WelcomeHome

Principles of Database Systems Project - github repository

Shreyash Dhamane - sd5971
Makesh Srinivasan - ms15138
Shubhan Kadam - sk12159

Project Description

WelcomeHome, a non-profit similar to Ruth's Refuge, seeks to improve its operations for tracking donations, orders, deliveries, clients, donors, and volunteers. Serving refugees and asylum seekers in need of housewares and furniture, the organization accepts donations of used items and monetary contributions for new purchases. Clients use an online portal to select needed items, which volunteers then collect and label for delivery to their homes. To enhance efficiency and service delivery, Welcome Home requires a streamlined management system.

Languages and Frameworks

To effectively manage the operations of Welcome Home, the following were used:

1. HTML/CSS/JavaScript: HTML structures the web pages, CSS styles the layout and appearance, and JavaScript adds interactivity and dynamic functionality to the user interface.
2. Python: The primary programming language used in the project
3. Flask: Lightweight Python web framework used for building web applications. It provides tools and libraries for routing web requests, handling server-side logic, and rendering dynamic web pages with minimal configuration.
4. MySQL: To implement and store the database needed for WelcomeHome.

Schema changes/additional constraints & triggers

The following are the schema changes made to the original WelcomeHome schema:

- Person Table Changes:
  These changes suggest improvements in password security (longer password field and salt for hashing)
  1. The password field's `VARCHAR` length has been increased from 100 to 1000.
  2. A new salt field has been added with `VARCHAR(64) NOT NULL`.

- Ordered Table Changes:
  These changes suggest improvements in order status tracking
  1. A new stat field has been added with `VARCHAR(50) DEFAULT` "Initiated"

- Additional Changes:
  Automating the assignment of the client role to new users.
  1. A new trigger `after_user_insert` has been added to automatically insert a 'client' role for new users in the act table.

Main SQL Queries for the features implemented

The following are the basic features described for WelcomeHome, as described by the Project Guidelines.

Register/Login & User Session Handling:

User Existence Check: Checks if a username already exists in the Person table, returns 1 if the username exists, else returns nothing.

```
SELECT 1 FROM Person WHERE userName = %s
```

User Registration: Inserts a new user's information into the Person table, stores the hashed password and salt for security

```
INSERT INTO Person (userName, password, fname, lname, email, salt)
VALUES (%s, %s, %s, %s, %s, %s)
```

Client Role Assignment: Assigns the "Client" role to the newly registered user in the Act table.

```
INSERT INTO Act (userName, roleID)
VALUES (%s, %s)
```

SQL Trigger:

```
DELIMITER $$
CREATE TRIGGER after_user_insert
AFTER INSERT ON Person
FOR EACH ROW
BEGIN
    INSERT INTO Act (userName, roleID)
    VALUES (NEW.userName, 'client');
END$$
DELIMITER ;
```

Find Single Item:

Item Existence Check: Checks if an item with the given ItemID exists in the Item table, retrieves the ItemID and hasPieces flag for the item

```
SELECT ItemID, hasPieces FROM Item WHERE ItemID = %s
```

Item Details Retrieval: Retrieves detailed information about an item from the Item table, fetches ItemID, description, hasPieces flag, color, material, and isNew status

```
SELECT ItemID, iDescription, hasPieces, color, material, isNew FROM Item WHERE ItemID = %s
```

Item Pieces and Location Retrieval: Joins the Item, Piece, and Location tables to retrieve comprehensive information, fetches details about each piece of the item, including dimensions and location

```
SELECT i.ItemID, i.iDescription, p.pieceNum, p.pDescription,
p.length, p.width, p.height, l.roomNum, l.shelfNum, l.shelf, l.shelfDescription
FROM Item i
JOIN Piece p ON i.ItemID = p.ItemID
JOIN Location l ON p.roomNum = l.roomNum AND p.shelfNum = l.shelfNum
WHERE i.ItemID = %s
```

Find Order Items:

Order Item Retrieval: query retrieves detailed information about items in an order, including their pieces and locations.

```
SELECT I.ItemID, I.iDescription, I.color, I.hasPieces,
P.pieceNum, P.pDescription, P.length, P.width, P.height,
L.roomNum, L.shelfNum, L.shelf, L.shelfDescription
FROM ItemIn AS II
JOIN Item AS I ON II.ItemID = I.ItemID
LEFT JOIN Piece AS P ON I.ItemID = P.ItemID
LEFT JOIN Location AS L ON P.roomNum = L.roomNum AND P.shelfNum = L.shelfNum
WHERE II.orderID = %s
ORDER BY I.ItemID, P.pieceNum
```

Accept Donation:

Check if category exists: Verifies if the specified category exists in the database.

```
SELECT 1 FROM Category WHERE mainCategory=%s AND subCategory=%s
```

Check if location exists: Confirms if the specified location (room and shelf) exists in the database.

```
SELECT 1 FROM Location WHERE roomNum=%s AND shelfNum=%s
```

Insert new item: Adds a new item to the Item table with the provided details.

```
INSERT INTO Item (ItemID, iDescription, photo, color, isNew, hasPieces, material, mainCategory,
subCategory)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
```

Record donation: Records the donation in the DonatedBy table, linking the item to the donor and setting the donation date to the current date.

```
INSERT INTO DonatedBy (ItemID, userName, donateDate)
VALUES (%s, %s, CURDATE())
```

Insert piece information: Adds piece information for items with multiple pieces to the Piece table.

```
INSERT INTO Piece (ItemID, pieceNum, pDescription, length, width, height, roomNum, shelfNum,
pNotes)
VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s)
```

Start an order:

Fetch client usernames: Retrieves all usernames except the current user's for client selection.

```
SELECT userName FROM Person WHERE userName ! =  %s
```

Validate client existence: Checks if the selected client exists in the database.

```
SELECT userName FROM Person WHERE userName = %s
```

Check staff privilege: Verifies if the user has staff or supervisor role.

```
SELECT A.roleID FROM Act A JOIN Role R ON A.roleID = R.roleID WHERE A.userName = %s
```

Create new order: Inserts a new order into the Ordered table with the current date, notes, supervisor, and client.

```
INSERT INTO Ordered (orderDate, orderNotes, supervisor, client)
VALUES (CURDATE(), %s, %s, %s)
```

Add to current order:

Query available items: Retrieves available items for a specific category that are not already in an order.
```
SELECT I.ItemID, I.iDescription, I.color, I.isNew, I.hasPieces, I.material
FROM Item I
WHERE I.mainCategory=%s AND I.subCategory=%s
AND I.ItemID NOT IN (SELECT ItemID FROM ItemIn)
```

Fetch distinct main categories: Retrieves all unique main categories from the Item table.
```
SELECT DISTINCT mainCategory FROM Item
```

Fetch distinct sub categories: Retrieves all unique sub categories from the Item table.
```
SELECT DISTINCT subCategory FROM Item
```

Insert items into order: Adds selected items to the current order in the ItemIn table.
```
INSERT INTO ItemIn (ItemID, orderID, found) VALUES (%s, %s, FALSE)
```

Check staff privilege: Verifies if the user has a staff or supervisor role.
```
SELECT A.roleID FROM Act A JOIN Role R ON A.roleID = R.roleID WHERE A.userName = %s
```

Prepare order:

Check if order exists: Retrieves order information for a given order ID.
```
SELECT * FROM Ordered WHERE orderID=%s
```

Find orders for a client: Retrieves all orders for a specific client.
```
SELECT * FROM Ordered WHERE client=%s
```

Move items to holding area: Updates the location of all pieces for items in a specific order to the holding area.
```
UPDATE Piece P
JOIN ItemIn II ON P.ItemID = II.ItemID
SET P.roomNum = 999, P.shelfNum = 999
WHERE II.orderID = %s
```

Update order status to prepared: Changes the status of a specific order to 'prepared'.
```
UPDATE Ordered
SET stat = 'prepared'
WHERE orderID = %s
```

Retrieve items in an order: Fetches all items associated with a specific order.
```
SELECT I.ItemID, I.iDescription, I.color, I.isNew, I.hasPieces, I.material
FROM ItemIn II
JOIN Item I ON II.ItemID = I.ItemID
.WHERE II.orderID = %s
```

Retrieve detailed order information: Retrieves detailed information about items, their pieces, and locations for a specific order.
```
SELECT I.ItemID, I.iDescription, I.color, I.hasPieces,
P.pieceNum, P.pDescription, P.length, P.width, P.height,
L.roomNum, L.shelfNum, L.shelf, L.shelfDescription
```

```
FROM ItemIn AS II
JOIN Item AS I ON II.ItemID = I.ItemID
LEFT JOIN Piece AS P ON I.ItemID = P.ItemID
LEFT JOIN Location AS L ON P.roomNum = L.roomNum AND P.shelfNum = L.shelfNum
WHERE II.orderID = %s
ORDER BY I.ItemID, P.pieceNum
```

User's tasks:

Retrieve user-related orders and donations: This query combines orders supervised by the user, orders where the user is a client, and donations made by the user.

```
SELECT
o.orderID, o.orderDate, o.orderNotes, 'Supervisor' AS Role
FROM Ordered o
WHERE o.supervisor = %s
UNION
SELECT
o.orderID, o.orderDate, o.orderNotes, 'Client' AS Role
FROM Ordered o
WHERE o.client = %s
UNION
SELECT
d.ItemID, d.donateDate AS orderDate, 'Donated an item' AS orderNotes, 'Donor' AS Role
FROM DonatedBy d
WHERE d.userName = %s
```

Rank System:

Fetch popular categories: This query retrieves categories sorted by the number of orders they appear in within a specified date range.

```
SELECT i.mainCategory, i.subCategory,
COUNT(DISTINCT o.orderID) AS totalOrders
FROM Item i
JOIN ItemIn ii ON i.ItemID = ii.ItemID
JOIN Ordered o ON ii.orderID = o.orderID
WHERE o.orderDate BETWEEN %s AND %s
GROUP BY i.mainCategory, i.subCategory
ORDER BY totalOrders DESC;
```

Update enabled:

Check if order exists: Verifies if the specified order ID exists in the Ordered table.

```
SELECT 1 FROM Ordered WHERE orderID = %s
```

Update order status: Updates the status of an order to a new specified status, but only if the current status is 'prepared'.

```
UPDATE Ordered SET stat = %s WHERE orderID = %s AND stat = 'prepared'
```

Year End Report:

Number of clients served: Counts the unique clients who placed orders within the specified year.

```
SELECT COUNT(DISTINCT client) AS num_clients
```

```
FROM Ordered
WHERE orderDate BETWEEN %s AND %s
```

Number of items donated by category: Counts donated items grouped by category for the specified year.

```
SELECT C.mainCategory, C.subCategory, COUNT(DISTINCT D.ItemID) AS donated_items
FROM DonatedBy D
JOIN Item I ON D.ItemID = I.ItemID
JOIN Category C ON I.mainCategory = C.mainCategory AND I.subCategory = C.subCategory
WHERE D.donateDate BETWEEN %s AND %s
GROUP BY C.mainCategory, C.subCategory
```

Summary of orders processed: Counts the total number of orders processed within the specified year.

```
SELECT COUNT(*) AS orders_processed
FROM Ordered
WHERE orderDate BETWEEN %s AND %s
```

Most frequent categories ordered by clients: Identifies the top 5 most frequently ordered categories for the specified year.

```
SELECT C.mainCategory, C.subCategory, COUNT(*) AS order_count
FROM Ordered O
JOIN ItemIn I ON O.orderID = I.orderID
JOIN Item It ON I.ItemID = It.ItemID
JOIN Category C ON It.mainCategory = C.mainCategory AND It.subCategory = C.subCategory
WHERE O.orderDate BETWEEN %s AND %s
GROUP BY C.mainCategory, C.subCategory
ORDER BY order_count DESC
LIMIT 5
```

Top 5 clients with the most donations: Lists the top 5 clients who made the most donations in the specified year.

```
SELECT D.userName, COUNT(DISTINCT D.ItemID) AS donation_count
FROM DonatedBy D
JOIN Item I ON D.ItemID = I.ItemID
WHERE D.donateDate BETWEEN %s AND %s
GROUP BY D.userName
ORDER BY donation_count DESC
LIMIT 5
```

Difficulties & Challenges

Initially, we were maintaining a single file for all the APIs, which grew fast and was difficult to manage. The same issue occurred with the front-end code; it was difficult to change anything quickly, so we modularized the code. We learned that it is very important to modularize the code from the start.

Also, we faced issues with the connection to the database. We were closing the connection and reconnecting for each API function, which was very inefficient, so we had to modify the code again. Now, the connection remains open, giving faster results.

The front-end was not reactive and JavaScript was not working. After spending some time, we found out that, since we implemented protection against XSS attacks, inline JavaScript was disabled. So, we had to create a JS file for all front-end functions and send it over during the request.

Also, after implementing a few features and adding data, we later had to modify the schema for two tables, which invalidated the previous data, and changes had to be made in previous API calls. What we learned is that it's best to have the table schema as perfect as possible from the start.

Although writing the SQL part was simpler, we had to make sure it's protected from different attacks. We had to write a lot of tests to check that other types of attacks or invalid query execution didn't occur, and even if they did, the system shouldn't crash .

Team Roles

Makesh Srinivasan - ms15138

- Initial Project Setup with database connection
- Implemented Login API with frontend
- Implemented SignUp API with frontend
- Partially Implemented Accept Donation Feature
- Implemented Start Order API with frontend
- Implemented Add To Current Order API with frontend
- Implemented Prepare Order Feature
- Implemented test cases

Shreyash Dhamane - sd5971

- Implemented Salt and Hashing for login and registration API's
- Completed Accept Donation Feature
- Implemented Update Delivery status feature with frontend
- Implemented Year end report with frontend
- Implemented A feature for staff registration apart from client registration with Frontend
- Added Protection For XSS Attacks
- Implemented the `after_user_insert` Trigger and updated the table schemas
- Updated the API's for multi user system with client, staff, donor, supervisor views

Shubhan Kadam - sk12159

- Incorporated Sessions
- Implemented Find Single Item Feature with frontend
- Implemented Find Order Item Feature with frontend
- Implemented User Tasks feature with frontend
- Implemented Rank System feature with frontend
- Implemented HomePage
- Tested all API's for invalid inputs
- Implemented User Orders for client role with frontend

Each of us also added proper error handling, input validation, worked with CSS, and tested each other's features for bugs.