Bookshop Management System

CSS325 Database System

Section 2

Group 19: Maew Yeab Duangchan

Group Member

Krittidech	Paijitjinda	6422781664
Supakorn	Vannathong	6422782712
Kittisak	Wanganansukdee	6422781441

ABSTRACT

This report is made to facilitate and record the creation of the “Bookshop Management System” project as a part of “CSS325 Database System” Subject.

The project is created to be used as the bookshop digital system such as a cash register to help with the everyday interactions inside the bookshop.

To create the system, an ER Diagram and a Data Flow Diagram are created to aid the determination of functions and tables needed for the program and the database system.

The main functions of the program are payment calculation, book searching, registration and editing of membership data, and authorisation of the staff.

Also, Included in this report are the user interfaces of the program. This is to help with the planning for the creation of the program.

TABLE OF CONTENTS

INTRODUCTION

This report is made to facilitate and record the creation of the “Bookshop Management System” project as a part of “CSS325 Database System” Subject.

The project aims to build a system that can be installed into the digital system of a bookshop such as a cash register, or the main computer etc. to help manage the everyday interactions inside the shop such as making transactions, searching for books, or membership registration etc.

ER DIAGRAM

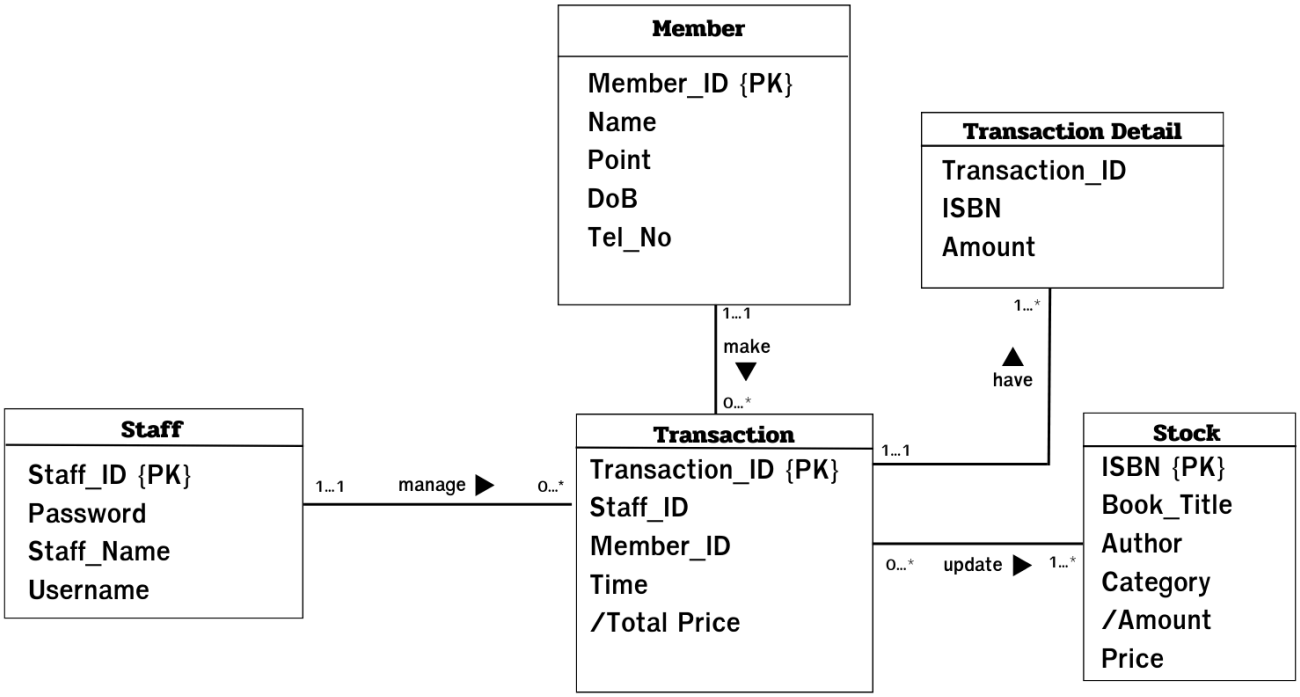


Figure 1: ER Diagram

TABLE DETAIL

1. Transaction - Store transaction records.

Attribute	Type	Null	Key	Note
Transaction_ID	VARCHAR(10)	☐	PK	Auto_Increment
Staff_ID	VARCHAR(10)	☐	FK -> Staff(Staff_ID)	
Member_ID	VARCHAR(10)	☑	FK -> Member(Member_ID)	
Transaction_time	TIMESTAMP	☐		Default: Current Timestamp
/Total	DECIMAL(6,2)	☐		

2. Stock - Store the list of books in the stock.

Attribute	Type	Null	Key	Note
ISBN	VARCHAR(13)	☐	PK	
Book_title	VARCHAR(255)	☐		
Author	VARCHAR(255)	☐		
Category	VARCHAR(255)	☐		
Price	DECIMAL(5,2)	☐		Default: 0.00
Amount	INT(11)	☐		Default: 0

3. Transaction_Detail - Store detail of each transaction.

Attribute	Type	Null	Key	Note
Transaction_ID	VARCHAR(10)	☐	PK	Auto_Increment
ISBN	VARCHAR(13)	☐	PK	
/Amount	INT(3)	☐		Default: 1

4. Staff - Keep information about the staff.

Attribute	Type	Null	Key	Note
Staff_ID	INT(11)	☐	PK	Auto_Increment
Password	VARCHAR(255)	☐		
Staff_Name	VARCHAR(255)	☐		Full Name
Username	VARCHAR(255)	☐	Unique	

5. Member - Keep information about the members.

Attribute	Type	Null	Key	Note
Member_ID	VARCHAR(11)	☐	PK	Auto_Increment
Name	VARCHAR(255)	☐		Full Name
Point	INT(11)	☐		Default: 0
DoB	DATE	☐		Date of Birth
Tel_No	VARCHAR(15)	☑	Unique	

Note: `Transaction` and `Transaction_detail` are linked in the way that one ‘transaction_id’ will have only one row in the `Transaction` table, but can have many rows in the `Transaction_detail`. In other words, one transaction can contain many books. The transaction information is kept in the `Transaction` table, and the information about each book in the transaction is kept in the `Transaction_detail` table.

DATA FLOW DIAGRAM

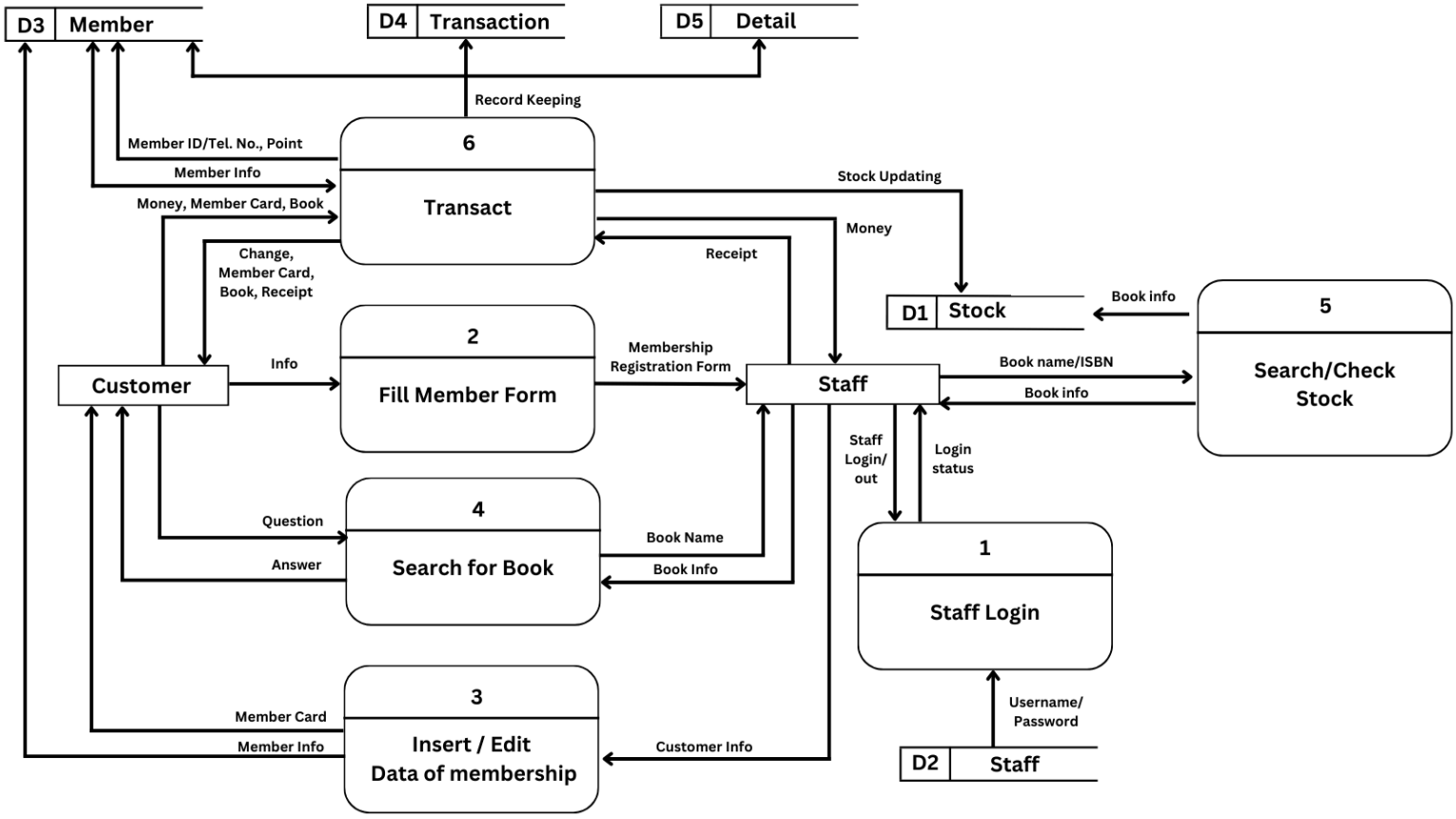


Figure 2: Data Flow Diagram

FUNCTIONAL DESIGN

User-Level Functionality

1. Transaction Making

- The cashier can fill in the book's ISBN to put them into the cart which is shown on the screen.
- The cashier can verify the membership by either filling in the membership ID, or inputting the customer's phone number.
- The subtotal and the discount is identified by the combined cost of the books and the member's point respectively, then the total cost is calculated.
- After the money has been received, the " Proceed to Check out " button can be clicked to finish the transaction.
- If the transaction is canceled, the transaction data is deleted.

2. Book Searching

- The customer can ask for the book they want, then staff/cashier will check stock
- Cashiers use the "search" function on the screen to search the book by inserting the book's name, category, book's title, author's name , and ISBN.

3. Register/Update Member

- Customers can sign-up for membership and get a discount on their purchase.
- If the member wants to change their information, the staff can update them by using the "edit member info" function.
- Need to verify the member identity before changing the data, either by phone number or membership ID.

4. Staff login

- Use staff username and password to log in to cashier counter

Application-Level Functionality

1. Update stock

- After a transaction is completed, the system will automatically update the stock by decreasing the number of sold books in the stock database.
- If the amount of a particular book becomes 0, the data will not disappear from the database.

2. Check Membership

- When membership ID is filled, it will be used to check the membership information.
- The membership point is used to determine the discount%.
- After payment, points will be added to the member info.

3. Check stock

- The book name or ISBN that is input by the cashier will be used to fetch the information about that book from the database.

4. Calculate the Price & Discount

- The transaction ID will be used to calculate the price of the transaction.
- The member point is used to determine the discount%.
- The net total is calculated using the information above.

IMPLEMENTATION OF DATABASE MANAGEMENT SYSTEM

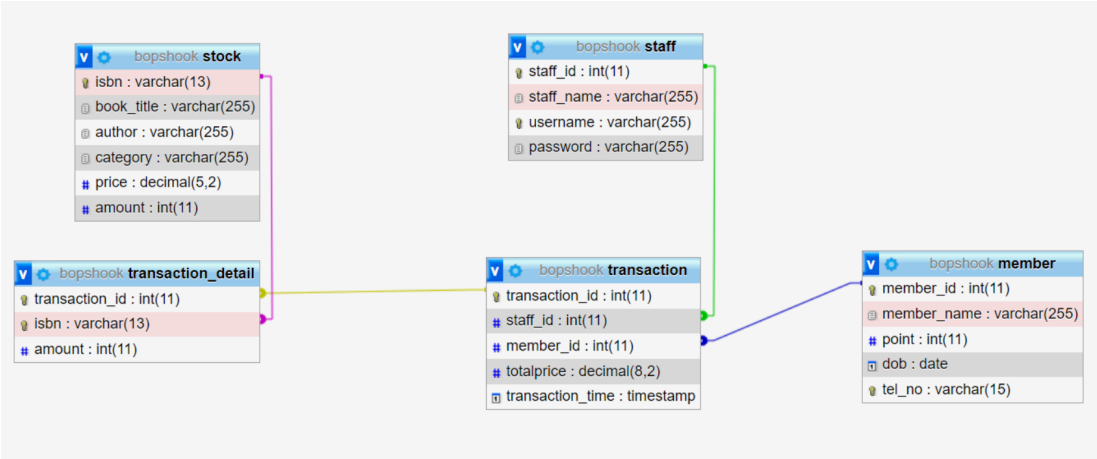


Figure 3: The Database

Our database management system uses the program “MAMP” to open the server locally and as the user interface for creating the database. Then, we use HTML and PHP language in order to create the website and link it to our database. The database, tables, relationships, and initial information in the tables are all done via the user interface function in the PHPMYADMIN via MAMP.

The relationship between tables:

1) Pink line: Parent is **`stock`.’isbn’**, Child is **`transaction_detail`.’isbn’,**
   ON UPDATE RESTRICT, ON DELETE RESTRICT
2) Yellow line: Parent is **`transaction`.’transaction_id’**, Child is
   `transaction_detail`.’transaction_id’,
   ON UPDATE RESTRICT, ON DELETE CASCADE
3) Green line: Parent is **`stock`.’staff_id’**, Child is **`transaction_detail`.’staff_id’,**
   ON UPDATE RESTRICT, ON DELETE RESTRICT
4) Blue line: Parent is **`stock`.’member_id’**, Child is
   `transaction_detail`.’member_id’,
   ON UPDATE RESTRICT, ON DELETE RESTRICT

FUNCTIONS, USER INTERFACE, AND THEIR RELATIONSHIP

In our database, we have:

- 2 Functions
    - **cal_price**: Calculate the total price of books in a transaction.

```
DELEMITER$$
CREATE FUNCTION `cal_price` (IN `id` INT) RETURNS DECIMAL(8,2)
    BEGIN
    DECLARE total DECIMAL(8,2) DEFAULT 0;
    DECLARE endloop INT DEFAULT 0;
    DECLARE amnt INT;
    DECLARE pr DECIMAL(8,2);
    DECLARE bookid VARCHAR(13);
    DECLARE sel_detail CURSOR FOR SELECT isbn, amount FROM transaction_detail WHERE transaction_id = id;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET endloop = 1;
    OPEN sel_detail;
        WHILE endloop = 0 DO
        FETCH sel_detail INTO bookid, amnt;
            If endloop = 0 THEN
                SET pr = (SELECT price FROM stock WHERE stock.isbn = bookid);
                SET total = total + (pr*amnt);
            END IF;
        END WHILE;
    CLOSE sel_detail;
RETURN (total);
END$$
DELIMETER ;
```

Figure 4: **cal_price** Function

This function works by using the cursor to fetch the ISBN and the amount of the books that are in the **`transaction_detail`** table by using the 'transaction_id' one by one, and use that information to calculate and update the total price, until all of the rows in the **`transaction_detail`** table with the same transaction_id are fetched and calculated.

<u>Detailed Explanation</u>: This function accepts one input 'id' and will return one decimal value. These variables are declared: 'total' to keep the total price, 'endloop' for loop condition, 'amnt' for keeping the 'amount' of the book, 'pr' for keeping the 'price' of the book, and 'bookid' for keeping the ISBN of the book. We have 1 cursor 'sel_detail' for selecting the amount and ISBN from the **`transaction_detail`** table where 'transaction_id' = 'id'. Then we open the cursor, set the condition when the cursor fetches nothing to 'endloop' = 1 to signal the end of the while loop, start the while loop, and fetch the data from the cursor into 'amnt' and 'bookid'. If 'endloop' is set to 0, set the value in 'pr' by selecting it from the **`stock`** table where 'isbn' = 'bookid'. Then calculate the total price and update the value to 'total'. Continue the loop until nothing is found. The cursor is closed, and 'total' is returned.

- **cal_discount**: Calculate the discount% based on member's point

```
DELEMITER$$
CREATE FUNCTION `cal_discount` (IN `id` INT) RETURNS DECIMAL(8,2)
    BEGIN
    DECLARE discount DECIMAL(8,2) DEFAULT 0;
    DECLARE pnt INT DEFAULT 0;
    SELECT point INTO pnt FROM member WHERE member_id = id;
    IF (pnt <= 4999) THEN
        SET discount = 0.03;
    ELSEIF (pnt >= 5000 AND pnt <= 9999) THEN
        SET discount = 0.05;
    ELSEIF (pnt >= 10000) THEN
        SET discount = 0.1;
    ELSE
        SET discount = 0;
    END IF;
RETURN (discount);
END$$
DELIMETER ;
```

Figure 5: **cal_discount** Function

This function works by using the point of the member from the **`member`** table to return the discount value.

<u>Detailed Explanation</u>: This function accepts one input 'id' and will return one decimal value. These variables are declared: 'discount' to keep the discount value and 'pnt' for keeping the point of the member. We insert the value into 'pnt' by selecting it from the **`member`** table where 'member_id' = 'id'. Then the 'discount' is determined by the 'pnt' using IF statements. The 'discount' is returned.

- 2 Procedures
    - **upd_stock**: Update the stock (Transaction is completed)

```
DELEMITER$$
CREATE PROCEDURE `upd_stock` (IN `id` INT)
    BEGIN
    DECLARE endloop INT DEFAULT 0;
    DECLARE amnt INT;
    DECLARE bookid VARCHAR(13);
    DECLARE sel_detail CURSOR FOR SELECT isbn, amount FROM transaction_detail WHERE transaction_id = id;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET endloop = 1;
    OPEN sel_detail;

    WHILE endloop = 0 DO

    FETCH sel_detail INTO bookid, amnt;
        If endloop = 0 THEN
            UPDATE stock set amount = amount - amnt WHERE stock.isbn = bookid;
        END IF;
    END WHILE;

    CLOSE sel_detail;
END$$
DELIMETER ;
```

Figure 6: **upd_stock** Procedure

This procedure works by using the cursor to fetch the ISBN and the amount of the books that are in the **`transaction_detail`** table by using the 'transaction_id' one by one, and use that information to update the `amount` in the `stock` table, until all of the rows in the **`transaction_detail`** table with the same transaction_id are fetched.

<u>Detailed Explanation</u>: This procedure accepts one input 'id. These variables are declared: 'endloop' for loop condition, 'amnt' for keeping the 'amount' of the book, and 'bookid' for keeping the ISBN of the book. We have 1 cursor 'sel_detail' for selecting the amount and ISBN from the **`transaction_detail`** table where 'transaction_id' = 'id'. Then we open the cursor, set the condition when the cursor fetches nothing to 'endloop' = 1 to signal the end of the while loop, start the while loop, and fetch the data from the cursor into 'amnt' and 'bookid'. If 'endloop' is set to 0, update the stock by subtracting the 'amount' by 'amnt' from the **`stock`** table where 'ISBN' = 'bookid'. Continue the loop until nothing is found.

- **revert_stock**: Revert the stock (The completed transaction is canceled)

```
DELEMITER$$
CREATE PROCEDURE `revert_stock` (IN `id` INT)   BEGIN
    DECLARE endloop INT DEFAULT 0;
    DECLARE amnt INT;
    DECLARE bookid VARCHAR(13);
    DECLARE sel_detail CURSOR FOR SELECT isbn, amount FROM transaction_detail WHERE transaction_id = id;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET endloop = 1;
    OPEN sel_detail;

    WHILE endloop = 0 DO

    FETCH sel_detail INTO bookid, amnt;
        If endloop = 0 THEN
            UPDATE stock set amount = amount + amnt WHERE stock.isbn = bookid;
        END IF;
    END WHILE;

    CLOSE sel_detail;
END$$
DELIMETER ;
```

Figure 7: **revert_stock** Procedure

This procedure works almost exactly the same as in **upd_stock**. The only difference is that instead of subtracting from the ‘amount’, we add the value back.

<u>Detailed Explanation</u>: This procedure accepts one input ‘id. These variables are declared: ‘endloop’ for loop condition, ‘amnt’ for keeping the ‘amount’ of the book, and ‘bookid’ for keeping the ISBN of the book. We have 1 cursor ‘sel_detail’ for selecting the amount and ISBN from the **`transaction_detail`** table where ‘transaction_id’ = ‘id’. Then we open the cursor, set the condition when the cursor fetches nothing to ‘endloop’ = 1 to signal the end of the while loop, start the while loop, and fetch the data from the cursor into ‘amnt’ and ‘bookid’. If ‘endloop’ is set to 0, update the stock by adding the ‘amount’ by ‘amnt’ from the **`stock`** table where ‘ISBN’ = ‘bookid’. Continue the loop until nothing is found.

- 4 Triggers
    - **check_amnt_bf_buy_isrt**: Check the amount of the book before adding it to the cart (BEFORE INSERT)

```
DELIMITER $$
CREATE TRIGGER `check_amnt_bf_buy_isrt` BEFORE INSERT ON transaction_detail
FOR EACH ROW
BEGIN
IF NEW.amount > (SELECT amount FROM stock WHERE stock.isbn = NEW.isbn) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = "The amount exceed the stock quantity!";
END IF;
END$$
DELIMITER ;
```

Figure 8: **check_amnt_bf_buy_isrt** Trigger

This trigger works before the insertion into the **`transaction_detail`** table to check that there are enough books in the stock.

<u>Detailed Explanation</u>: This trigger will be triggered before the insertion into the **`transaction_detail`** table. It will check whether the 'amount' that is going to be inserted exceeds the 'amount' that is selected from the **`stock`** table using ISBN as the index or not. If it is, then signal the error to stop the trigger.

- **check_amnt_bf_buy_updt**: Check the amount of the book before adding it to the cart (BEFORE UPDATE)

```
DELIMITER $$
CREATE TRIGGER `check_amnt_bf_buy_updt` BEFORE UPDATE ON transaction_detail
BEGIN
IF NEW.amount > (SELECT amount FROM stock WHERE stock.isbn = NEW.isbn) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = "The amount exceed the stock quantity!";
END IF;
END$$
DELIMITER ;
```

Figure 9: **check_amnt_bf_buy_updt** Trigger

This trigger works before the update into the **`transaction_detail`** table to check that there are enough books in the **`stock`**.

<u>Detailed Explanation</u>: This trigger will be triggered before the update into the **`transaction_detail`** table. It will check whether the 'amount' that is going to be inserted exceeds the 'amount' that is selected from the **`stock`** table using ISBN as the index or not. If it is, then signal the error to stop the trigger.

- **upd_stock_on_sale**: Call **upd_stock**

```
DELIMITER $$
CREATE TRIGGER `upd_stock_on_sale` BEFORE UPDATE ON transaction
FOR EACH ROW
BEGIN
        IF (OLD.totalprice=0 AND NEW.totalprice > OLD.totalprice) THEN
                CALL upd_stock(OLD.transaction_id);
        END IF;
END$$
DELIMITER ;
```

Figure 10: **upd_stock_on_sale** Trigger

This trigger works before the update into the **`transaction`** table to call the **upd_stock procedure**.

Detailed Explanation: This trigger will be triggered before the update into the **`transaction`** table. It will check whether the update is about the 'totalprice' and whether the update is done when the 'totalprice' is at its default value. If it is, call the **upd_stock** using the existing 'transaction_id' as the input.

- **revert_stock_on_del**: Call **revert_stock**

```
DELIMITER $$
CREATE TRIGGER `revert_stock_on_del` BEFORE DELETE ON transaction
FOR EACH ROW
BEGIN
        IF (OLD.totalprice!=0) THEN
                CALL revert_stock(OLD.transaction_id);
        END IF;
END$$
DELIMITER ;
```

Figure 11: **revert_stock_on_del** Trigger

This trigger works before the update into the **`transaction`** table to call the **revert_stock** procedure.

<u>Detailed Explanation</u>: This trigger will be triggered before the update into the **`transaction`** table. It will check whether the update is about the 'totalprice' and whether the update is done when the 'totalprice' is not at its default value. If the condition is cleared, call the **revert_stock** using the existing 'transaction_id' as the input.

Next is about the User Interface and how they interact with the database.
Note: Sometimes, **“SELECT *”** is used but not all the information is used or displayed. This is intentional for the backend reason.

- Login Page

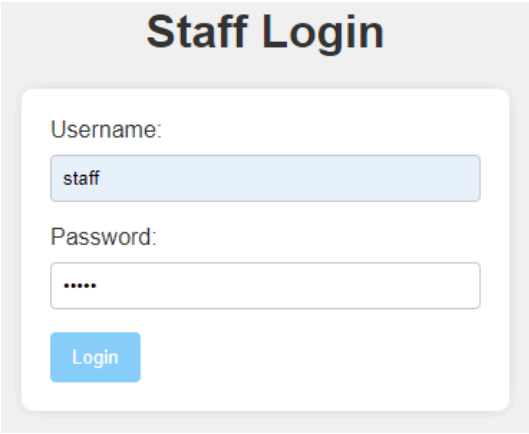


Figure 12: Login Page

When a staff log in, the system will check username of the staff and use query **"SELECT * FROM staff WHERE username = '$username';"** to get information of that staff. We verify passwords by using **“password_verify($password, $row['password'])”** which is a PHP function. After logging in, we set the “staff_id” and “staff_name” as the global variables for this login session by using **“$_SESSION['staff_id'] = $row['staff_id'];”** (will be used as **$staffId** in the queries) and **“$_SESSION['staff_name'] = $row['staff_name'];”** (will be used as **$staffName** in the queries), so that the system will have the value of the staff currently using it. Then the user will be redirected to the ‘Scan’ page.

- Scan Page

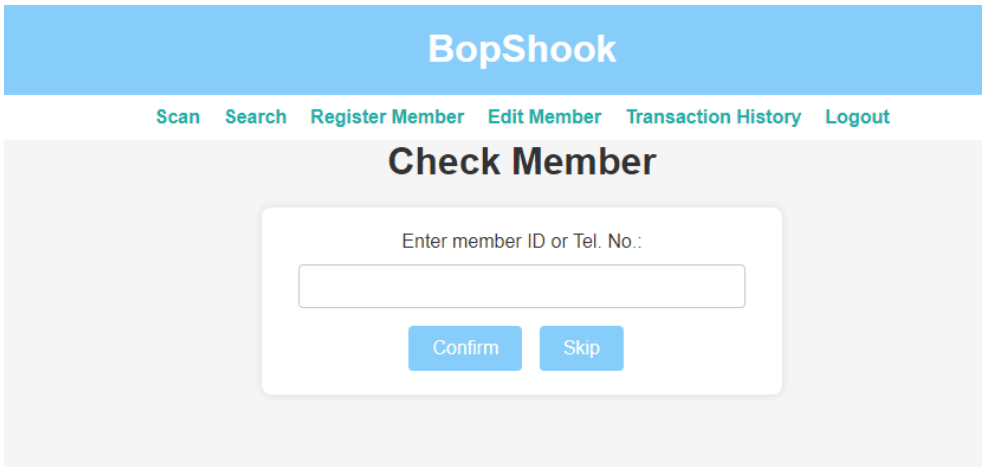


Figure 14: Scan page

When customers want to purchase our book, they will tell the membership_id or the telephone number to staff to collect the points and get a discount. We will insert that information into the text box to check the membership and click the 'Confirm' button. The system will check the membership by using query **"SELECT * FROM member WHERE member_id = $memberInfo OR tel_no = '$memberInfo';".**

The system will set the 'member_id' and 'member_name' to be global variables for this transaction session by **" $_SESSION['member_id'] = $member['member_id']; "** (will be used as **$memberId** in the queries) and **"$_SESSION['member_name'] = $member['member_name'];"** (will be used as **$memberName** in the queries)**.**

But some customers might not have the membership, they can also choose to register the membership or choose to skip this process.

- Membership Confirmation Page

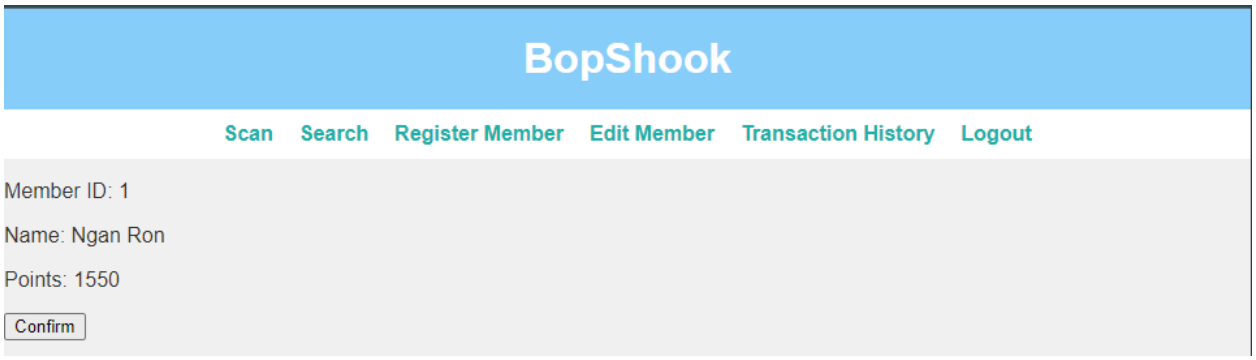


Figure 15: Membership Confirmation Page

On this page, **"SELECT * FROM member WHERE member_id = '$memberId';"** is used to fetch the displayed information.

If the customer chose to use their membership, the staff will be redirected to this page to check the member's info. After we click on the 'Confirm' button in this page, the system automatically moves to the Transaction Page and creates a row in the **`transaction`** table using query **"INSERT INTO transaction (staff_id, member_id) VALUES ('$staffId', '$memberId')".**

The values inserted in the query come from the global variables we set earlier. In this process, we can use a PHP function to get the **'transaction_id'** from the insertion, and set that into a global variable **$_SESSION['transaction_id'] (**will be used as **$transactionId** in the queries).

- Transaction Page

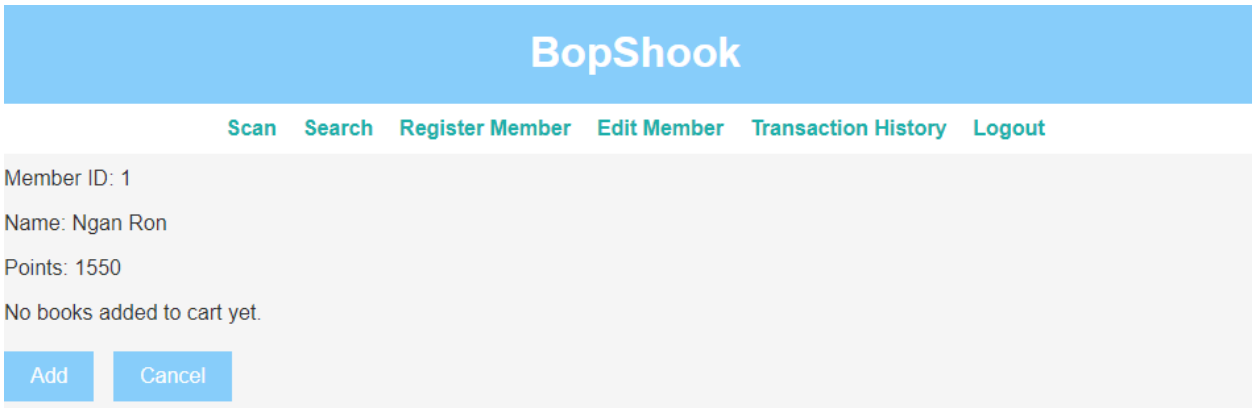


Figure 16:Transaction Page

On this page, staff can add books into the cart or cancel the transaction. Staff can cancel by clicking the “Cancel” button. This will also delete the data inside the **`transaction`** table by **"DELETE FROM transaction WHERE transaction_id = '$transactionId';"** and redirect them back to the scan page. If the staff want to add the book to the cart, they have to click the “add” button which will send them to the “Add to Cart” page.

- Add to Cart Page

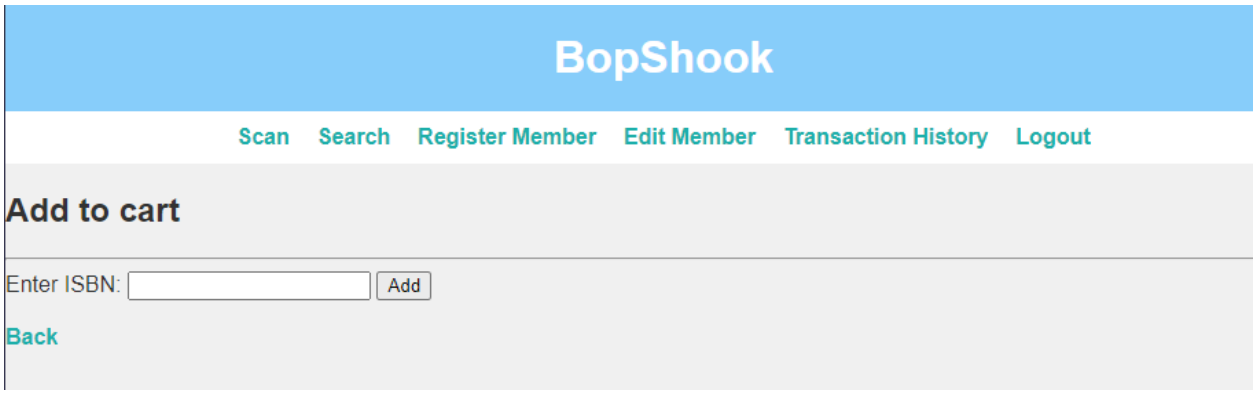


Figure 17: Add to cart Page

In this page, staff will fill in the ISBN of the book and click the “Add” button to send the information to the backend. There, the data will be inserted into **`transaction_detail`** by **“INSERT INTO transaction_detail (transaction_id, isbn, amount) VALUES ('$transactionId', '$isbn', 1);”.** This will trigger **check_amnt_bf_buy_isrt.**

- Transaction Page (With data in `**transaction_detail**` table)

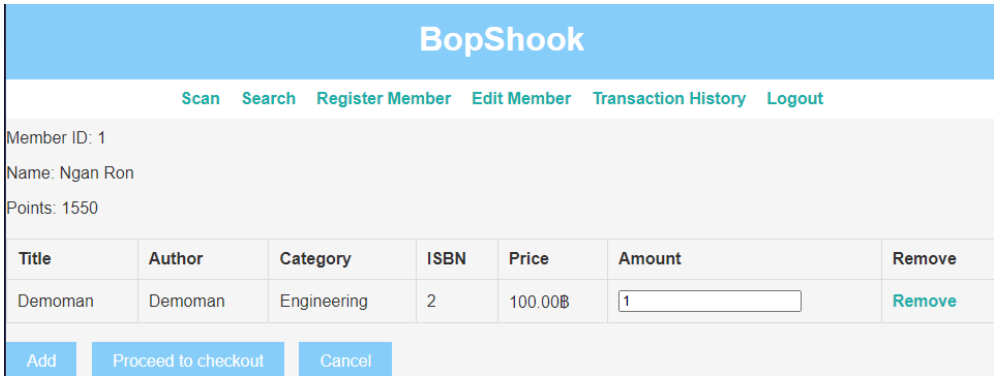


Figure 18: Transaction Page with information about the book

When back to the transaction page, this query **"SELECT td.isbn, td.amount, s.book_title, s.author, s.category, s.price FROM transaction_detail td INNER JOIN stock s ON td.isbn = s.isbn WHERE td.transaction_id = '$transactionId';"** is used to fetch the book information to be displayed in the page.

Staff can set the amount of each book, add more books, and remove books out of this transaction.

If the customer decides to cancel this transaction, **"DELETE FROM transaction WHERE transaction_id = '$transactionId';"** will run.

If a book needs to be removed from the cart, after clicking 'Remove', this **"DELETE FROM transaction_detail WHERE transaction_id = '$transactionId' AND isbn = '$isbn';"** will run.

When all of the books are added to the cart and their amount is adjusted, the staff can click "Proceed to checkout" to calculate the price.

- Completing the Transaction Process

After clicking the “Proceed to checkout” button, these will happen.

The amount we set will be used to update the ‘amount’ in the `transaction_detail` table by **"UPDATE transaction_detail SET amount = $amount WHERE transaction_id = '$transactionId' AND isbn = '$isbn';".** This will trigger **check_amnt_bf_buy_updt.**

cal_price and **cal_discount** are called to calculate the net total.

That net total will be used to update the ‘totalprice’ in the **"UPDATE transaction SET totalprice = '$totalprice' WHERE transaction_id = '$transactionId';".** This will trigger **upd_stock_on_sale.**

The ‘transaction_time’ will also be updated by **"UPDATE transaction SET transaction_time = CURRENT_TIMESTAMP() WHERE transaction_id = '$transactionId';"**

If the customer is a member, their point will be updated by the net total using **"UPDATE member SET point = point + (SELECT totalprice FROM transaction WHERE transaction_id = '$transactionId') WHERE member_id = '$memberId'";**

- Transaction Summary Page

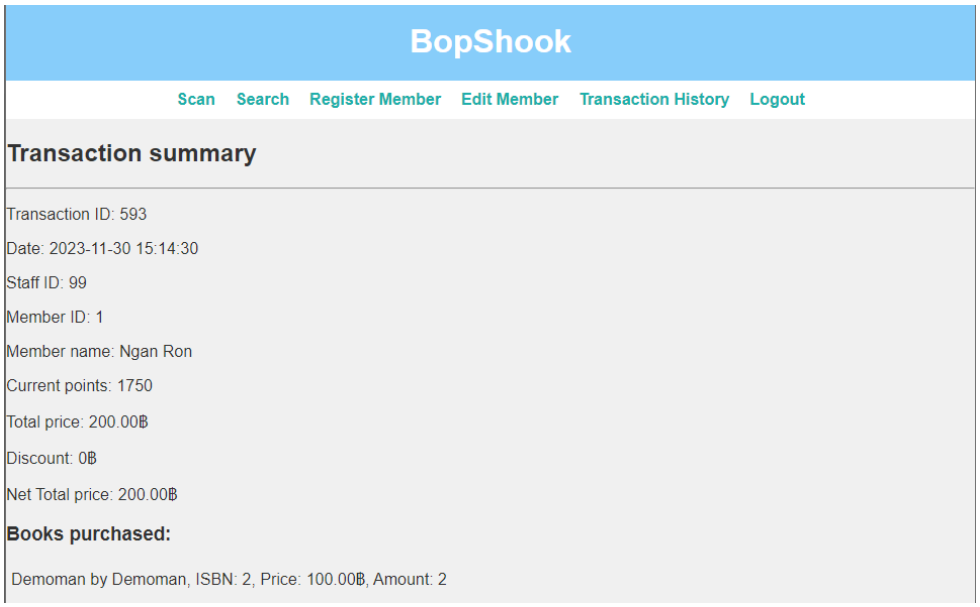


Figure 19: Transaction Summary Page

After clicking “Proceed to checkout”, it will show the all info of this transaction by using the information from three SELECT queries:

1. **"SELECT * FROM transaction WHERE transaction_id = '$transactionId';"**
2. **"SELECT * FROM member WHERE member_id = '$memberId';"**
3. **"SELECT td.amount, s.book_title, s.author, s.category, s.isbn, s.price**
   FROM transaction_detail td
   INNER JOIN stock s ON td.isbn = s.isbn
   WHERE td.transaction_id = '$transactionId';"

- Search Page

Figure 20: Search page and search result

Staff can check the book in stock by searching the title, author, category, or ISBN of the book. In the system, we use query **“SELECT * FROM stock WHERE isbn = '$searchTerm' OR book_title LIKE '%$searchTerm%' OR author LIKE '%$searchTerm%' OR category LIKE '%$searchTerm%';"** where **$searchTerm** is the text that staff fill in the search box.

- Register Member Page

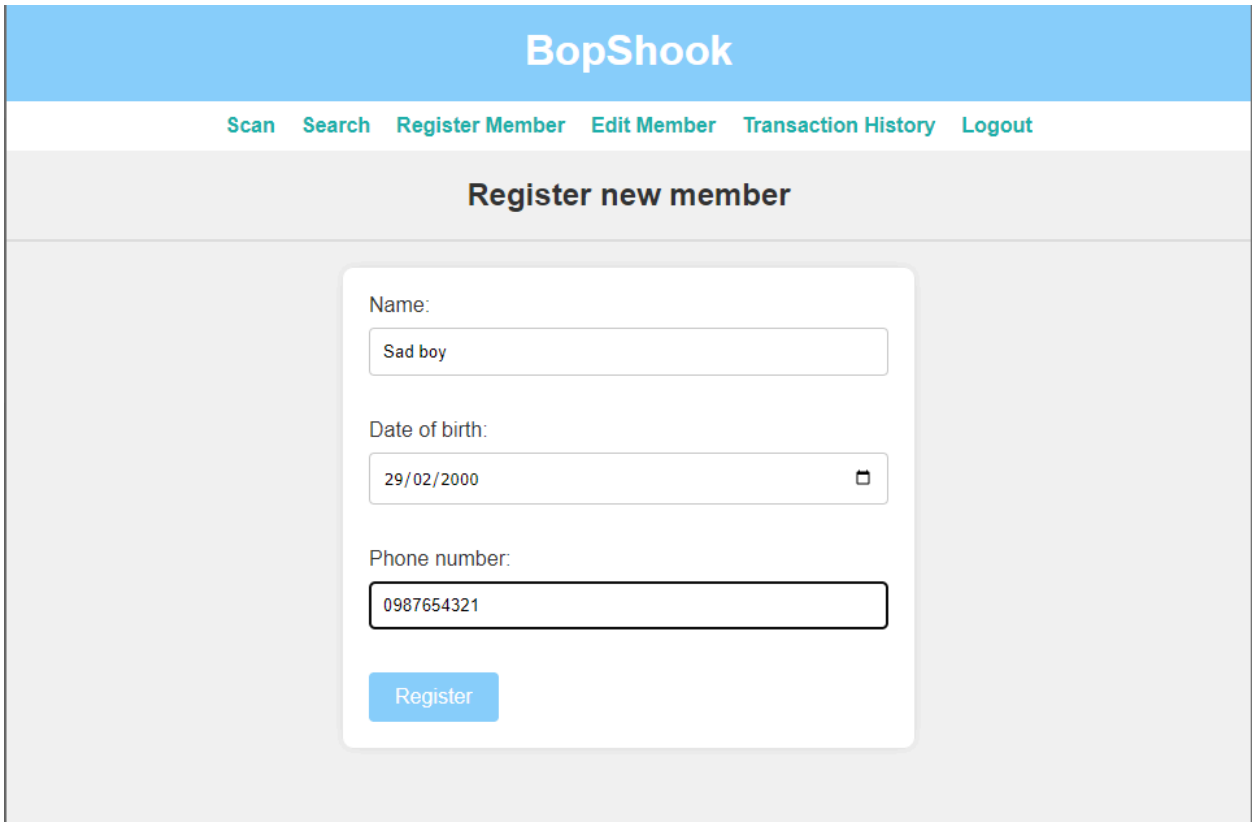


Figure 21: Register Member Page

This page is used to register new members into our store.

When staff fill in the information of customers and click 'Register', we use query **"INSERT INTO member (member_name, dob, tel_no) VALUES ('$memberName', '$dob', '$telNo');"** to register this customer as one of our members. The values inserted in the query come from the information filled in those fields. Then the system will redirect to the Scan Page.

- Edit Member Page

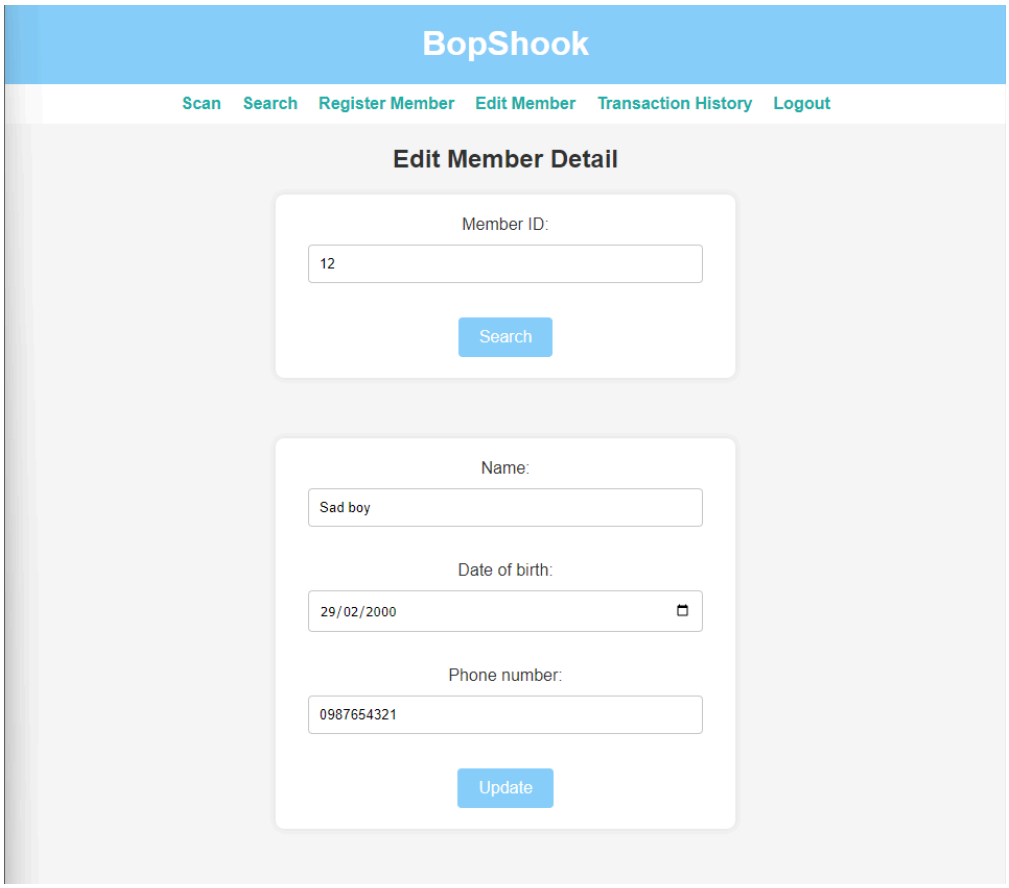
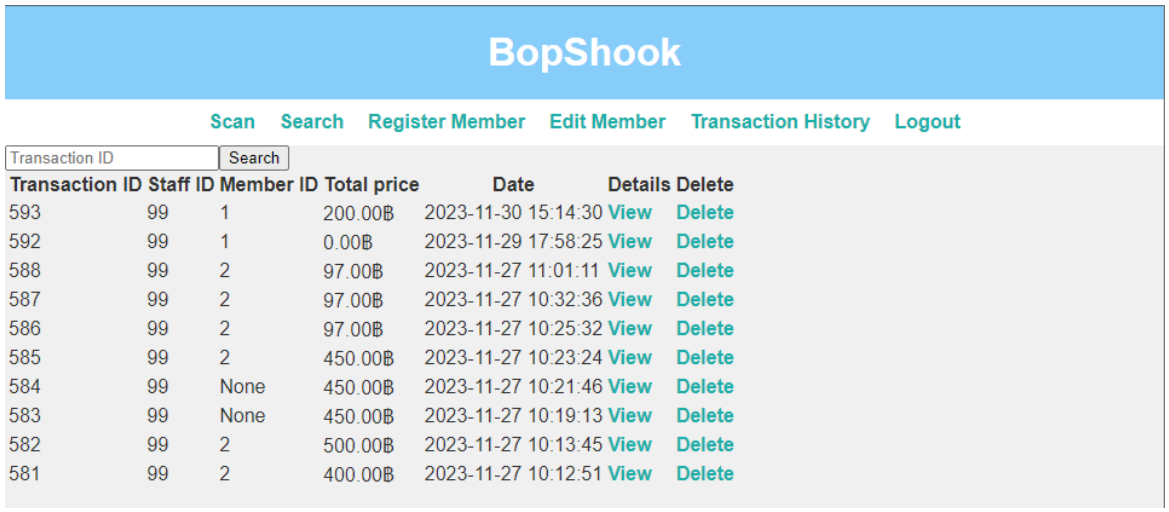


Figure 22: Edit Member Data Page

We can edit the member information too. First, staff must insert the member_id of the member, then click the “Search” button to see and edit the data. The member data will be fetched by **"SELECT * FROM member WHERE member_id = '$memberId';".** After the data is edited, staff have to click “Update” button to update the member information using **“UPDATE member SET member_name = '$memberName', dob = '$dob', tel_no = '$telNo' WHERE member_id = '$memberId';”.**

- Transaction History Page



Transaction ID	Staff ID	Member ID	Total price	Date	Details	Delete
593	99	1	200.00฿	2023-11-30 15:14:30	View	Delete
592	99	1	0.00฿	2023-11-29 17:58:25	View	Delete
588	99	2	97.00฿	2023-11-27 11:01:11	View	Delete
587	99	2	97.00฿	2023-11-27 10:32:36	View	Delete
586	99	2	97.00฿	2023-11-27 10:25:32	View	Delete
585	99	2	450.00฿	2023-11-27 10:23:24	View	Delete
584	99	None	450.00฿	2023-11-27 10:21:46	View	Delete
583	99	None	450.00฿	2023-11-27 10:19:13	View	Delete
582	99	2	500.00฿	2023-11-27 10:13:45	View	Delete
581	99	2	400.00฿	2023-11-27 10:12:51	View	Delete

Figure 23: Transaction History Page

In this page, It show latest 10 transaction by using query **“SELECT * FROM transaction ORDER BY transaction_time DESC LIMIT 10;”**. Staff can also search a specific transaction by the transaction_id. After clicking ‘Search’, this query **"SELECT * FROM transaction WHERE transaction_id = '$searchquery';"** will run.

After clicking ‘View’, the page that looks like the Transaction Summary Page will be pulled up.

If for whatever reason, a transaction needs to be reverted, the staff can click ‘Delete’ to delete the transaction by using this query **"DELETE FROM transaction WHERE transaction_id = '$transactionId';".** This will trigger **revert_stock_on_del.**

- Logout

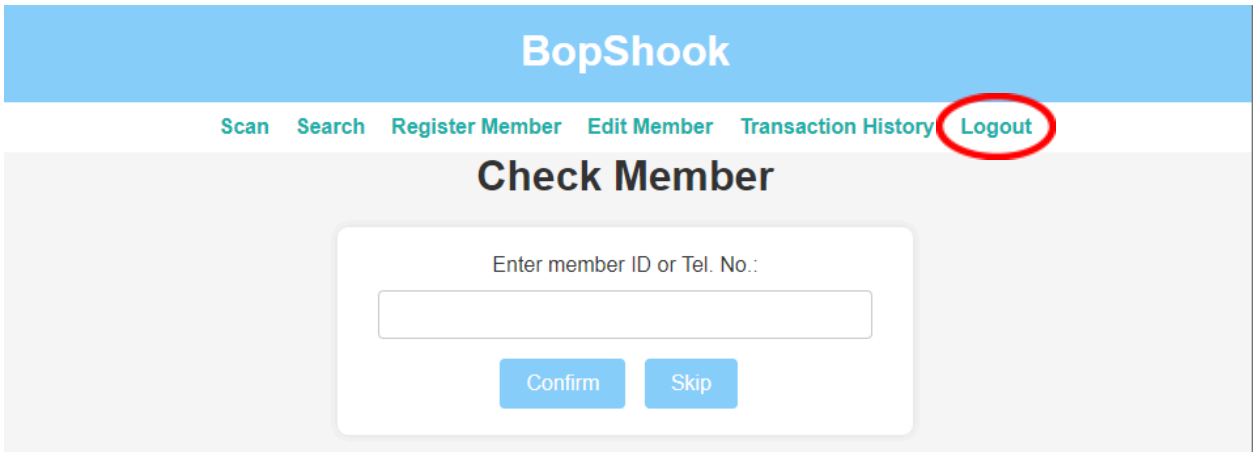


Figure 24: The logout button

The last function of our system is Logout. When staff is off, They must log out of the system. After clicking the “Logout” button, we use **“session_destroy();”** to reset the global variable that we set after logging in and head back to the “login” page.

CONCLUSION

In conclusion, the creation of the database management system is successful. We managed to create a database and the interfaces with all of the functions we had planned.

This project, in this state, is not practical in the real situation yet, but the experiences we gained from doing this project and the project itself can serve as a base that we can come back to review or improve further in the future.