“Sports Complex Management System” Database Project Report

Anar Nyambayar, Doniyor Erkinov, and Sean Allen Siegfreid Bugarin

Duke Kunshan University
Kunshan, Jiangsu, China
{anar.nyambayar, doniyor.erkinov, seanallensiegfreid.bugarin}@dukekunshan.edu.cn

Abstract. This project report describes the “Sports Complex Management System,” a new proof-of-concept database-driven web app designed to improve the efficiency and accuracy of operations at recreational facilities and sports centers. The system addresses inefficiencies and human error in inventory management, space reservations, and membership handling. Using a relational database management system (RDBMS) with a Flask-based web interface, the project implements a mouse-centric design to optimize user interactions.
The database structure encompasses a wide range of operational data, including users, equipment, penalties, facilities, classes, and notifications. Triggers and stored procedures are employed to automate key processes, such as overdue equipment reminders, penalties, and complex facility reservation logic. The interface leverages a combination of radio buttons and drop-down menus to enhance usability, speed, and accuracy, validated through testing.
This system also lays the groundwork for further user interface development for high-traffic customer service fields, leveraging the benefits of an RDBMS in user search and selection using buttons instead of keyword search.

Keywords: Inventory Management System · Resource Management · Administration Software.

1 Introduction

Recreational centers such as the Duke Kunshan University (DKU) Sports Complex often rely on one unified system to manage inventory, space reservations, and memberships. The target clients for this type of system are (recreational) facility administrators, with the users being customer service workers/associates positioned at the front desk, often manually typing equipment or facility names in a drawn-out process to search and select.

Although these systems have the advantages that come with being a unified system for all business operations, they (and other customer service providers) exhibit interface designs that slow down essential operations, leading to labored operations and customer queues.

This project aims to use a relational database management system (RDBMS) to speed up user-system interactions by utilizing the entity-relationship (ER) model to design a lightweight point-and-click user interface and automate business logic on the database side to reduce the chances of human error and speed up interactions considerably.

This project proposes and tests a theory based on the current literature on form interface design [1] that indicates that users are generally faster at clicking than they are at typing their desired data. The theory is that necessitating every interaction be done via the mouse may greatly improve the speed at which the operator can make selections, namely through the combination of radio buttons and drop-down menu interfaces.

2 Problem

Current in-use systems, such as DSE Rec (shown in Figure 1), require a combination of mouse and keyboard inputs to allow the user to complete tasks such as inspecting the equipment. This forces the operator to repeatedly grab and let go of the mouse, worsening efficiency.

The mouse and keyboard combo approach becomes more problematic when the operator must also get up and move equipment every time a customer wants to borrow or return an item.

Another, arguably slightly larger, issue with the combo approach is that some operators may not know how a specific item is named in the database, requiring them to try typing another name or scrolling through a long list of items. Storing aliases for equipment names may not be an ideal solution in cases where the person populating the database does not know every name or aliases that an item may have.

Furthermore, ID card readers often act like a keyboard to interface with the system, which brings the issue of unprompted customers tapping their cards and interfering with the search keywords that the operator may be typing.

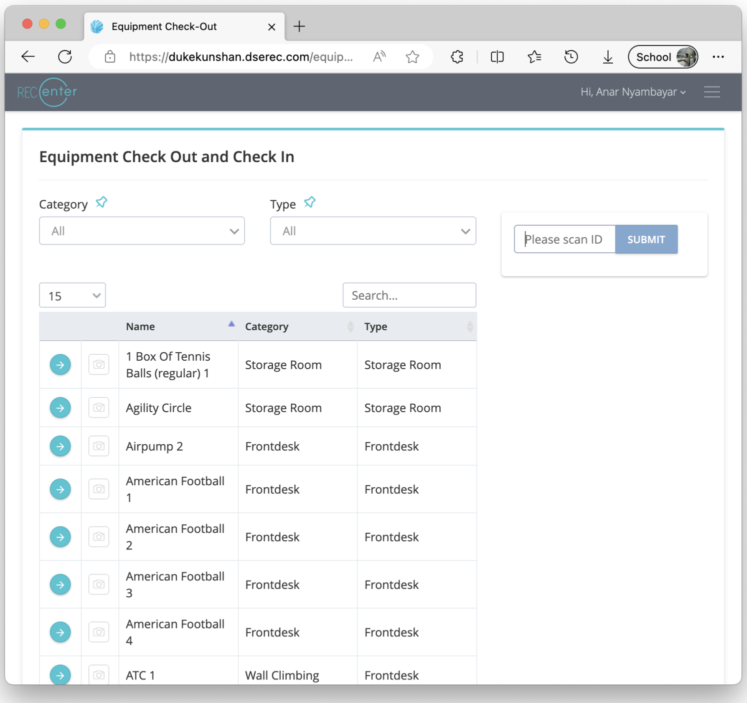
These are only the most common issues that current interfaces possess. Apart from the interface, the underlying database does not automate certain business logic on the database side, leaving operators having to manually oversee and approve or reject actions that could have been done without the risk of human error.

This project seeks to demonstrate that these problems and more can be solved through the strategic use of a simple relational database for interface design.

3 Method / System

3.1 Technologies Used

User–database interface and task automation were the key areas to develop for this project. This is why we used Flask, a lightweight Python web framework to connect to a MySQL database encompassing all operational data necessary to



Equipment Check Out and Check In

Category: All

Type: All

Please scan ID SUBMIT

15

Search...

Name	Category	Type
1 Box Of Tennis Balls (regular) 1	Storage Room	Storage Room
Agility Circle	Storage Room	Storage Room
Airpump 2	Frontdesk	Frontdesk
American Football 1	Frontdesk	Frontdesk
American Football 2	Frontdesk	Frontdesk
American Football 3	Frontdesk	Frontdesk
American Football 4	Frontdesk	Frontdesk
ATC 1	Wall Climbing	Frontdesk

Fig. 1. DSE Rec equipment check out/in screen for the DKU Sports Complex

create a streamlined app built for quick, effective edits and testing. We named the app “Sports Complex Management System”.

Due to the way card reader devices work, achieving a mouse-only selection interface is only partially possible since the reader must input data to an HTML text box. Thus, this necessary text box will be disregarded in discussions about mouse-only user interface.

3.2 Database Design

To design a realistic database for this project, all design decisions aimed to follow the business logic outlined by the Sports Complex Operation Manual provided to us by the Sports Operation team of the DKU Department of Athletics.

We defined and isolated only the necessary components that make up a recreation center database and created a database design document to provide our development an outline to follow.

Figure 2 shows the entity-relationship diagram (ERD) we created for the database design document. At the center is the `user` entity, as it participates in nearly every functionality/relationship. An original addition to the database is

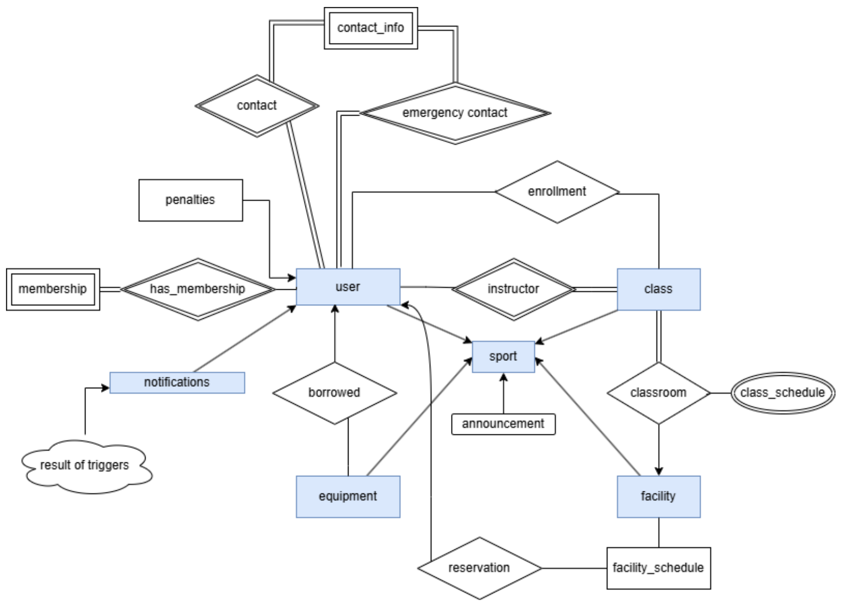


Fig. 2. ER Diagram of the 'sports' database

the `sports` entity, which serves to categorize every equipment, class, and facility as well as announcements, which connect to the `user` entity via subscriptions, as shown in the relational schema visualization in figure 3.

Our database is designed to manage the information for various users (students, sports-complex staff, workers, faculty, etc.), different types of sports offered at the DKU Sports Complex, sports classes along with their corresponding schedules, sports equipment, and announcements to users about various events that take place at DKU.

1. `user`: stores information about the user, including personal details such as the user's netID, card number, and role.

```
CREATE TABLE
    user(
        user_id     INT PRIMARY KEY AUTO_INCREMENT,
        netid       VARCHAR(10) UNIQUE,
        card_number VARCHAR(20) UNIQUE,
        first_name  VARCHAR(50) NOT NULL,
        last_name   VARCHAR(50),
        gender      ENUM('male', 'female', 'other',
        'undisclosed'),
        birthday    DATE,
        role        ENUM(
            'student',
            'faculty',
```

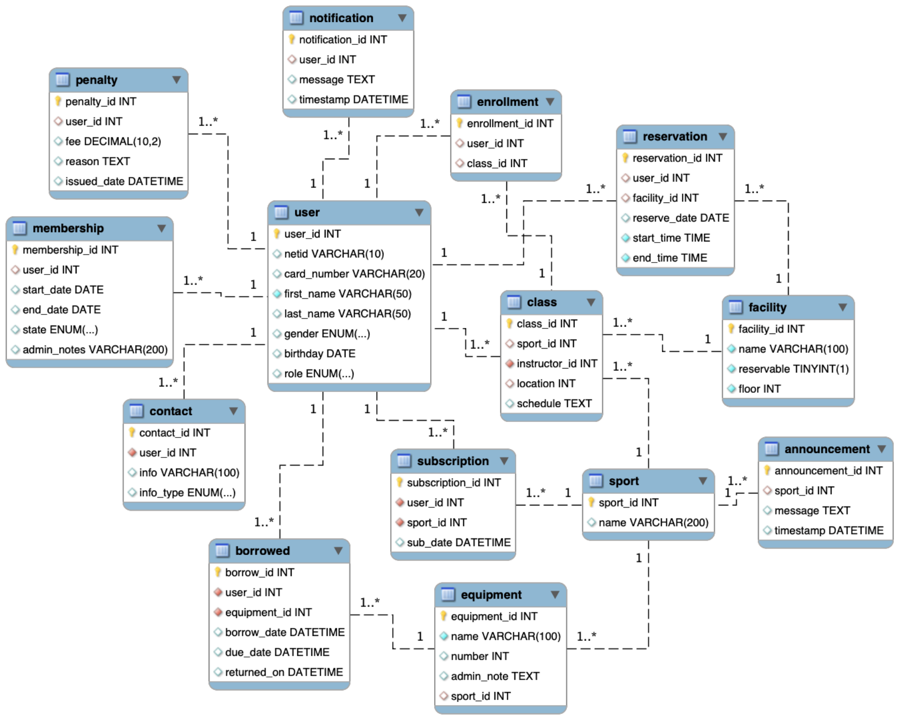


Fig. 3. Visualization of the `sports` database relational schema

```
                'staff',
                'student_worker',
                'affiliate',
                'guest',
            )
        );
```

2. `sport`: contains data about different sports offered at DKU.
```
CREATE TABLE
    sport (
        sport_id    INT PRIMARY KEY AUTO_INCREMENT,
        name        VARCHAR(200)
    );
```

3. `equipment`: to manage the equipment at the Sports Complex, including equipment names, and the equipment number available for a user to borrow.
```
CREATE TABLE
    equipment (
        equipment_id    INT PRIMARY KEY AUTO_INCREMENT,
        name            VARCHAR(100) NOT NULL,
        number          INT CHECK (number >= 0),
        DEFAULT         'available',
        admin_note      TEXT,
        sport_id        INT,
        FOREIGN KEY (sport_id) REFERENCES sport (sport_id)
    );
```

4. `facility`: this table manages information about the sports facilities at DKU, such as tennis courts, football stadiums, etc., where various classes can be held and the facilities can be reserved by users.
```
CREATE TABLE
    facility (
        facility_id INT PRIMARY KEY AUTO_INCREMENT,
        name        VARCHAR(100) NOT NULL,
        reservable  BOOLEAN NOT NULL,
        floor       INT NOT NULL,
        CONSTRAINT  unique_facility UNIQUE (name, floor)
    );
```

5. `class`: this table serves to store information about the classes offered at DKU within the scope of the sports complex.
```
CREATE TABLE
    class (
        class_id        INT PRIMARY KEY AUTO_INCREMENT,
        sport_id        INT,
        instructor_id   INT NOT NULL,
        location        INT,
        FOREIGN KEY (sport_id) REFERENCES sport (sport_id),
        FOREIGN KEY (instructor_id) REFERENCES user (user_id),
        FOREIGN KEY (location) REFERENCES facility (facility_id),
        CONSTRAINT unique_class UNIQUE (sport_id, instructor_id,
        location)
    );
```

6. `notifications`: the notifications for each user are stored in this table. Those notifications serve as a reminder when the user borrows equipment and forgets to return the item, or when there is an announcement for a particular sport to receive the announcements of which the user has subscribed.

```
CREATE TABLE
    notification (
        notification_id INT PRIMARY KEY AUTO_INCREMENT,
        user_id         INT,
        message         TEXT,
        timestamp       DATETIME DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (user_id) REFERENCES user(user_id)
    );
```

To automate tasks like sending reminders to users who have borrowed equipment from the Sports Complex but have not returned it on time, to update the user on new announcements made for the sports they are subscribed to, or to implement penalties on the users who have not returned the borrowed items, we decided to design triggers for this task. Our triggers for the Sports Complex project include:

1. `send_reminder_to_return`: This trigger is executed right after a new value is recorded into the ’borrowed’ table. The trigger checks if an item in the ’borrowed’ table has not been returned and if the borrowed equipment has not been returned yet with its return deadline approaching within an hour from the current time. If these conditions are met, a notification reminding about returning the equipment is sent to a user to help the user avoid a penalty.

```
DELIMITER $$
CREATE TRIGGER send_reminder_to_return
    AFTER INSERT ON borrowed
    FOR EACH ROW
    BEGIN
        IF NEW.returned_on IS NULL
        AND NEW.due_when > DATE_SUB(NOW(), INTERVAL 1 HOUR) THEN
            INSERT INTO notification(user_id, message, timestamp)
            SELECT
```

```
                    NEW.user_id,
                    CONCAT('Dear ', user.first_name, ' ',
                    user.last_name, ', please return the following item: ',
                    equipment.name, ' within an hour; otherwise,
                    you will be issued a 25RMB penalty. Thank you!'),
                    NOW()
            FROM user
            JOIN borrowed USING(user_id)
            JOIN equipment USING (equipment_id)
            WHERE user.user_id= NEW.user_id;
        END IF;
    END $$
DELIMITER ;
```

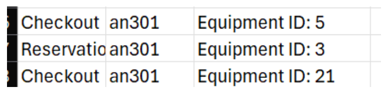
2. `apply_penalty_overdue_equipment`: the following trigger is executed after a new record is added to the 'borrowed' table. The trigger checks if borrowed equipment has not been returned and if the due date has passed; providing that the equipment has not been returned and the due date has passed, the trigger helps to automatically insert a penalty record into the 'penalty' table, applying a 25 RMB fine to the user with an overdue item.

```
DELIMITER $$

CREATE TRIGGER apply_penalty_overdue_equipment
    AFTER INSERT ON borrowed
    FOR EACH ROW
    BEGIN
    IF NEW.returned_on IS NULL AND NEW.due_when < NOW() THEN
            INSERT INTO penalty (user_id, fee, equipment_id, issued_date)
            VALUES (
                NEW.user_id,
                25.00, -- Penalty fee amount
                 NEW.equipment_id,
                NOW()
            );
        END IF;
    END$$

DELIMITER ;
```

3. `penalty_notification`: once a new record for the 'penalty' table is created, the trigger sends a notification to the user who has been fined, informing the user of the 25 RMB penalty for failing to return an item by specified due time (usually the next day 9 PM). The users who were penalized for not being able to return an item by the due date are not allowed to borrow new equipment unless they return the borrowed item to the Sports Complex and pay the fine fee.

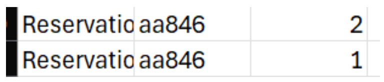
```
DELIMITER $$
CREATE TRIGGER penalty_notification
    AFTER INSERT ON penalty
    FOR EACH ROW
    BEGIN
        INSERT INTO notification(user_id, message, timestamp)
        SELECT
                NEW.user_id,
                CONCAT('Dear ', user.first_name, ' ', user.last_name,
                ' since you have not returned the item: ', equipment.name,
                       ' before the due time which was at ',
                       borrowed.due_when, ','
                       'you will now be fined with 25 RMB penalty'),
                NOW()
        FROM user
        JOIN borrowed USING(user_id)
        JOIN equipment USING (equipment_id)
        WHERE user.user_id= NEW.user_id;
    END $$

DELIMITER ;
```

The project also incorporates an `activity_log.csv` file to record data for each check-out and check-in equipment transaction, along with reservation information at the DKU Sports Complex. Each entry includes the entry 'checkout', 'checkin', and 'reservation' to denote the type of log, as well as, the entries about the netid (user's school-based ID) of the user and the id of equipment that has been borrowed or the id of the facility that has been booked.



Checkout	an301	Equipment ID: 5
Reservatio	an301	Equipment ID: 3
Checkout	an301	Equipment ID: 21

Fig. 4. Screenshot of checkout entries in the log file



Reservatio	aa846	2
Reservatio	aa846	1

Fig. 5. Screenshot of reservation entries in the log file

To more effectively manage key operations within the Sports Complex, several procedures were developed to help users perform essential tasks more efficiently. Those procedures include:

1. `checkout_equipment`: this procedure manages the checking out process for equipment in the Sports Complex. To do this, it takes two input parameters: 'user_id' and 'equipment_id'. In the procedure, it first checks if the current time is outside the operational hours, which in the case of Sports Complex

is from 7:00 AM to 9:00 PM. Following that, the procedure checks if the requested item for borrowing has not been borrowed already, or if the user has an outstanding penalty preventing them from borrowing any equipment. Last, but not least, only active members of the Sports Complex are allowed to check out equipment to borrow them.

```
DELIMITER //

CREATE PROCEDURE checkout_equipment(
    IN p_user_id INT,
    IN p_equipment_id INT
)
BEGIN
    DECLARE existing_checkouts INT;

    SELECT COUNT(*) INTO existing_checkouts
    FROM borrowed
    WHERE equipment_id = p_equipment_id AND returned_on IS NULL;

    IF CURTIME() > '21:00:00' OR CURTIME() < '07:00:00' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'The Sports Complex is closed for
        the day, come back tomorrow after 7am!';
    ELSEIF existing_checkouts > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Equipment unavailable.';
    ELSEIF p_user_id IN (SELECT user_id FROM penalty) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'User has a penalty and can not
        check out equipment until fees are paid.';
    ELSEIF p_user_id NOT IN (SELECT user_id FROM
    active_members) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'User has no active membership.';
    ELSE
        INSERT INTO borrowed (user_id, equipment_id)
        VALUES (p_user_id, p_equipment_id);
    END IF;
END //

DELIMITER ;
```

2. `reserve_facility`: the following procedure handles the reservation of sports facilities. Before creating a reservation for users, the procedure checks for potential conflicts with existing reservations for the indicated facility and date by the user. The procedure also makes sure the reservation is possible only between working hours of the Sports Complex (7:00 AM to 9:00 PM)

and does not exceed an hour. In addition, the procedure facilitates that only users with an active membership can make a reservation.

```
DELIMITER //

CREATE PROCEDURE reserve_facility(
    IN p_user_id INT,
    IN p_facility_id INT,
    IN p_reserve_date DATE,
    IN p_start_time TIME,
    IN p_end_time TIME
)
BEGIN
    DECLARE conflict_count INT;
    DECLARE reservation_duration INT;

    -- Calculating the reservation duration in minutes
    SET reservation_duration = TIME_TO_SEC(p_end_time) / 60 -
    TIME_TO_SEC(p_start_time) / 60;

    -- Checking for time conflicts
    SELECT COUNT(*) INTO conflict_count
    FROM reservation
    WHERE facility_id = p_facility_id
      AND reserve_date = p_reserve_date
      AND (
        (p_start_time >= start_time AND p_start_time <
        end_time) OR
        (p_end_time > start_time AND p_end_time <= end_time) OR
        (p_start_time <= start_time AND p_end_time >= end_time)
      );

    IF CURTIME() > '21:00:00' OR CURTIME() < '07:00:00' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'The Sports Complex is closed for
        the day, come back tomorrow after 7am!';
    ELSEIF p_user_id IN (SELECT user_id FROM penalty) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'User has a penalty and can not make
        reservations until fees are paid.';
    ELSEIF p_user_id NOT IN (SELECT user_id FROM
    active_members) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'User has no active membership.';
    ELSEIF conflict_count > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Reservation time conflict
        detected.';
    ELSEIF reservation_duration > 60 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Reservation duration exceeds one hour.';
    ELSEIF p_reserve_date < CURRENT_DATE() THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT  = 'Please select a date in the future.';
    ELSEIF DATE(p_reserve_date) = CURDATE() AND
    TIME(p_start_time) < CURTIME() THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT  = 'Please select a date in the future.';
    ELSE
        INSERT INTO reservation (user_id, facility_id, reserve_date, start_time, end_tim
        VALUES (p_user_id, p_facility_id, p_reserve_date, p_start_time, p_end_time);
    END IF;
END //

DELIMITER ;
```

3. `register_user`: the procedure is designed to register a new user into the Sports Complex system. The procedure works by first checking if the parameters such as 'netid', and 'card_number' are unique, the birthday is valid, namely not indicated in the future, and other fields such as name, gender, role, and contact information are properly provided. Provided all the validation checks pass, the procedure inserts the user's details into the 'user' table and records the user's contact information in the 'contact' table.

```
DELIMITER $$

CREATE PROCEDURE register_user(
    IN p_netid VARCHAR(10),
    IN p_card_number VARCHAR(10),
    IN p_first_name VARCHAR(50),
    IN p_last_name VARCHAR(50),
    IN p_gender ENUM('male', 'female', 'other', 'undisclosed'),
    IN p_birthday DATE,
    IN p_role ENUM('student', 'faculty', 'staff', 'worker',
    'student_worker', 'affiliate', 'alumni', 'guest',
    'security', 'other'),
    IN p_info_type ENUM('phone', 'email', 'address',
    'emergency'),
    IN p_info VARCHAR(100)
)
BEGIN
    DECLARE new_user_id INT;

    IF EXISTS(SELECT * FROM user WHERE netid = p_netid) THEN
```

```
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'The user already exists. Please add
        another user.';
    ELSEIF EXISTS(SELECT * FROM user WHERE card_number =
    p_card_number) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'The user already exists. Please add
        another user.';
    ELSEIF p_birthday > CURRENT_DATE() THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'You cannot add a user whose
        birthday is in the future! Please change the birthday value';
    ELSEIF p_first_name IS NULL OR TRIM(p_first_name) = '' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'First name value cannot be empty';
    ELSEIF p_gender NOT IN ('male', 'female', 'other', 'undisclosed') THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid gender specified';
    ELSEIF p_role NOT IN ('student', 'faculty', 'staff',
    'worker', 'student_worker', 'affiliate', 'alumni', 'guest',
    'security', 'other') THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid role specified';
    ELSEIF p_info_type NOT IN ('phone', 'email', 'address', 'emergency') THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid contact information type';
    ELSE
        INSERT INTO user(netid, card_number, first_name, last_name, gender, birthday, ro
        VALUES (p_netid, p_card_number, p_first_name, p_last_name, p_gender, p_birthday,

        SET new_user_id = LAST_INSERT_ID();
        INSERT INTO contact( user_id, info_type, info)
        VALUES( new_user_id,p_info_type, p_info);
    END IF;
END $$

DELIMITER ;
```

4. `enroll_users_to_classes`: This procedure works to enroll users in classes while ensuring there are no conflicts in the users' schedules. It does so, by retrieving the start and end times of the class from the 'schedule' table, then using this information it checks if the user is already in the class or if there is any time conflict with other enrolled classes. If both of these conditions are satisfied, the user is successfully enrolled in the class.
   `DELIMITER $$`

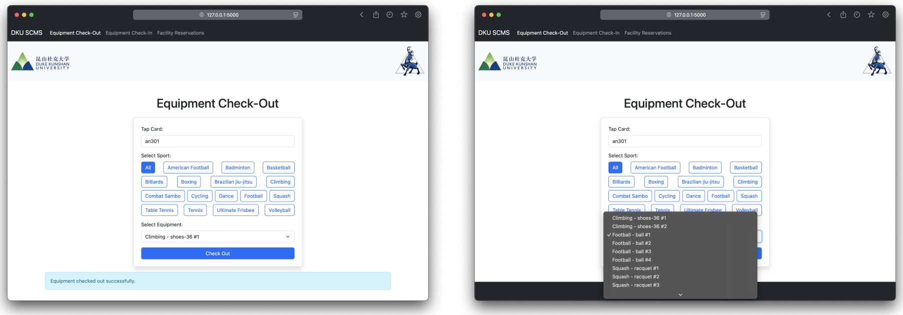
```
CREATE PROCEDURE enroll_users_to_classes(
    IN p_user_id INT,
    IN p_class_id INT
)
BEGIN
    DECLARE conflict_count INT;
    DECLARE class_start_time DATETIME;
    DECLARE class_end_time DATETIME;

    -- deriving start and end times of classes from the schedule
    SELECT start_time, end_time INTO class_start_time, class_end_time
    FROM schedule
    WHERE class_id= p_class_id
    LIMIT 1;

    -- Checking for time conflicts
    SELECT COUNT(*) INTO conflict_count
    FROM enrollment
    JOIN schedule USING (class_id)
    WHERE enrollment.user_id= p_user_id AND
        (
            (class_start_time >= schedule.start_time AND
            class_start_time < schedule.end_time) OR
            (class_end_time > schedule.start_time AND
            class_end_time <= schedule.end_time) OR
            (class_start_time <= schedule.start_time AND
            class_end_time >= schedule.end_time)
        );

    IF EXISTS(SELECT * FROM enrollment WHERE user_id= p_user_id AND class_id= p_class_id
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'The user has already enrolled in this class';
    ELSEIF conflict_count > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Class schedule time conflict detected.';
    ELSE
        INSERT INTO enrollment(user_id, class_id)
        VALUES (p_user_id, p_class_id);
    END IF;
END $$

DELIMITER ;
```

4 Computational Results / Evaluation

The project was hosted on GitHub[1], which allowed for effective team communication and collaborative programming. All necessary files required to download and run the project on any machine are located in the GitHub repository, as well as detailed instructions in the README.md file for setting up the project to run. The instructions assume that the user has a MySQL server installed and running on their system.

During development, radio buttons in the shape of rectangle buttons with names were found to be the most convenient to locate and use. Bootstrap was used to try out and test various different types of radio buttons.

Figures 6 and 7 are screenshots of the project running on a macOS machine, with the Safari browser. Tests have also been conducted on market-leading Chromium-based browsers, and no discernible user experience differences were shown apart from slightly different date selectors.



Equipment Check-Out

Tap Card: an301

Select Equipment: Climbing - shoes-36 #1

Equipment checked out successfully.

Equipment Check-Out

Tap Card: an301

Climbing - shoes-36 #1
Climbing - shoes-36 #2
✓ Football - ball #1
Football - ball #2
Football - ball #3
Football - ball #4
Squash - racquet #1
Squash - racquet #2
Squash - racquet #3

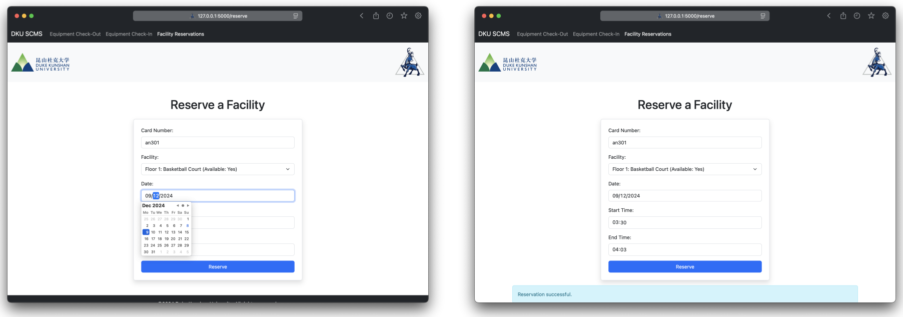
Fig. 6. Successful Checkout

5 Conclusion

After initial debugging runs, the user interface was successfully modified to accommodate mouse-only selections as set out in the beginning, apart from the 'Tap Card' text box, which is actually the input interface for the card reader device, as explained in the method/system section.

Since two of the three authors have experience using this type of software, we were able to conclude that the final user experience is better than had they used keyboard input.

[1] Project repository: github.com/pomegranar/cs310-Group-Project



Reserve a Facility

Card Number: an301

Facility: Floor 1: Basketball Court (Available: Yes)

Date: 09/12/2024

Reserve a Facility

Card Number: an301

Facility: Floor 1: Basketball Court (Available: Yes)

Date: 09/12/2024

Start Time: 03:30

End Time: 04:03

Reservation successful.

Fig. 7. Successful Reservation

As for business logic automation, after testing for various edge cases, it was clear that the MySQL RDBMS solution made it easy to develop and ensure reliability, using SQL triggers and procedures.

6 Appendix

6.1 Contributions

Anar Nyambayar: Provided initial conceptualization and project aims. Carried out team management tasks, including the creation of the project's GitHub repository, writing the README.md file, and setting up the project in the Flask framework. Created the core database schema as well as procedures for checking in/out equipment and facility reservations, with their corresponding HTML and JavaScript files.

Doniyor Erkinov: Contributed to designing an initial draft for the ER diagram as well as the final Entity-Relational (ER) Model and Relational Model; helped to build the relation schemas for the database; built the following triggers to maintain database automation: 'apply_penalty_overdue_equipment', 'prevent_duplicate_borrow', 'send_reminder_to_return', 'penalty_notification', 'announcement_notification'; and following procedures to provide seamless management of the Sports Complex Database design procedures: 'enroll_users_to_classes', 'create_new_announcement', 'create_new_subscription_for_user', and 'register_user'.

Sean Allen Siegfreid Bugarin: Contributed to shaping the core concepts of the System Requirements Specification (SRS) and establishing the basic structure of the ER diagram. Worked on frontend development by integrating the Bootstrap framework into the HTML files and enhancing the user interface. Additionally, developed backend routes in backend design within app.py, ensuring proper routing and URL connections between the frontend pages.

6.2 Project Timeline

A weekly project timeline should also be included, highlighting key accomplishments, obstacles encountered, and the strategies employed to overcome them.

1. `Week_1`: Project proposal. Throughout the first week, we formed a team to work on a final project and brainstormed potential ideas to discuss with a Professor. Eventually, our team chose to design the Sports Complex database and the project was approved by Professor Mustafa Misir.
2. `Week_2`: Project requirement specification. Throughout the second week, our team worked on formulating the problem of the existing DKU Sports Complex system and tried to propose a scalable solution to enhance the users' experience in the Sports Complex. Within the second week, essential system and user requirements were specified for further project development.
3. `Week_3`: Throughout week 3 we designed the draft ER diagram for our Sports Complex Database design project and discussed further project development with Professor.
4. `Week_4`: Database design documents were created. The final version of the ER diagram was designed and additional tables were added along with triggers and procedures.
5. `Week_5`: Creation of app.py and other back-end,front-end elements were tried, tested, and developed. The database was also further updated.
6. `Week_6`: Further debugging and testing was carried out to iteratively improve the project. Additional constraints to tables were added.
7. `Week_7`: Further business logic was implemented in the app and was demonstrated in a class presentation. Other features were included based on peers' responses during the presentation.

Declarations

The authors declare that they have no known competing financial interests or personal relationships that could have influenced the work reported in this paper.

References

1. Husser, J.A., Fernandez, K.E.: To Click, Type, or Drag? Evaluating Speed of Survey Data Input Methods. Survey Practice **6**(2) (jul 1 2013). `https://doi.org/10.29115/SP-2013-0008`