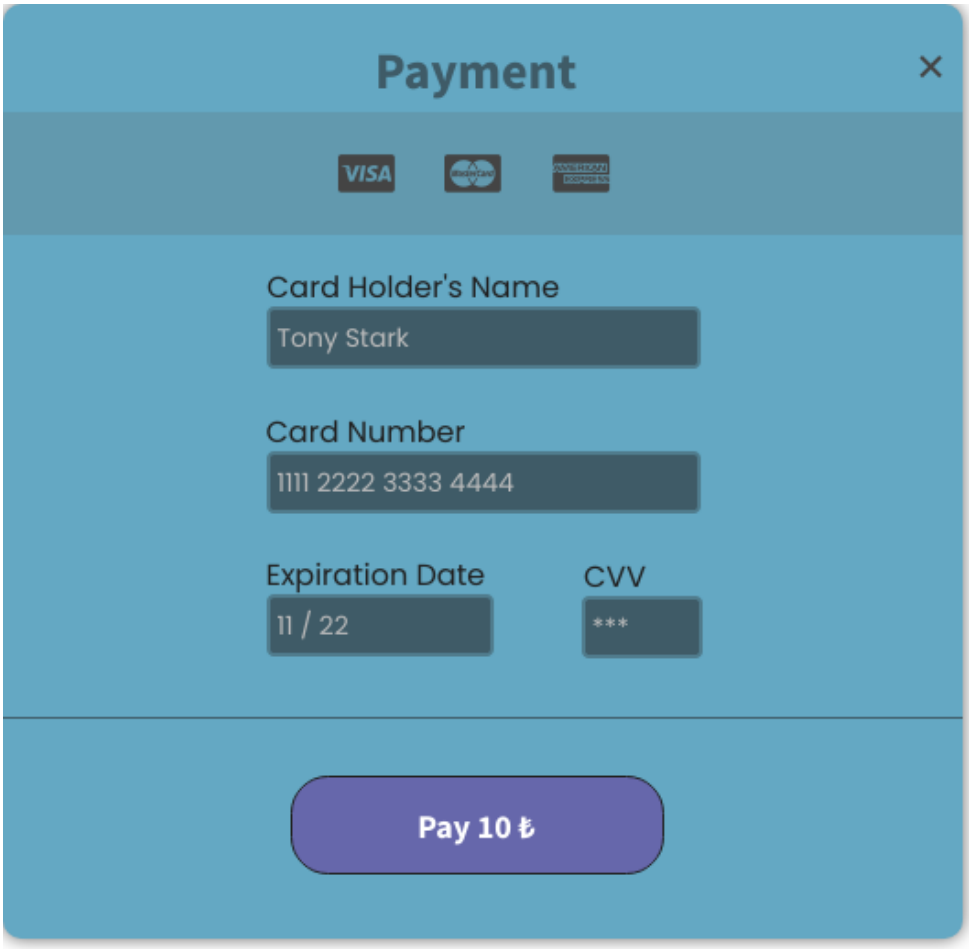
3.2. Fine Payment

A customer's movie rental period is 1 week and is determined automatically by the system. When withdrawing movies that exceed the due date, a fine is calculated based on the number of days passed from the due date and the customer is asked to pay. The customer cannot withdraw the movie without making fine payment. In the first UI below, the user who clicks the withdraw button corresponding to the movie s/he wants to withdraw will see the payment screen in the second UI. S/he can pay the fine there.

3.2.1. UIs



Withdraw Page



Payment Page

3.2.2. SQL Queries

a. Search the movie by a criteria such as name, director etc.

Inputs: @searchInput

Query:

```
SELECT title, director_first_name, director_middle_name,
director_last_name, genre_name, overall_rating,
production_year, price
FROM movie NATURAL JOIN act NATURAL JOIN actor NATURAL
JOIN direct NATURAL JOIN director NATURAL JOIN movie_genre
NATURAL JOIN genre
WHERE title = @searchInput OR actor_first_name =
@searchInput OR actor_middle_name = @searchInput OR
actor_last_name = @searchInput
OR director_first_name = @searchInput OR
director_middle_name = @searchInput OR director_last_name =
@searchInput;
```

b. Withdraw the movie

Inputs: @rentId (comes from the previous page),

Query:

```
UPDATE rent_movie
```

```
SET withdrawn = true
WHERE rentId = @rentId
```

c. Pay the fine

Inputs: @userId (comes from the previous page), @paymentId (comes from the previous page), @fineId (comes from the previous page),

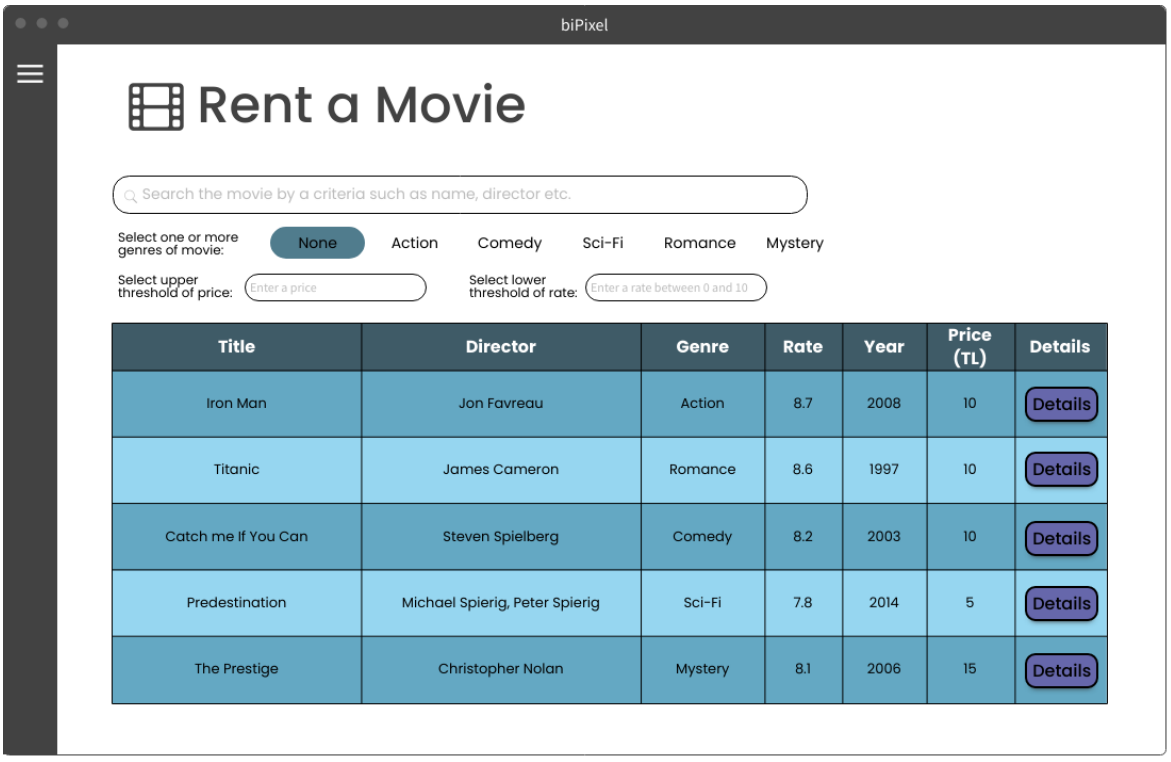
Query:

```
INSERT INTO fine_payment (user_id, payment_id, fine_id)
VALUES (@userId,@paymentId, @fineId);
```

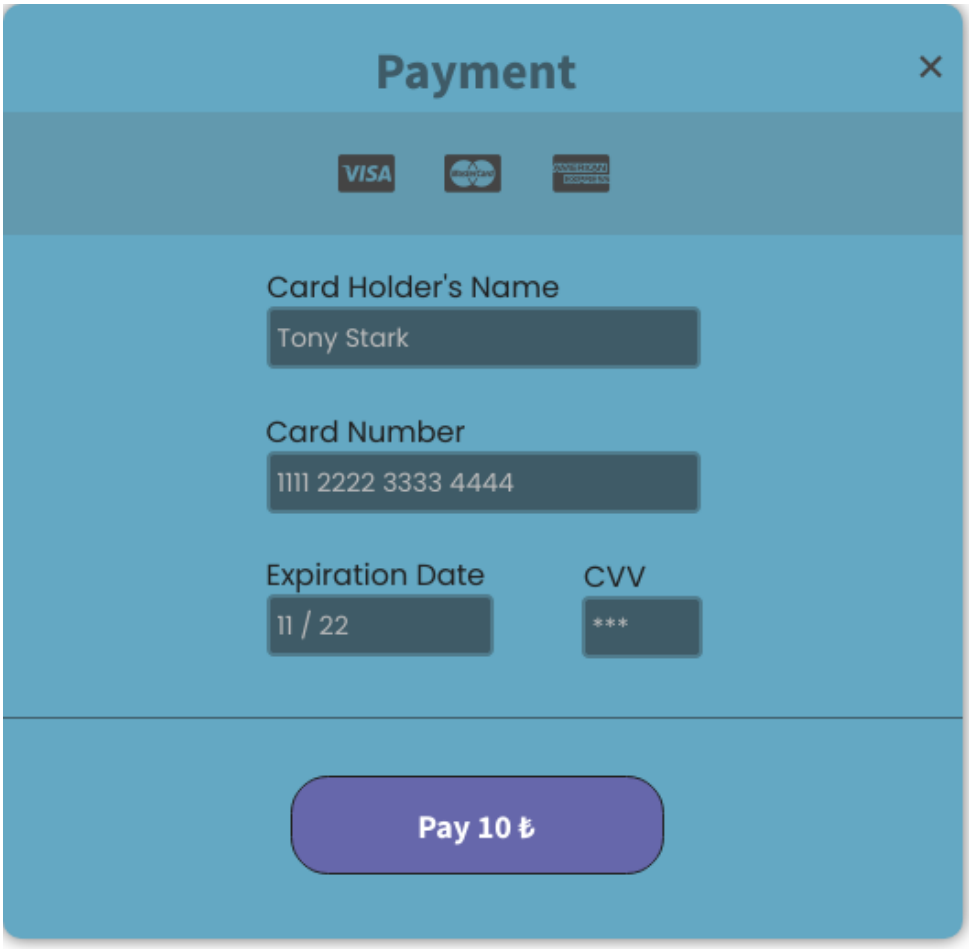
3.3. Movie Rental

The customer can search the movie by a criteria such as name, director, etc. Moreover, if s/he wants, s/he can apply a category filter to the list of movies. When s/he clicks the details button, s/he can see the pop-up containing the detailed information about the corresponding movie. Moreover, s/he can rent the movie from this pop-up. When s/he clicks the rent button, s/he can see the payment screen.

3.3.1. UIs



Rent Page



Payment Page

3.3.2. SQL Queries

a. Search the movie by a criteria such as name, director etc.

Inputs: @searchInput

Query:

```
SELECT title, director_first_name, director_middle_name,
director_last_name, genre_name, overall_rating,
production_year, price
FROM movie NATURAL JOIN act NATURAL JOIN actor NATURAL
JOIN direct NATURAL JOIN director NATURAL JOIN movie_genre
NATURAL JOIN genre
WHERE title = @searchInput OR actor_first_name =
@searchInput OR actor_middle_name = @searchInput OR
actor_last_name = @searchInput
OR director_first_name = @searchInput OR
director_middle_name = @searchInput OR director_last_name =
@searchInput;
```

b. Apply filters if necessary

Inputs: @genreInput, @ratingInput, @priceInput

Query:

```
#genre filter
SELECT title, director_first_name, director_middle_name,
director_last_name, genre_name, overall_rating,
production_year, price
FROM movie NATURAL JOIN direct NATURAL JOIN director
NATURAL JOIN movie_genre NATURAL JOIN genre
WHERE genre_name = @genreInput;

#rating filter
SELECT title, director_first_name, director_middle_name,
director_last_name, genre_name, overall_rating,
production_year, price
FROM movie NATURAL JOIN direct NATURAL JOIN director
NATURAL JOIN movie_genre NATURAL JOIN genre
WHERE overall_rating > @ratingInput;

#price filter
SELECT title, director_first_name, director_middle_name,
director_last_name, genre_name, overall_rating,
production_year, price
FROM movie NATURAL JOIN direct NATURAL JOIN director
NATURAL JOIN movie_genre NATURAL JOIN genre
WHERE price < @priceInput;
```

c. Select the movie (due date is specified automatically)

Inputs: No input

Query:

```
INSERT INTO rent (rent_date, due_date, last_renew_date,
renew_times)
VALUES (CURRENT_DATE(), DATE_ADD(CURRENT_DATE(), INTERVAL
7 DAY), CURRENT_DATE(), 0);

INSERT INTO rent_movie (movie_id, rent_id, user_id)
VALUES (@movieId, (SELECT MAX(rent_id) from rent),
@userId);
```

d. Pay the rent

Inputs: @movieId (comes from the previous page), @rentId (comes from the previous page)

Query:

```
INSERT INTO payment (cost, payment_date)
VALUES ((SELECT price FROM movie WHERE movie_id =
@movieId), CURRENT_DATE());

INSERT INTO rental_payment (payment_id, rent_id)
VALUES ((SELECT MAX(payment_id) FROM payment), @rentId);
```