Final Project - PDS

Languages and Frameworks Used

- **Languages**:
  - Python (Flask for web backend development).
  - SQL (MySQL for database schema and queries).
- **Frameworks and Libraries**:
  - Flask: For routing, session management, and form handling.
  - Werkzeug: For cryptographic password hashing (SHA-256 with salt).
  - Jinja2: For template rendering in HTML.
- **Tools**
  - VSCode : For coding
  - MySql workbench - for sql queries

Schema Changes and Their Purpose

Feature	Schema Changes (Old vs. New)	Purpose
Password Handling	Increased `password` column size in `Person` from `VARCHAR(100)` to `VARCHAR(255)`.	To support stronger cryptographic hashing mechanisms with salt.
Donor Role Verification	Added `client` role to the `Role` table and allowed assignments via the `Act` table.	Enabled role-based access for both clients and staff for actions like managing orders and donations.
Tracking Item Pieces	Retained `pieceNum` for unique identification of item pieces in the	Supported precise tracking of individual item pieces within orders.

	`Piece` table and added references in the `ItemIn` table.	
Order Management	Added logic to update `roomNum` and `shelfNum` in `Piece` table for holding location during `Prepare Order`.	Ensured ordered items were marked unavailable and moved to a designated delivery holding area.

Additional Constraints, Triggers, Stored Procedures

- Constraints: Foreign keys and primary keys to maintain referential integrity and uniqueness.
- Triggers: None implemented.
- Stored Procedures: None implemented.
- Other mechanisms: Parameterized queries for security and data integrity.

Main Queries for Features

1. Login & User Session Handling

- **Purpose**: Authenticate users with roles and session handling.

```
-- Fetch user details for authentication
SELECT * FROM Person WHERE userName = %s;

-- Retrieve user roles for access control
SELECT roleID FROM Act WHERE userName = %s;
```

2. Find Single Item

- **Purpose**: Retrieve locations of all pieces of an item.

```
-- Fetch piece details for a given itemID
```

```
SELECT p.pieceNum, l.shelfDescription AS address
FROM Piece p
JOIN Location l ON p.roomNum = l.roomNum AND p.shelfNum = l.shelfNum
WHERE p.ItemID = %s;
```

3. Find Order Items

- **Purpose**: Return all items and their piece locations for a specific order.

```
-- Fetch items and their locations for an order
SELECT i.ItemID, i.iDescription AS itemName, l.shelfDescription AS address, p.pieceNum
FROM ItemIn ii
JOIN Piece p ON ii.ItemID = p.ItemID AND ii.pieceNum = p.pieceNum
JOIN Item i ON p.ItemID = i.ItemID
JOIN Location l ON p.roomNum = l.roomNum AND p.shelfNum = l.shelfNum
WHERE ii.orderID = %s;
```

4. Accept Donation

- **Purpose**: Allow staff to accept donations and register them in the system.

```
-- Insert a new item
INSERT INTO Item (iDescription, mainCategory, subCategory) VALUES (%s, %s, %s);

-- Add the donor's record
INSERT INTO DonatedBy (ItemID, userName, donateDate) VALUES (%s, %s, %s);
```

```
-- Add a piece for the donated item
INSERT INTO Piece (ItemID, pieceNum, pDescription, length, width, height, roomNum, shelfNum)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s);
```

5. Start an Order

- **Purpose**: Create a new order for a specific client.

```
-- Create a new order record
INSERT INTO Ordered (orderDate, orderNotes, supervisor, client) VALUES (%s, %s, %s, %s);
```

6. Add to Current Order

- **Purpose**: Allow staff to add items to an ongoing order.

```
-- Fetch available items for selection
SELECT i.ItemID, i.iDescription, p.pieceNum
FROM Item i
JOIN Piece p ON i.ItemID = p.ItemID
WHERE i.mainCategory = %s AND i.subCategory = %s
AND NOT EXISTS (
    SELECT 1 FROM ItemIn ii WHERE ii.ItemID = p.ItemID AND ii.pieceNum = p.pieceNum
);

-- Add selected item to the order
INSERT INTO ItemIn (ItemID, pieceNum, orderID) VALUES (%s, %
s, %s);
```

7. Prepare Order

- **Purpose**: Update items to a holding location, making them unavailable.

```
-- Mark items as prepared for delivery
UPDATE Piece p
JOIN ItemIn ii ON p.ItemID = ii.ItemID AND p.pieceNum = ii.pieceNum
SET p.roomNum = 999, p.shelfNum = 999
WHERE ii.orderID = %s;
```

8. User’s Tasks

- **Purpose**: Show all orders linked to the logged-in user.

```
-- Fetch orders related to the logged-in user
SELECT o.orderID, o.orderDate, o.orderNotes, o.supervisor, o.client, d.status, d.date AS deliveryDate
FROM Ordered o
LEFT JOIN Delivered d ON o.orderID = d.orderID
WHERE o.client = %s OR o.supervisor = %s OR d.userName = %s;
```

9. Rank System

- **Purpose**: Rank volunteers by the number of tasks completed within a time period.

```
-- Rank volunteers by number of orders delivered in the last
```

```
30 days
SELECT d.userName, COUNT(*) as delivered_count
FROM Delivered d
WHERE d.date >= CURDATE() - INTERVAL 30 DAY
GROUP BY d.userName
ORDER BY delivered_count DESC;
```

Difficulties and Lessons Learned

- **Difficulties**:
    - Migrating from the old schema required careful management of foreign key relationships, especially around `pieceNum` and `ItemIn`.
    - Role-based access control required extending both the database and application logic to handle multiple user roles.
    - Testing dynamic queries and ensuring they handled edge cases (e.g., nonexistent IDs) was time-intensive.
- **Lessons Learned**:
    - Schema changes should be planned and documented thoroughly to avoid breaking dependencies.
    - Implementing cryptographic password handling significantly improved security but required updates to accommodate longer hash values.
    - Role-based access ensured scalability and maintainability for future feature additions.
    - **Security Mechanisms**
        - **SQL Injection Prevention:** All database queries were parameterized using placeholders (e.g., `%s`), ensuring that user inputs are not executed as part of the query. This effectively mitigates SQL injection risks.
        - **XSS (Cross-Site Scripting) Mitigation:** Inputs and outputs were sanitized. User-provided data, such as form inputs and rendered

HTML, were properly escaped to prevent malicious scripts from being injected and executed.

- **Password Security:** Passwords are securely hashed with cryptographic hashing (e.g., `generate_password_hash` using bcrypt or SHA-256) and salted before storage, ensuring they cannot be directly reversed or cracked if the database is compromised.

Team Contributions

- **Laba Deka**:
  - Worked on backend and flask, and implementing the security mechanism. Also worked together on the first four features, and in testing everything thoroughly.
  - Extra implementation : Rank System , Update Enabled
- **Gaurav Wadhwa**:
  - Worked on backend ,flask .Also worked together on the first four features, and making sure the database schema was modified as required.
  - Extra Implementation : Start an order, Add to current order