CS 353

2024-2025 Fall Semester

Design Report

Group 8

Umay Dündar 22202573

Murathan Işık 22103416

Ali Deniz Sözer 22202218

Emre Uçar 22203675

Damla İmre 22001640

TABLE OF CONTENTS

1. Introduction

Many use public pools, just like any facility that works with an entry registration system. Many people are registered to a pool system: Regulars, irregulars, people working in the facility, and admins. Managing a pool comes with many responsibilities, varying from registering users to the system, managing bookings of the lanes and pools, scheduling lessons, and generating reports to see details of this information. However, these attributes are hard to manage manually, as many active pool systems are currently used, posing a challenge. These challenges highlight the need for a robust, automated system to streamline pool activity management, helping users and administrators achieve smooth operations.

2. Description Of The Application

In this project, we will create a pool facility system that serves many different users, such as member and nonmember swimmers, pool administrators, lifeguards, and coaches. In this application, all the swimmers can register for pool access and select lanes, book sessions, and register for swimming classes according to their level. Swimmers can also gain points that can be used to buy some items if they have a membership and they keep booking for the pool or registering for swimming classes. Swimmers can see their rating ranking among other swimmers in the system. Every swimmer can rate the swimming class coaches' instructor and see the coaches' overall rating. They can also comment on the swimming coaches' past performance. Swimmers and swimming coaches can see their swimming lesson schedules and available/unavailable daily time slots for accessing pools. Coaches can also see their ratings in the system. Administrators should be able to see coach ratings, feedback, swimmer points and rankings, available courses, and their course capacities. Admins can also generate reports for the demand of the pools with respect to their hours, most active members, and coach performances.

In this project, database usage is crucial since our application requires storing many different user types, their profile and lesson information and other necessary information like comments about coaches and courses or rating systems for member swimmers. Therefore, a database system is needed to handle all these data management tasks, including insertions, deletions, and modifications, according to the interaction of the user with the frontend part of our system.

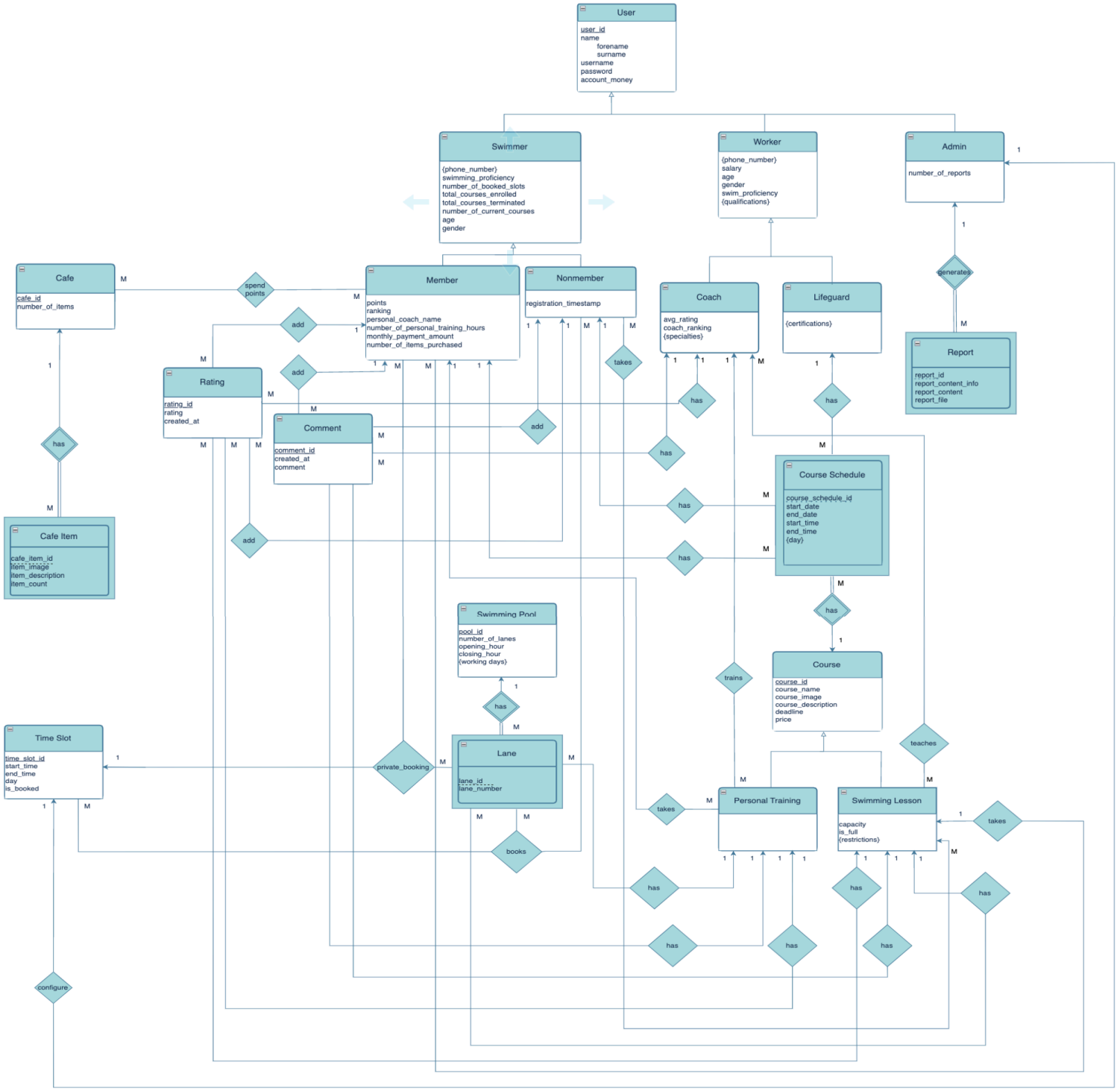
3. Revised ER Diagram

3.1. Changes

Here is a list of changes in our diagram from the previous version

- Missing primary keys are added in tables.
- Wrong arrow directions are fixed.
- Relationship cardinalities fixed.
- Additional username and account_money attributes are added to the user.
- Lane and Time Slot tables are connected with swimming lessons and personal training.
- Swimming lessons and personal training are connected to a more generalized entity course.
- The Report entity is made as a weak entity.
- Booking pool lane functionality fixed.
- Rating and Comment Tables entities are added to represent the commenting and rating functionality of our system.
- In the Cafe table, the menu attribute is fixed. Instead, the Cafe Item entity and a relationship between the Cafe and the Cafe Item table is introduced.
- Instead of the schedule attribute in swimmer and coach tables, the Course Schedule entity is introduced.

3.2. Revised ER Diagram



[1] https://drive.google.com/file/d/19T17ddcRAOtlJUwzh8Hsv1P1V_jbrDnp/view?usp=sharing

4. Database Schema

4.1 Table Schemas

4.1.1 users

Table Name: users

Relational Model: users(user_id, forename, surname, username, password, account_money)

Functional Dependencies: user_id -> forename, surname, username, password, account_money

Candidate Keys: {user_id}

Primary Key: user_id

Foreign Keys: none

Normal Form: BCNF

Table Definition:

```
CREATE TABLE users(
   user_id INT PRIMARY KEY,
   forename VARCHAR(255),
   surname VARCHAR(255),
   username VARCHAR(255),
   password VARCHAR(255),
   account_money BIGINT
);
```

4.1.2 swimmer

Table Name: swimmer

Relational Model: swimmer(swimmer_id, phone_number, age, gender, swimming_proficiency, number_of_booked_slots, total_courses_enrolled, total_courses_terminated)

Functional Dependencies: swimmer_id -> phone_number, age, gender, swimming_proficiency, number_of_booked_slots, total_courses_enrolled, total_courses_terminated

Candidate Keys: {swimmer_id}

Primary Key: swimmer_id

Foreign Keys: swimmer_id -> users(user_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE swimmer (
   swimmer_id INT PRIMARY KEY,
   phone_number VARCHAR(15),
   age INT,
   gender VARCHAR(100),
   swimming_proficiency VARCHAR(100),
   number_of_booked_slots INT,
   total_courses_enrolled INT,
   total_courses_terminated INT,
   FOREIGN KEY (swimmer_id) REFERENCES users(user_id)
);
```

4.1.3 member_swimmer

Table Name: member_swimmer

Relational Model: member_swimmer(swimmer_id, points, ranking, monthly_payment_amount, personal_coach_id, number_of_personal_training_hours, number_of_items_purchased)

Functional Dependencies: swimmer_id -> oints, ranking, monthly_payment_amount, personal_coach_id, number_of_personal_training_hours, number_of_items_purchased

Candidate Keys: {swimmer_id}

Primary Key: swimmer_id

Foreign Keys: swimmer_id -> swimmer(swimmer_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE member_swimmer(
   swimmer_id INT PRIMARY KEY,
   points INT,
   monthly_payment_amount INT,
   personal_coach_id INT,
   number_of_personal_training_hours INT,
   ranking INT,
   number_of_items_purchased INT,
   FOREIGN KEY (swimmer_id) REFERENCES swimmer(swimmerr_id),
   FOREIGN KEY (personal_coach_id) REFERENCES coach(coach_id)
);
```

4.1.4 nonmember_swimmer

Table Name: nonmember_swimmer

Relational Model: nonmember_swimmer(swimmer_id, registration_timestamp)

Functional Dependencies: swimmer_id -> registration_timestamp

Candidate Keys: {swimmer_id}, {registration_timestamp}

Primary Key: {swimmer_id}

Foreign Keys: swimmer_id -> swimmer(swimmer_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE nonmember_swimmer (
   swimmer_id INT PRIMARY KEY,
   registration_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (swimmer_id) REFERENCES swimmer(swimmer_id)
);
```

4.1.5 worker

Table Name: worker

Relational Model: worker(worker_id, pool_id, salary, age, gender, phone_number, swim_proficiency, qualifications)

Functional Dependencies: worker_id -> pool_id, salary, age, gender, phone_number, swim_proficiency, qualifications

Candidate Keys: {worker_id}

Primary Key: worker_id

Foreign Keys: worker_id -> users(user_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE worker (
   worker_id INT PRIMARY KEY,
   pool_id INT,
   salary INT,
   age INT,
   gender VARCHAR(100),
   phone_number VARCHAR(15),
   swim_proficiency VARCHAR(100),
   qualifications TEXT,
   FOREIGN KEY (worker_id) REFERENCES users(user_id),
   FOREIGN KEY (pool_id) REFERENCES swimming_pool(pool_id)
);
```

4.1.5 coach

Table Name: coach

Relational Model: coach(coach_id, avg_rating, coach_ranking, specialties)

Functional Dependencies: coach_id -> avg_rating, ranking, specialties

Candidate Keys: {coach_id}

Primary Key: coach_id

Foreign Keys: coach_id -> worker(worker_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE coach (
   coach_id INT PRIMARY KEY,
   avg_rating FLOAT,
   coach_ranking INT,
   specialties TEXT,
   FOREIGN KEY (coach_id) REFERENCES worker(worker_id)
);
```

4.1.6 lifeguard

Table Name: lifeguard

Relational Model: lifeguard(lifegfuard_id, certifications)

Functional Dependencies: lifeguard_id -> certifications

Candidate Keys: {lifeguard_id}

Primary Key: lifeguard_id

Foreign Keys: lifeguard_id -> worker(worker_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE lifeguard (
   lifeguard_id INT PRIMARY KEY,
   certifications TEXT,
   FOREIGN KEY (lifeguard_id) REFERENCES worker(worker_id)
);
```

4.1.7 administrator

Table Name: administrator

Relational Model: administrator(administrator_id, number_of_reports)

Functional Dependencies: administrator_id -> number_of_reports

Candidate Keys: {administrator_id}

Primary Key: administrator_id

Foreign Keys: administrator_id -> user(user_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE administrator (
   administrator_id INT PRIMARY KEY,
   number_of_reports INT,
   FOREIGN KEY (administrator_id) REFERENCES users(user_id)
);
```

4.1.8 report

Table Name: report

Relational Model: report(report_id, admin_id, report_content_info, report_content, report_file)

Functional Dependencies: report_id -> admin_id, report_content_info, report_content, report_file

Candidate Keys: {report_id}

Primary Key: report_id

Foreign Keys: admin_id -> administrator(administrator_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE report (
   report_id INT PRIMARY KEY,
   admin_id INT,
   report_content_info VARCHAR(255),
   report_content TEXT,
   report_file BYTEA,
   FOREIGN KEY (admin_id) REFERENCES administrator(administrator_id)
);
```

4.1.9 course

Table Name: course

Relational Model: course(course_id, course_name, course_image, coach_id, course_description, restrictions, deadline, pool_id, lane_id, price)

Functional Dependencies: course_id -> course_name, course_image, coach_id, course_description, restrictions, deadline, pool_id, lane_id, price

Candidate Keys: {course_id}, {coach_id, course_name}

Primary Key: course_id

Foreign Keys: coach_id -> coach(coach_id)
pool_id -> swimming_pool(pool_id)
lane_id -> lane(lane_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE course(
    course_id INT PRIMARY KEY,
    course_name VARCHAR(255) NOT NULL,
    course_image BYTEA,
    coach_id INT NOT NULL,
    course_description TEXT,
    restrictions VARCHAR(255),
    deadline DATE,
    pool_id INT NOT NULL,
    lane_id INT NOT NULL,
    price INT NOT NULL,
    FOREIGN KEY (coach_id) REFERENCES coach(coach_id),
    FOREIGN KEY (pool_id) REFERENCES swimming_pool(pool_id),
    FOREIGN KEY (lane_id) REFERENCES lane(lane_id),
);
```

4.1.10 course_schedule

Table Name: course_schedule

Relational Model: course_schedule(course_schedule_id, course_id, swimmer_id, coach_id, start_date, end_date, start_time, end_time, day)

Functional Dependencies: course_schedule_id -> course_id, swimmer_id, coach_id, start_date, end_date, start_time, end_time, day

Candidate Keys: {course_schedule_id}

Primary Key: course_schedule_id

Foreign Keys: course_id -> course(course_id)
coach_id -> coach(coach_id)
swimmer_id -> swimmer(swimmer_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE course_schedule(
    course_schedule_id INT PRIMARY KEY,
    course_id INT,
    swimmer_id INT,
    coach_id INT,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL,
    day VARCHAR(255),
    FOREIGN KEY (swimmer_id) REFERENCES swimmer(swimmer_id),
    FOREIGN KEY (coach_id) REFERENCES coach(coach_id),
    FOREIGN KEY (course_id) REFERENCES course(course_id)
);
```

4.1.11 personal_training

Table Name: personal_training

Relational Model: personal_training(training_id)

Functional Dependencies: training_id -> training_id (trivial dependency)

Candidate Keys: {training_id}

Primary Key: training_id

Foreign Keys: training_id -> course(course_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE personal_training(
   training_id INT PRIMARY KEY,
   FOREIGN KEY (training_id) REFERENCES course(course_id)
);
```

4.1.12 swimming_lesson

Table Name: swimming_lesson

Relational Model: swimming_lesson(lesson_id, capacity, is_full)

Functional Dependencies: lesson_id -> capacity, is_full

Candidate Keys: {lesson_id}

Primary Key: lesson_id

Foreign Keys: lesson_id -> course(course_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE swimming_lesson(
   lesson_id INT PRIMARY KEY,
   capacity INT NOT NULL,
   is_full BOOLEAN NOT NULL,
   FOREIGN KEY (lesson_id) REFERENCES course(course_id)
);
```

4.1.13 swimming_pool

Table Name: swimming_pool

Relational Model: swimming_pool(pool_id, number_of_lanes, opening_hour, closing_hour, working_days)

Functional Dependencies: pool_id -> number_of_lanes, opening_hour, closing_hour, working_days

Candidate Keys: {pool_id}

Primary Key: pool_id

Foreign Keys: none

Normal Form: BCNF

Table Definition:

```
CREATE TABLE swimming_pool(
   pool_id INT PRIMARY KEY,
   number_of_lanes INT NOT NULL,
   opening_hour TIME NOT NULL,
   closing_hour TIME NOT NULL,
   working_days VARCHAR(255) NOT NULL,
);
```

4.1.14 lane

Table Name: lane

Relational Model: lane(lane_id, pool_id, lane_number, lifeguard_id)

Functional Dependencies: lane_id -> pool_id, lane_number, lifeguard_id

Candidate Keys: {lane_id}

Primary Key: lane_id

Foreign Keys: pool_id -> swimming_pool(pool_id)
lifeguard_id -> lifeguard(lifeguard_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE lane(
   lane_id INT PRIMARY KEY,
   pool_id INT,
   lane_number INT,
   lifeguard_id INT,
   FOREIGN KEY (pool_id) REFERENCES swimming_pool(pool_id),
   FOREIGN KEY (lifeguard_id) REFERENCES lifeguard(lifeguard_id)
);
```

4.1.15 time_slot

Table Name: time_slot

Relational Model: time_slot(time_slot_id, start_time, end_time, lane_id, is_booked)

Functional Dependencies: time_slot_id -> start_time, end_time, lane_id, is_booked

Candidate Keys: {time_slot_id}

Primary Key: time_slot_id

Foreign Keys: lane_id -> lane(lane_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE time_slot(
   time_slot_id INT PRIMARY KEY,
   start_time TIME NOT NULL,
   end_time TIME NOT NULL,
   lane_id INT NOT NULL,
   is_booked BOOLEAN NOT NULL,
   FOREIGN KEY (lane_id) REFERENCES lane(lane_id)
);
```

4.1.16 cafe

Table Name: cafe

Relational Model: cafe(cafe_id, number_of_items, pool_id)

Functional Dependencies: cafe_id -> number_of_items, pool_id

Candidate Keys: {cafe_id}

Primary Key: cafe_id

Foreign Keys: pool_id -> swimming_pool(pool_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE cafe(
   cafe_id INT PRIMARY KEY,
   number_of_items INT,
   pool_id INT,
   FOREIGN KEY (pool_id) REFERENCES swimming_pool(pool_id)
);
```

4.1.17 cafe_item

Table Name: cafe_item

Relational Model: cafe_item(cafe_item_id, cafe_id, item_image, item_description, item_count)

Functional Dependencies: cafe_item_id -> cafe_id, item_image, item_description, item_count

Candidate Keys: {cafe_item_id}

Primary Key: cafe_item_id

Foreign Keys: cafe_id -> cafe(cafe_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE cafe_item(
   cafe_item_id INT PRIMARY KEY,
   cafe_id INT NOT NULL,
   item_image BYTEA,
   item_description VARCHAR(255),
   item_count INT,
   purchaser_id INT,
   FOREIGN KEY (cafe_id) REFERENCES cafe(cafe_id),
   FOREIGN KEY (purchaser_id) REFERENCES member_swimmer(swimmer_id)
);
```

4.1.18 rating

Table Name: rating

Relational Model: rating(rating_id, swimmer_id, coach_id, course_id, rating, created_at)

Functional Dependencies: rating_id -> swimmer_id, coach_id, course_id, rating, created_at

Candidate Keys: {rating_id}

Primary Key: rating_id

Foreign Keys: swimmer_id -> swimmer(swimmer_id), coach_id -> coach(coach_id), course_id -> course(course_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE rating(
   rating_id INT PRIMARY KEY,
   swimmer_id INT NOT NULL,
   coach_id INT NOT NULL,
   course_id INT NOT NULL,
   rating INT CHECK (rating BETWEEN 1 AND 5),
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (swimmer_id) REFERENCES swimmer(swimmer_id),
   FOREIGN KEY (coach_id) REFERENCES coach(coach_id),
   FOREIGN KEY (course_id) REFERENCES course(course_id)
);
```

4.1.19 comment

Table Name: comment

Relational Model: comment(comment_id, swimmer_id, coach_id, course_id, comment, created_at)

Functional Dependencies: comment_id -> swimmer_id, coach_id, course_id, comment, created_at

Candidate Keys: {comment_id}

Primary Key: comment_id

Foreign Keys: swimmer_id -> swimmer(swimmer_id), coach_id -> coach(coach_id), course_id -> course(course_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE comment(
   comment_id INT PRIMARY KEY,
   swimmer_id INT NOT NULL,
   coach_id INT NOT NULL,
   course_id INT NOT NULL,
   comment VARCHAR(255),
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (swimmer_id) REFERENCES swimmer(swimmer_id),
   FOREIGN KEY (coach_id) REFERENCES coach(coach_id),
   FOREIGN KEY (course_id) REFERENCES course(course_id)
);
```

4.1.20 cart

Table Name: cart

Relational Model: cart(cart_id, purchase_id, course_id, cafe_item_id, cafe_id)

Functional Dependencies: cart_id -> purchaser_id, course_id, cafe_item_id,cafe_id

Candidate Keys: {cart_id}

Primary Key: cart_id

Foreign Keys: purchaser_id -> swimmer(swimmer_id), course_id -> course(course_id), cafe_item_id -> cafe_item(cafe_item_id), cafe_id -> cafe(cafe_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE cart(
   cart_id INT PRIMARY KEY,
   purchaser_id INT NOT NULL,
   course_id INT,
   cafe_item_id INT,
   cafe_id INT,
   FOREIGN KEY (purchaser_id) REFERENCES swimmer(swimmer_id),
   FOREIGN KEY (course_id) REFERENCES course(course_id),
   FOREIGN KEY (cafe_item_id) REFERENCES cafe_item(cafe_item_id),
   FOREIGN KEY (cafe_id) REFERENCES cafe(cafe_id)
);
```

4.1.21 private_booking

Table Name: private_booking

Relational Model: private_booking(private_booking_id, swimmer_id, coach_id, lane_id, time_slot_id)

Functional Dependencies: private_booking_id → swimmer_id, coach_id, lane_id, time_slot_id

Candidate Keys: {private_booking_id}

Primary Key: private_booking_id

Foreign Keys: swimmer_id -> swimmer(swimmer_id), coach_id -> coach(coach_id), lane_id -> lane(lane_id)

Normal Form: BCNF

Table Definition:

```
CREATE TABLE private_booking (
   private_booking_id SERIAL PRIMARY KEY,
   swimmer_id INT,
   lane_id INT,
   time_slot_id INT,
   status VARCHAR(50) DEFAULT 'active',
   FOREIGN KEY (swimmer_id) REFERENCES swimmer(swimmer_id),
   FOREIGN KEY (lane_id) REFERENCES lane(lane_id),
   FOREIGN KEY (time_slot_id) REFERENCES time_slot(time_slot_id),
   UNIQUE (swimmer_id, lane_id, time_slot_id)
);
```

4.2 Views And Triggers

4.2.1 Views:

View Stores Of Instance Login:

```
CREATE VIEW user_login AS
SELECT user_id, username, password
FROM users
```

View for cafe transactions:

```
CREATE VIEW Cafe_User AS
SELECT user_id, points
FROM Member
```

View for course comments and ratings

```
CREATE VIEW course_comments_and_ratings AS
SELECT C.course_id, C.course_name, AVG(R.rating) AS avg_rating,
COUNT(Com.comment) AS
        comment_count
FROM course as C, rating and R, comment as Com
WHERE C.course_id = R.course_id AND C.course_id = Com.course_id
GROUP BY C.course_id
```

View administrator report for swimmers

```
CREATE VIEW swimmer_report AS
SELECT S.swimming_proficiency, S.number_of_booked_slots, S.total_courses_enrolled,
        S.total_courses_terminated
FROM swimmer as S
GROUP BY S.swimmer_id, S.swimming_proficiency
```

View the coach's schedule

```
CREATE VIEW coach_schedule AS
SELECT C.coach_id, C.name AS coach_name, CO.course_id, CO.course_name,
        CS.start_time, CS.end_time, CS.day, PT.training_id AS personal_training_id
FROM COACH C
        LEFT JOIN COURSE CO ON C.coach_id = CO.coach_id
        LEFT JOIN COURSE_SCHEDULE CS ON CO.course_id = CS.course_id
        LEFT JOIN PERSONAL_TRAINING PT ON PT.training_id = CO.course_id
WHERE CS.day IS NOT NULL
ORDER BY C.coach_id, CS.start_time;
```

4.2.2 Triggers:

A trigger that updates coaches' account money after the course is purchased by the swimmer

```
CREATE TRIGGER update_coach_account
AFTER INSERT ON course_schedule
REFERENCING NEW ROW AS nrow
FOR EACH ROW
BEGIN
        UPDATE users
        SET account_money = account_money + (SELECT price
                                                FROM course as C
                                                WHERE course.course_id =  nrow.course_id)
        WHERE users.user_id = nrow.coach_id
END;
```

A trigger that updates the course capacity after the swimmer purchase

```
CREATE TRIGGER update_capacity
AFTER INSERT ON course_schedule
REFERENCING NEW ROW AS nrow
FOR EACH ROW
BEGIN
        UPDATE swimming_lesson
        SET capacity = capacity - 1
        WHERE lesson_id = nrow.course_id
END;
```

A trigger that updates the cafe item amount in the cafe after swimmer purchase (cafe item amount is for keeping track of the number of cafe items of the same kind)

```
CREATE TRIGGER update_cafe_item_amount
AFTER DELETE ON cart
REFERENCING OLD ROW AS orow
FOR EACH ROW
BEGIN
        UPDATE cafe_item
        SET item_count = item_count -1
        WHERE orow.cafe_item_id IS NOT NULL AND cafe_item.cafe_item.cafe_item_id
                = orow.cafe_item_id
END;
```

A trigger that updates the total number of cafe items after the swimmer purchase (total number of cafe items is for keeping track of total number of cafe items of all types)

```
CREATE TRIGGER update_total_cafe_item_count
AFTER DELETE ON cart
REFERENCING OLD ROW AS orow
FOR EACH ROW
BEGIN
        UPDATE cafe
        SET number_of_items = number_of_items -1;
        WHERE orow.cafe_id IS NOT NULL AND cafe.cafe_id = orow.cafe_id
END;
```

A trigger that updates an administrator's total report count after report generation

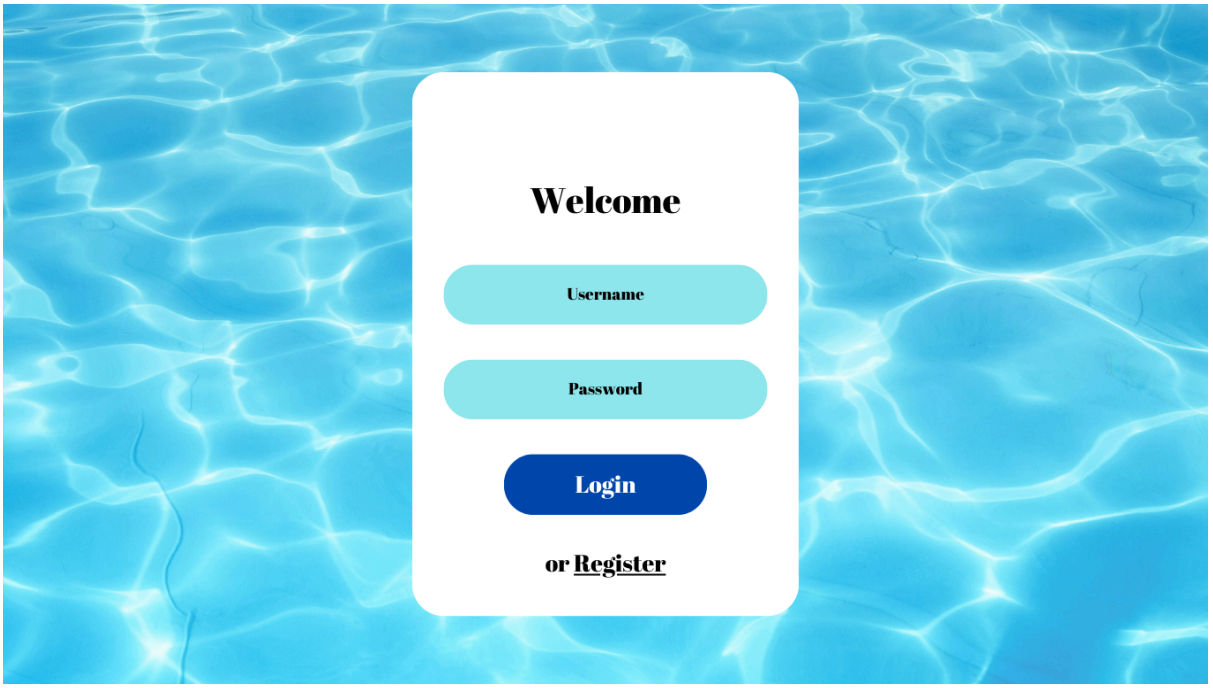
```
CREATE TRIGGER update_report_count
AFTER INSERT ON report
REFERENCING NEW ROW AS nrow
FOR EACH ROW
BEGIN
        UPDATE  administrator
        SET number_of_reports = number_of_reports + 1
        WHERE administrator.administrator_id = nrow.admin_id
END;
```

A trigger that updates an administrator's total report count after report deletion

```
CREATE TRIGGER update_report_count
AFTER DELETE ON report
REFERENCING NEW ROW AS nrow
FOR EACH ROW
BEGIN
        UPDATE  administrator
        SET number_of_reports = number_of_reports - 1
        WHERE administrator.administrator_id = nrow.admin_id
END;
```

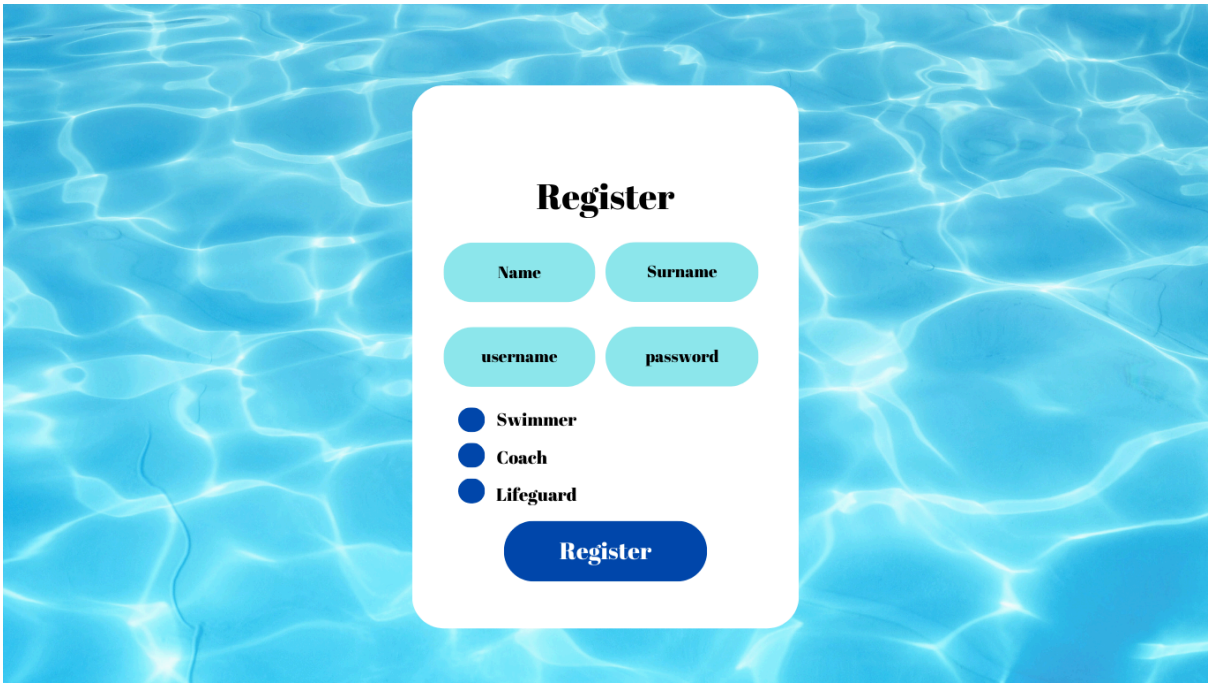
5. User Interface Design And Corresponding SQL Statements

5.1 Login Page:



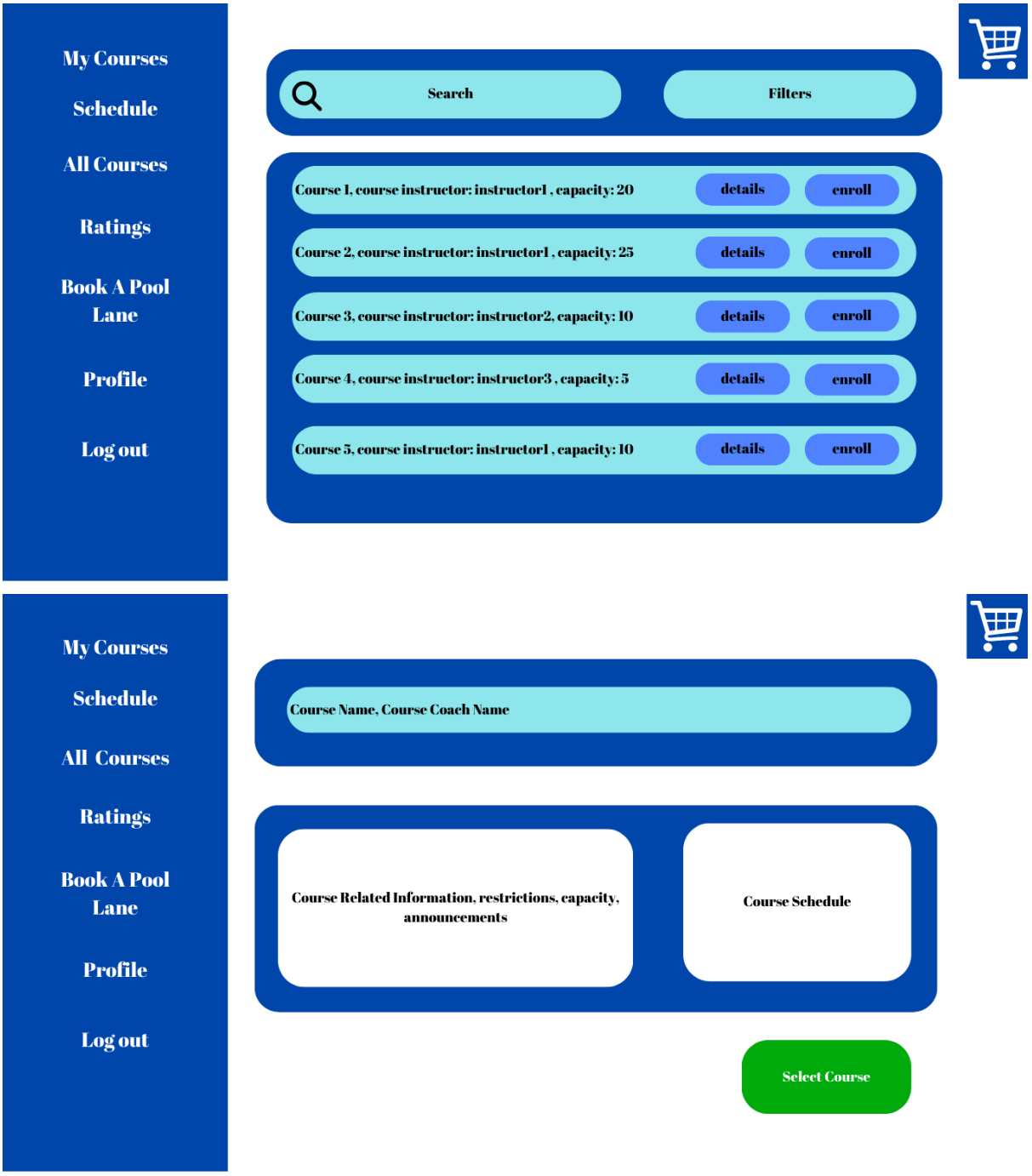
SELECT user_id, username, forename, surname, account_money FROM users WHERE user.username =username AND user.password = 'password';

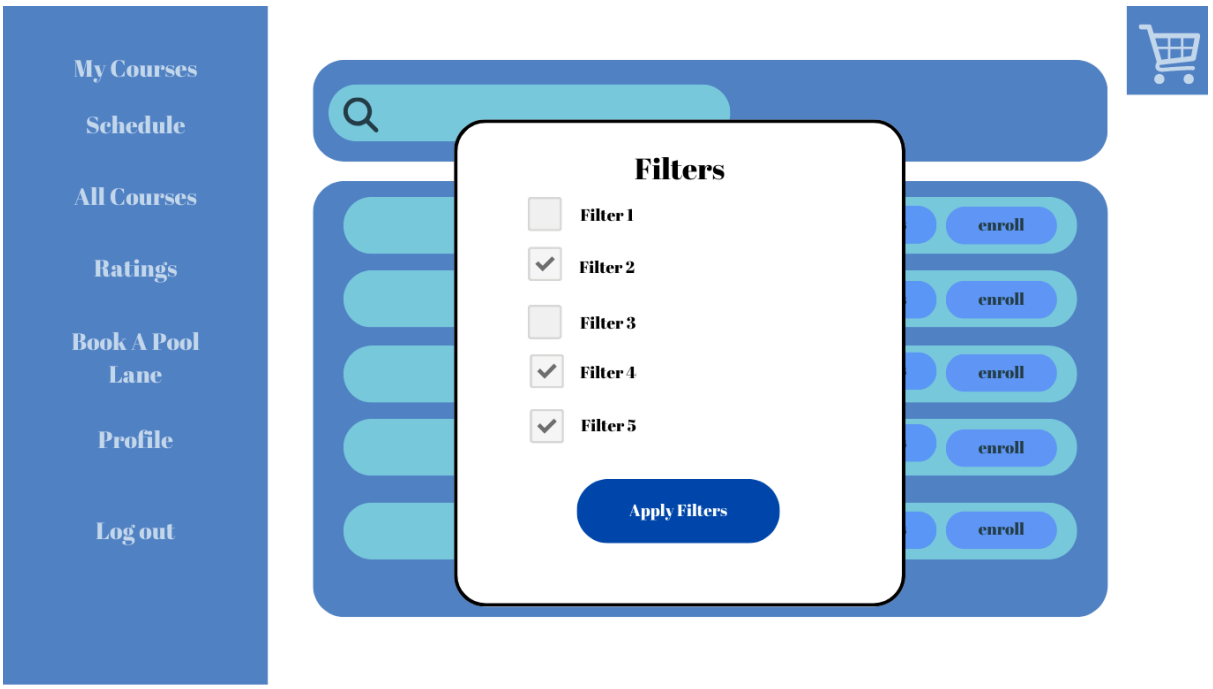
5.2 Register Page:

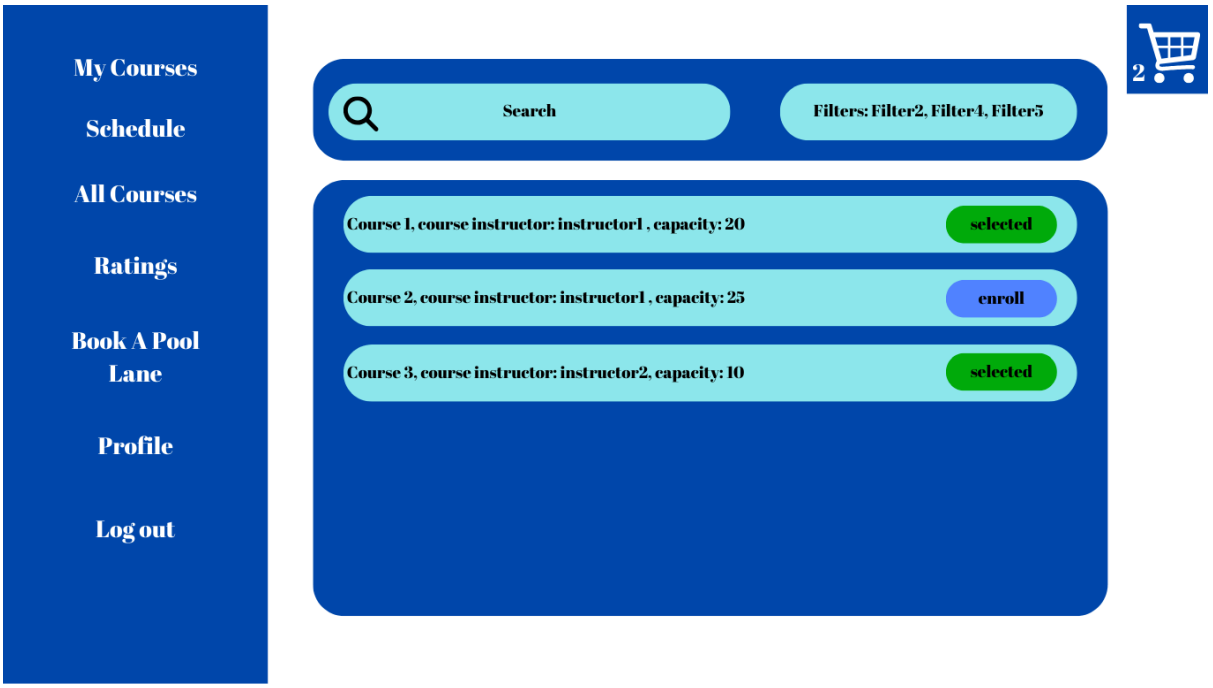


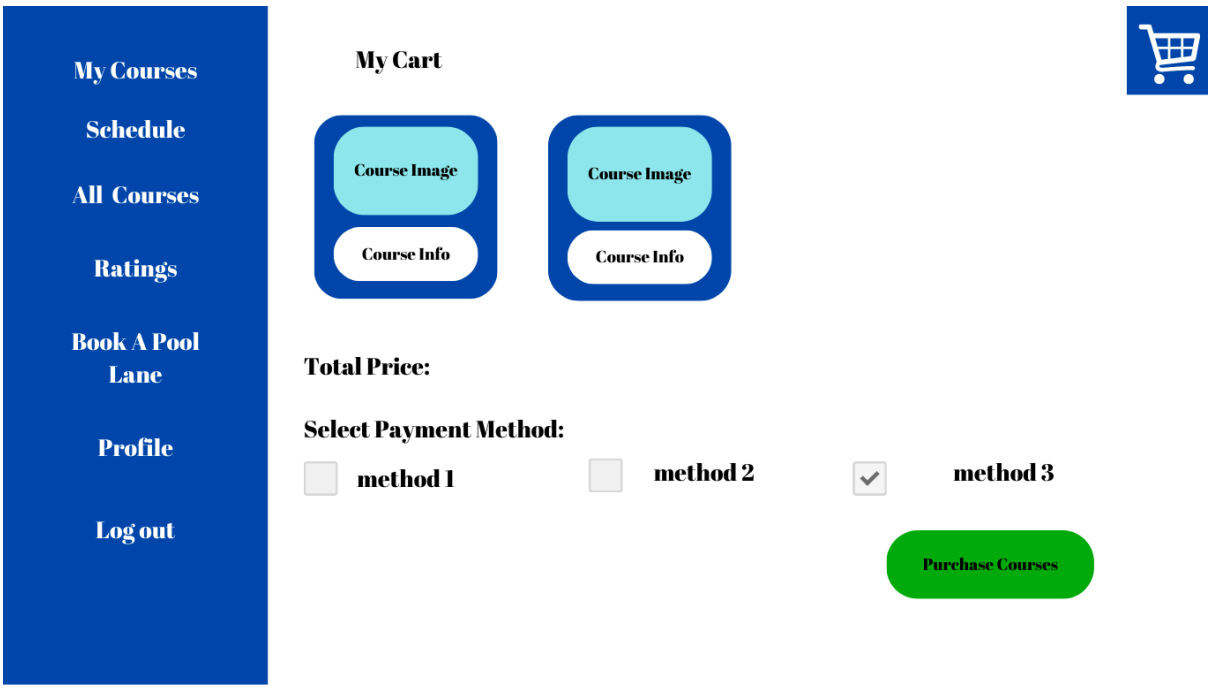
INSERT INTO users (user_id, username forename, surname, password, account_money)
VALUES (ID, 'John1', 'John', 'Doe', 'password', 100.00);

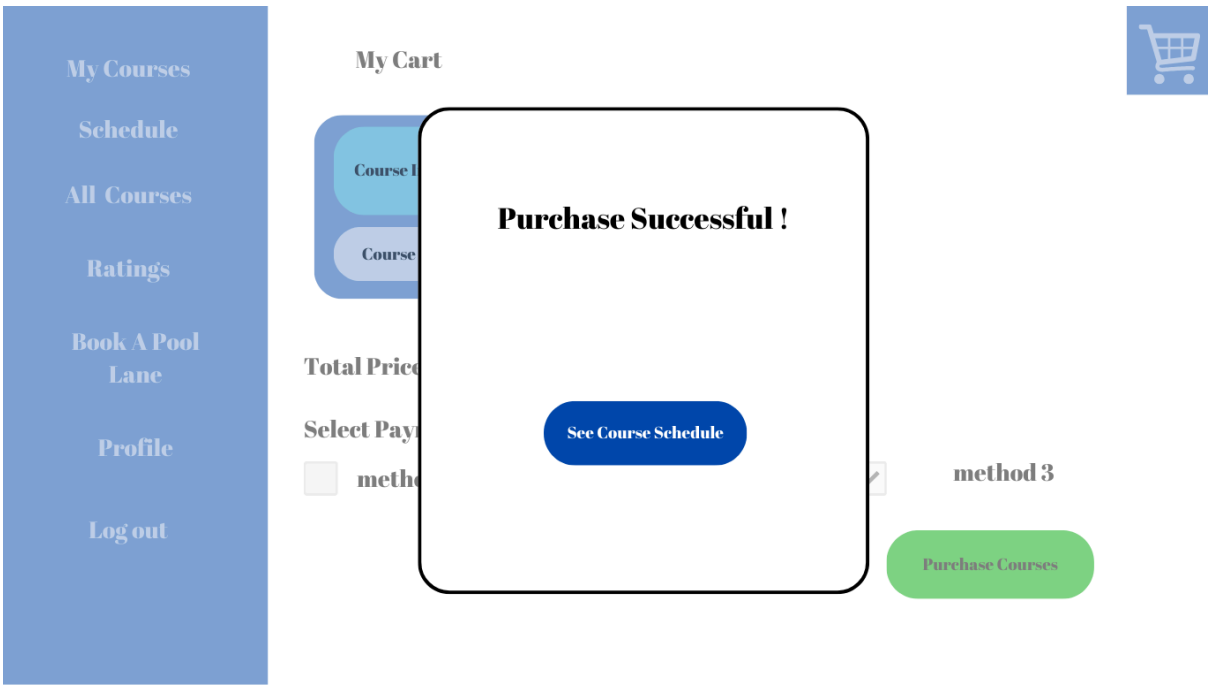
5.3 Signing Up For A Class Pages:




My Courses
Schedule
All Courses
Ratings
Book A Pool Lane
Profile
Log out
Filters
Filter 1
Filter 2
Filter 3
Filter 4
Filter 5
Apply Filters
enroll
enroll
enroll
enroll
enroll


2
My Courses
Schedule
All Courses
Ratings
Book A Pool Lane
Profile
Log out
Search
Filters: Filter2, Filter4, Filter5
Course 1, course instructor: instructor1 , capacity: 20
selected
Course 2, course instructor: instructor1 , capacity: 25
enroll
Course 3, course instructor: instructor2, capacity: 10
selected


My Courses
Schedule
All Courses
Ratings
Book A Pool Lane
Profile
Log out
My Cart
Course Image
Course Info
Course Image
Course Info
Total Price:
Select Payment Method:
method 1
method 2
method 3
Purchase Courses


My Courses
Schedule
All Courses
Ratings
Book A Pool Lane
Profile
Log out
My Cart
Course I
Course
Total Price
Select Pay
meth
Purchase Successful !
See Course Schedule
method 3
Purchase Courses


My Courses
Schedule
All Courses
Ratings
Book A Pool Lane
Profile
Log out
Month Year
Sun Mon Tue Wed Thu Fri Sat
29 30 31 01 02 03 04
05 06 07 08 09 10 11
course 1
12 13 14 15 16 17 18
course 2
19 20 21 22 23 24 25
26 27 28 29 30 31 01

Filter Classes

-No filter (Show All Classes)
C_has_S is the relationship between course and course_schedule
C_has_S(course_id,course_schedule_id)

```
SELECT *
FROM swimming_lesson S, course_schedule C, C_has_S H
WHERE S.course_id = H.course_id
AND C.course_schedule_id =  H.course_schedule_id
```

-Date

```
SELECT *
FROM swimming_lesson as S, course schedule  as C, C_has_S  as H
WHERE S.course_id = H.course_id
AND C.course_schedule_id =  H.course_schedule_id
AND S.is_full = False AND start_date >= filter_date
```

-Time

```
SELECT *
FROM swimming_lesson as S, course_schedule  as C, C_has_S  as H
WHERE S.course_id = H.course_id
AND C.course_schedule_id =  H.course_schedule_id
AND S.is_full = False AND start_time >= filter_time
```

-Gender

```
SELECT *
FROM swimming_lesson as S, C_has_S as H
WHERE S.course_id = H.course_id
AND C.course_schedule_id =  H.course_schedule_id
AND is_full = False AND gender= filter_gender;
```

-Age

```
SELECT *
FROM swimming_lesson as S, C_has_S  as H
WHERE S.course_id = H.course_id
AND C.course_schedule_id =  H.course_schedule_id
AND is_full = False AND filter_age <= max_age;
```

-Coach

```
SELECT *
FROM swimming_lesson as S, C_has_S as H
WHERE S.course_id = H.course_id
AND C.course_schedule_id =  H.course_schedule_id
```

AND is_full = False AND coach = filter_coach;

-Pool
SELECT *
FROM swimming_lesson as S, C_has_S as H
WHERE S.course_id = H.course_id
AND C.course_schedule_id = H.course_schedule_id
AND is_full = False AND pool = filter_pool;

After Class Selection

-Class restrictions (age, capacity, etc.) are checked.

SELECT S.user_id, L.course_id
FROM swimmer as S, swimming_lesson as L
WHERE L.course_id = c_id AND L.max_age >= S.age AND L.isFull = False
AND L.gender = S.gender
→Returns the ID of user and course if it fits the restrictions so it can be accessed easily

-If the class collides with any already booked classes
U_takes_C is the relationship between Course and User
U_takes_C(user_id,course_id)

SELECT C.course_name
FROM U_takes_C as T, course as C, course_schedule as S
WHERE T.user_id = id AND T.course_id = C.course_id
AND C.course_schedule_id = S.course_schedule_id
AND ((S.start_date <= course_start AND S.start_date <= course_start)
OR (S.start_date <= course_end AND S.start_date <= course_end))
AND S.start_time <> course_start_time;
→Name of the course that is conflicting

-The system checks if the deadline for sign-up has passed.
SELECT deadline
FROM course as C,
WHERE C.course_id = course;

-Add the class into the swimmers cart
INSERT INTO cart (purchaser_id, course_id, cafe_item_id)
SELECT S.swimmer_id, C.course_id
FROM swimmer as S, course as C
WHERE C.course_id = course .id, S.swimmer_id = swimmer.id

Make the payment: The money is deducted from the swimmer's account.

```
SELECT account_money
FROM users
WHERE user_id = id
```
→ Pull the money first to check if its enough

```
UPDATE users
SET account_money = account_money - ( SELECT price
                                      FROM course
                                      WHERE course.course_id = course_id)
WHERE users.user_id = id;
```

After payment, the courses are removed from the cart

```
DELETE FROM cart (purchaser_id, course_id, cafe_item_id)
WHERE (
SELECT S.swimmer_id, C.course_id
FROM  swimmer as S, course as C
WHERE C.course_id = course_id, S.swimmer_id = swimmer_id);
```

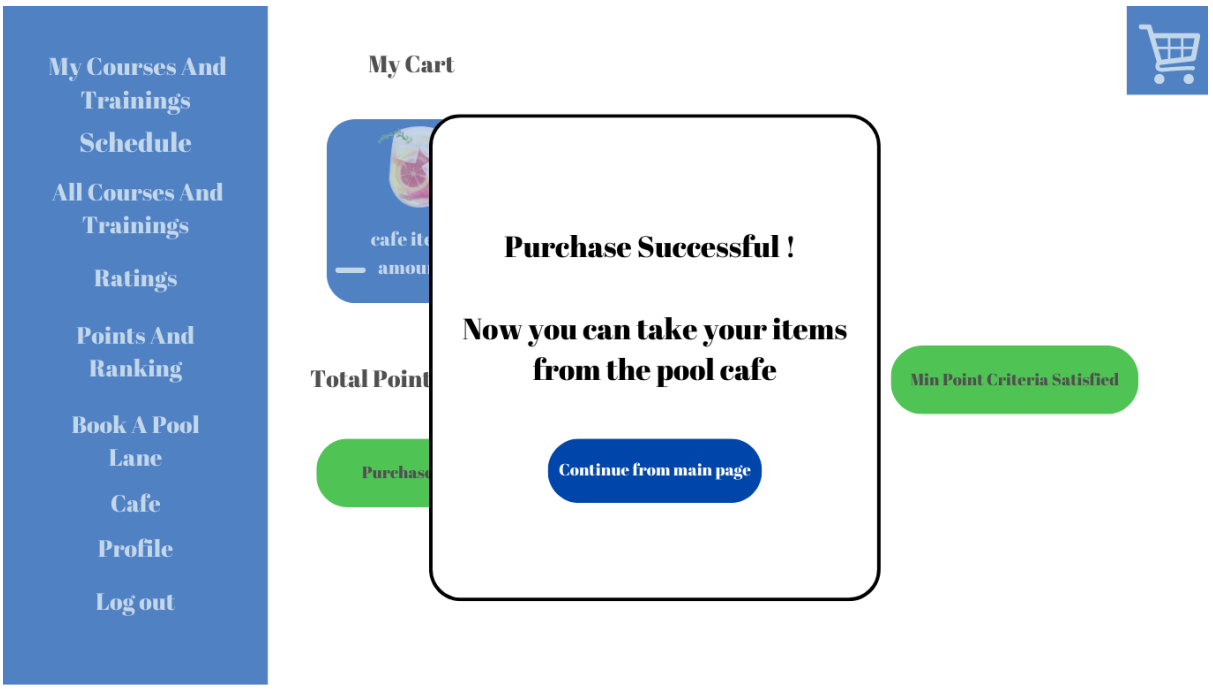
The class is added to the swimmer's calendar

```
INSERT INTO course_schedule(user_id,course_id) VALUES (course_id,
swimmer_id,coach_id, start_date, end_date, start_time, end_time, day);
```

6. Additional Functional Requirements

In addition to the expected features like book a swimming lesson or administrative features, we added two extra features to our application. Firstly we added the Cafe feature so that if the swimmers are subscribed to a monthly membership to the swimming pool they can gain points by joining classes or taking personal training. Then they can spend their points to purchase items from the cafeteria. In addition to this feature we created a second extra feature which enables all the swimmers to book time slots and lanes for their personal usage other than taking swimming lessons. Both of these features UI design and SQL queries are given below.

6.1 Cafe Page and Item Purchasing With Points:



-cafe_item is added to the cart

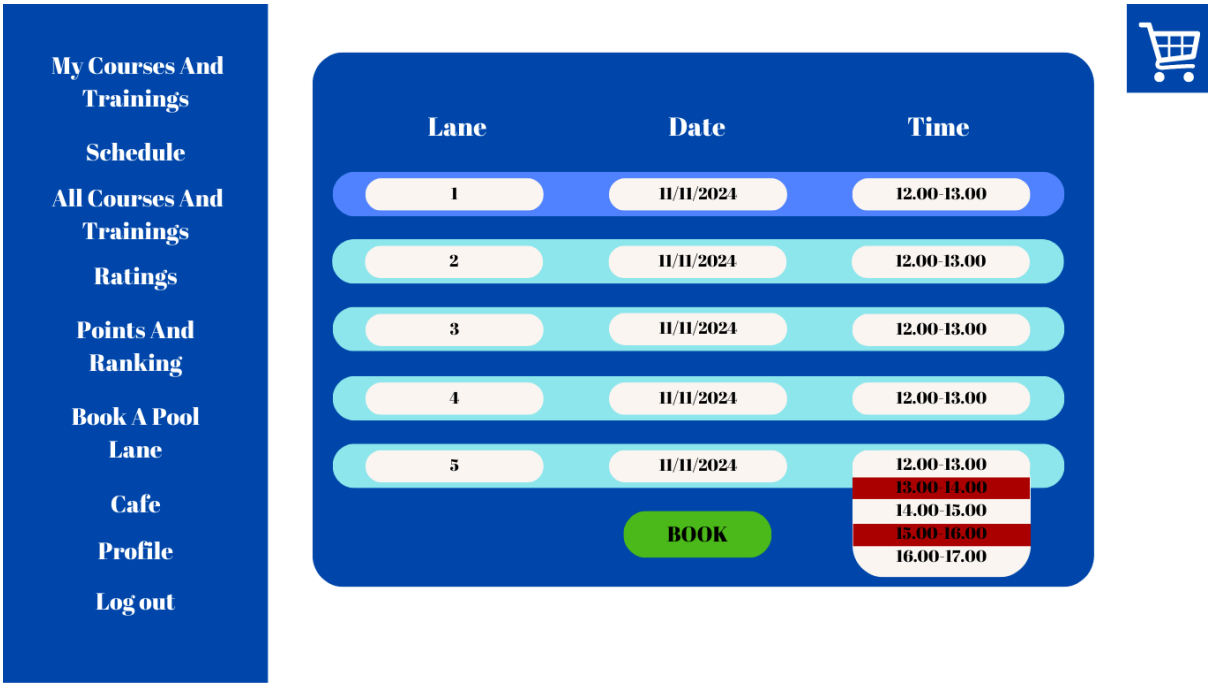
INSERT INTO cart (purchaser_id, course_id, cafe_item_id)
SELECT S.swimmer_id, I.cafe_item_id
FROM swimmer as S, cafe_item as I
WHERE I.cafe_item_id = item_id, S.swimmer_id = swimmer_id

SELECT points FROM member_swimmer WHERE swimmer_id = id
→ Pull the money first to check if its enough

UPDATE member_swimmer
SET points = points - (
SELECT (price * item_count)
FROM cafe_item as I
WHERE I.cafe_item_id = item_id)
WHERE member_swimmer.swimmer_id = id;

DELETE FROM cart (purchaser_id, course_id, cafe_item_id)
WHERE (
SELECT S.swimmer_id, I.cafe_item_id
FROM swimmer as S, cafe_item as I
WHERE I.cafe_item_id = item_id, S.swimmer_id = swimmer_id);

6.2 Book A Pool Lane:



L_books_T is the relationship between time slot and lane
L_books_T(time_slot_id,lane_id)

SELECT *
FROM time_slot as T, lane as L, L_books_T as B
WHERE B.time_slot_id = T.time_slot_id
AND B.lane_id = L.lane_id
AND L.day = input_day AND L.start_time = input_start
→Check if time slot + lane combination is available

7. Technologies And Project Implementation Plan

For the implementation, we will mainly use Python Django for the backend and Postgre SQL for the database part of our project since it can be easily configured with Django and can handle large chunks of data, making our application more scalable. We will handle all database operations by directly executing SQL queries with management commands or Django cursor.execute (SQL query) functionality in Django views. We won't use Django ORM. For the frontend part, we will use Django templates with JavaScript Ajax requests and Bootstrap templates for a more concise user interface.

8. References

[1] Canva, “Canva: Herkes için Görsel Çalışma Seti,” *Canva*. https://www.canva.com/tr_tr/

[2] “Draw.io - free flowchart maker and diagrams online,” Flowchart Maker & Online Diagram Software, https://app.diagrams.net/#G19T17ddcRAOtlJUwzh8Hsv1P1V_jbrDnp#%7B%22pageId%22%3A%22R2lEEEUBdFMjLlhIrx00%22%7D (accessed Nov. 12, 2024).