DATABASE MANAGEMENT SYSTEM
PROJECT FINAL DELIVERABLE

Team Members:

1. Harish Kumar Yandrapragada 001617473

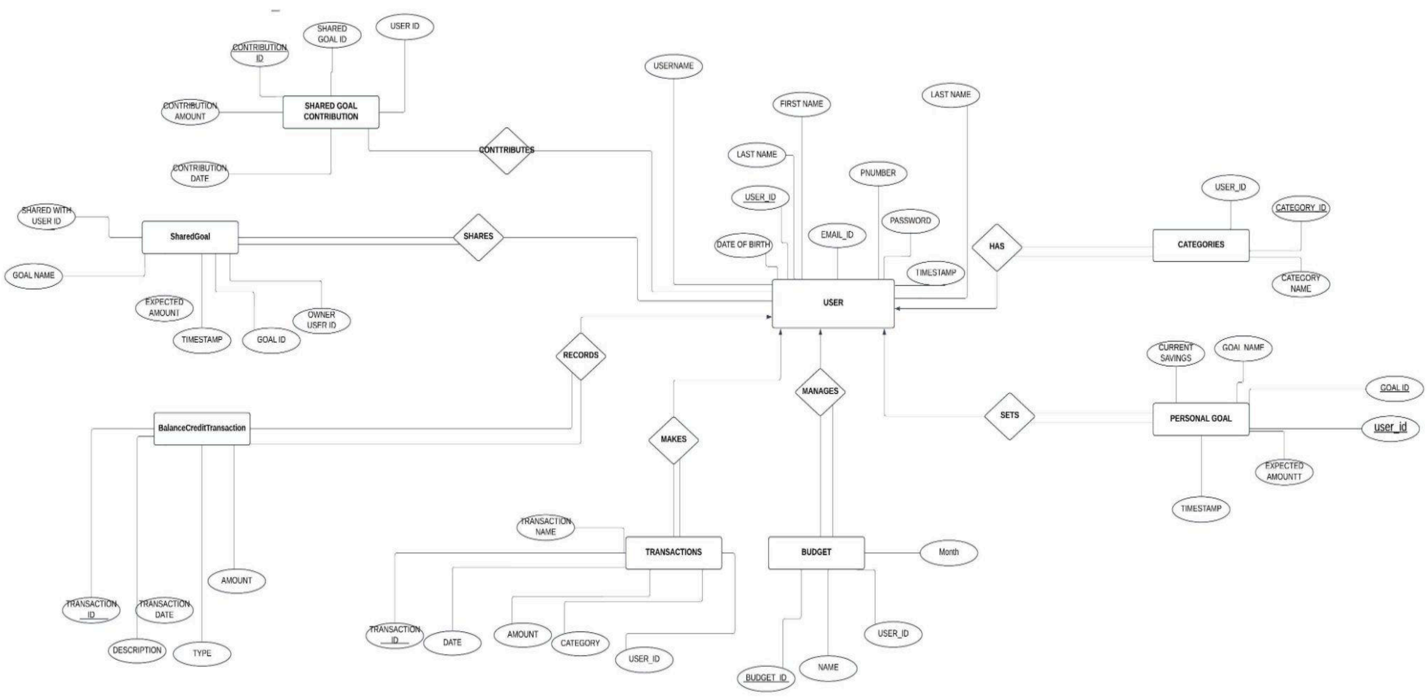
2. Sneha Bhavana 001609406

3. Abdul Shukur Shaik 001617464

4. Srinivasulu Bikki 001630905

1. Database design

E-R Diagram :



Tables :

# Users table

```
cursor.execute('''
    CREATE TABLE IF NOT EXISTS users (
        user_id INT AUTO_INCREMENT PRIMARY KEY,
        first_name VARCHAR(50) NOT NULL,
        last_name VARCHAR(50) NOT NULL,
        email_id VARCHAR(100) UNIQUE NOT NULL,
        phone_number VARCHAR(15) UNIQUE NOT NULL,
        username VARCHAR(100) UNIQUE NOT NULL,
        date_of_birth DATE NOT NULL,
        password VARCHAR(255) NOT NULL,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    );
''')
```

# Categories table

```
cursor.execute('''
    CREATE TABLE IF NOT EXISTS categories (
        category_id INT AUTO_INCREMENT PRIMARY KEY,
            user_id INT,

            category_name VARCHAR(100) NOT NULL,

            UNIQUE(user_id, category_name),

            FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE

        );

    ''')


    # Budgets table

    cursor.execute('''

        CREATE TABLE IF NOT EXISTS budgets (

            budget_id INT AUTO_INCREMENT PRIMARY KEY,

            user_id INT NOT NULL,

            year YEAR NOT NULL,

            month TINYINT NOT NULL, -- 1 for January, 2 for February, etc.

            category_id INT NOT NULL,

            budget DECIMAL(10, 2) NOT NULL,

            FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,

            FOREIGN KEY (category_id) REFERENCES categories(category_id) ON DELETE
CASCADE,

            UNIQUE(user_id, year, month, category_id)

        );
    ''')


    # Transactions table

    cursor.execute('''

        CREATE TABLE IF NOT EXISTS transactions (

            transaction_id INT AUTO_INCREMENT PRIMARY KEY,

            user_id INT NOT NULL,

            transaction_date DATE NOT NULL,

            amount DECIMAL(10, 2) NOT NULL,

            transaction_name VARCHAR(255) NOT NULL,

            category_id INT,

            type TINYINT NOT NULL,  -- 0: Cash, 1: Card, 2: Credit

            FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,

            FOREIGN KEY (category_id) REFERENCES categories(category_id) ON DELETE
SET NULL

        );

    ''')


# Balance Credit Transaction Table

    cursor.execute('''

        CREATE TABLE IF NOT EXISTS balance_credit_transactions (
            transaction_id INT AUTO_INCREMENT PRIMARY KEY,
            user_id INT NOT NULL,
            amount DECIMAL(10, 2) NOT NULL,
            description VARCHAR(255),
            type TINYINT NOT NULL,  -- 0: Balance, 1: Credit
            transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
        );
    ''')
```

#Personal Goals Table

```
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS personal_goals (
            goal_id INT AUTO_INCREMENT PRIMARY KEY,
            user_id INT NOT NULL,
            goal_name VARCHAR(255) NOT NULL,
            expected_amount DECIMAL(10, 2) NOT NULL,
            current_savings DECIMAL(10, 2) DEFAULT 0,
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
        );
    ''')
```

#Shared Goals Table

```
    cursor.execute('''
      CREATE TABLE IF NOT EXISTS shared_goals (
        goal_id INT AUTO_INCREMENT PRIMARY KEY,
        owner_user_id INT NOT NULL,
        shared_with_user_id INT NOT NULL,
        goal_name VARCHAR(255) NOT NULL,
        expected_amount DECIMAL(10, 2) NOT NULL,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (owner_user_id) REFERENCES users(user_id) ON DELETE
CASCADE,
        FOREIGN KEY (shared_with_user_id) REFERENCES users(user_id) ON DELETE
CASCADE
      );
    ''')
```

# Shared Goal Contribution Table

```
    cursor.execute('''
      CREATE TABLE IF NOT EXISTS shared_goal_contributions (
        contribution_id INT AUTO_INCREMENT PRIMARY KEY,
        shared_goal_id INT NOT NULL,

        user_id INT NOT NULL,

        contribution_amount DECIMAL(10, 2) NOT NULL,

        contribution_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

        FOREIGN KEY (shared_goal_id) REFERENCES shared_goals(goal_id) ON DELETE CASCADE,

        FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE

      );

    ''')
```

2. Database programming

Prerequisites

- **Python** installed on your system.
- **MySQL** installed and running locally.
- **pip** package manager.
- `requirements.txt` file provided (generated by `pip freeze`).
- **`Install streamlit`**

Step 1: Set Up the MySQL Database

1. **Start MySQL Server**: Ensure your MySQL server is running on `localhost`.
2. **Create Database**:Access the MySQL command-line interface or use a GUI tool.

Execute the following command to create the database:

```
CREATE DATABASE finance_app;
```

Step 2: Prepare Your Project

1. **Update Database Credentials in `db.py`**:

Replace the placeholders with your actual credentials:

```
DB_USER = 'root'

DB_PASSWORD = 'your_database_password'

DB_HOST = 'localhost'

DB_NAME = 'finance_app'
```

   - Ensure these match your MySQL user and database settings.

2. **Place `requirements.txt` in Project Directory**:

   Ensure the `requirements.txt` file is in the same directory as your `app.py`.

Step 3: Install streamlit

Command : pip install streamlit

Install Required Packages:

- Use the provided `requirements.txt` to install dependencies:

  ```
  pip install -r requirements.txt
  ```

Step 4: Run the Streamlit Application

Execute the Streamlit App: `streamlit run app.py`

1. **Access the Application**:

- Open your web browser and navigate to the URL shown in the terminal, typically http://localhost:8501.

Advanced Sql :

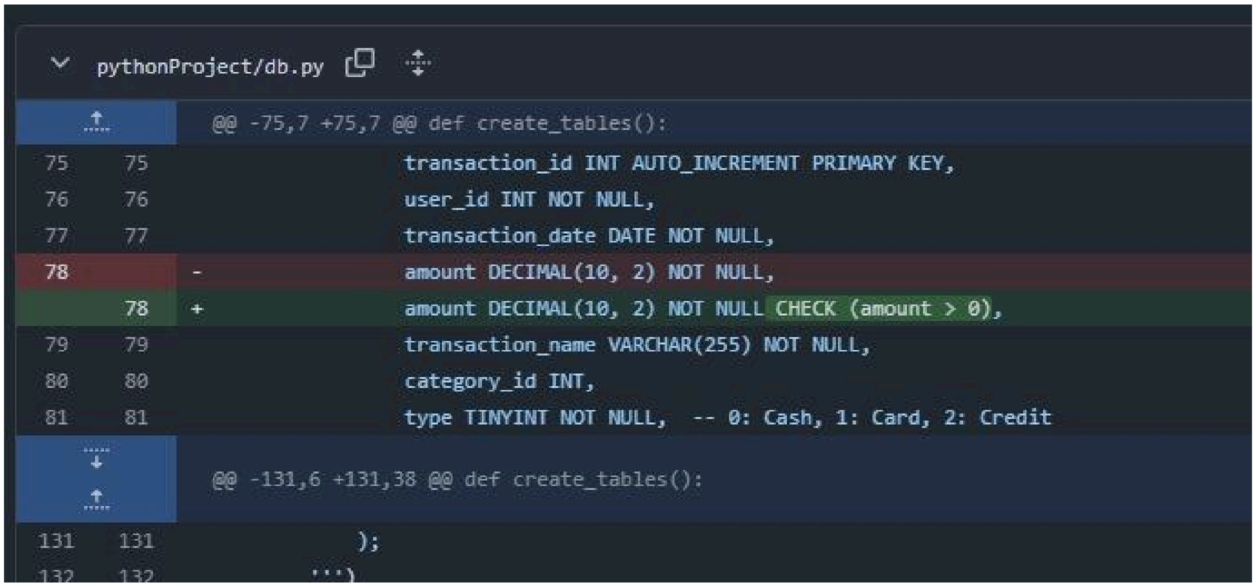
- **Updated `amount` field**: `amount DECIMAL(10, 2) NOT NULL CHECK (amount > 0)`.
- Added `check_contribution_before_insert` trigger for shared goals.
- Trigger ensures total contributions do not exceed the expected amount.
- Raises error if new contribution exceeds allowed limit using `SIGNAL SQLSTATE '45000'`.

CHECK (amount > 0),

SQL Query for Delimiter:

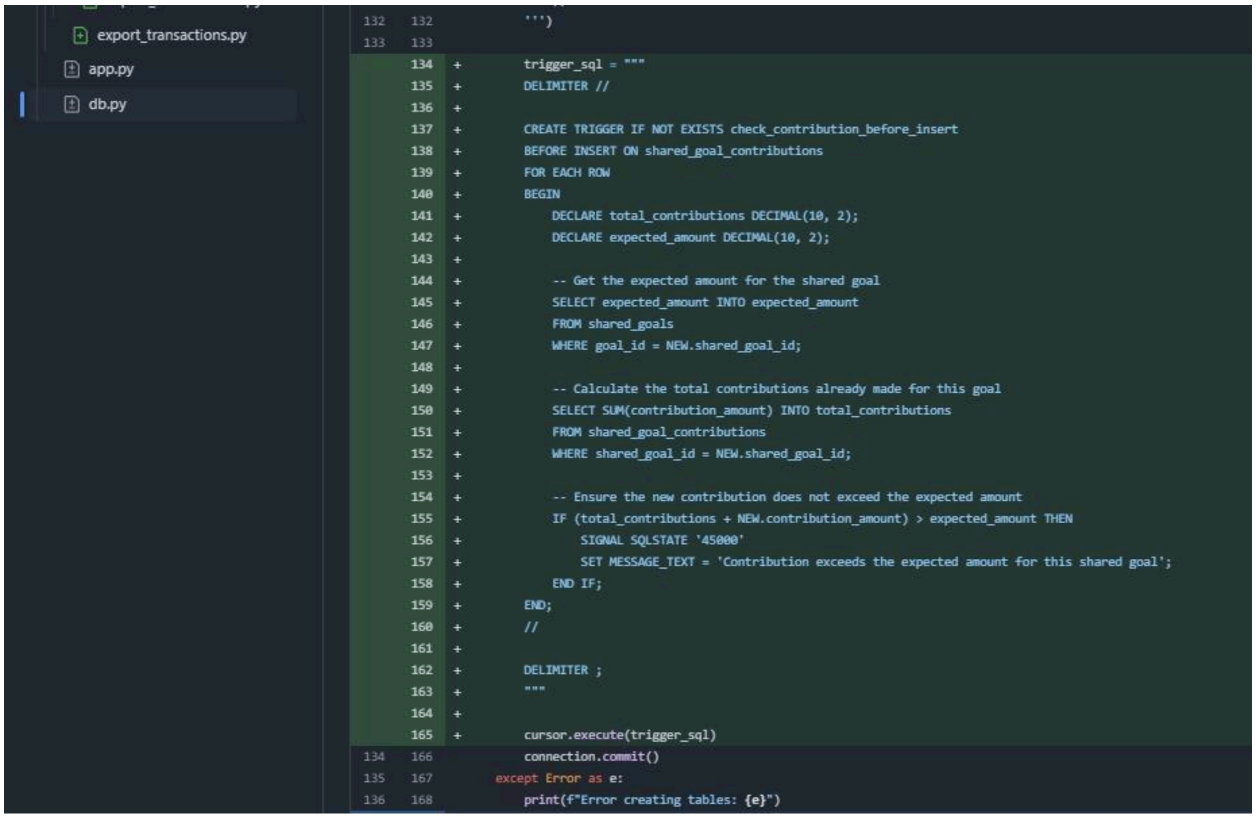
```
trigger_sql = """

DELIMITER //

CREATE TRIGGER IF NOT EXISTS check_contribution_before_insert

BEFORE INSERT ON shared_goal_contributions

FOR EACH ROW

BEGIN

    DECLARE total_contributions DECIMAL(10, 2);

    DECLARE expected_amount DECIMAL(10, 2);

    -- Get the expected amount for the shared goal

    SELECT expected_amount INTO expected_amount

    FROM shared_goals
```

```
        WHERE goal_id = NEW.shared_goal_id;

        -- Calculate the total contributions already made for this goal

        SELECT SUM(contribution_amount) INTO total_contributions

        FROM shared_goal_contributions

        WHERE shared_goal_id = NEW.shared_goal_id;

        -- Ensure the new contribution does not exceed the expected amount

        IF (total_contributions + NEW.contribution_amount) > expected_amount THEN

            SIGNAL SQLSTATE '45000'

            SET MESSAGE_TEXT = 'Contribution exceeds the expected amount for this
shared goal';

        END IF;

    END;

    //

    DELIMITER ;

  cursor.execute(trigger_sql)
```



pythonProject/db.py

```
@@ -75,7 +75,7 @@ def create_tables():
                transaction_id INT AUTO_INCREMENT PRIMARY KEY,
                user_id INT NOT NULL,
                transaction_date DATE NOT NULL,
-               amount DECIMAL(10, 2) NOT NULL,
+               amount DECIMAL(10, 2) NOT NULL CHECK (amount > 0),
                transaction_name VARCHAR(255) NOT NULL,
                category_id INT,
                type TINYINT NOT NULL,  -- 0: Cash, 1: Card, 2: Credit
@@ -131,6 +131,38 @@ def create_tables():
            );
        ''')
```



```
        ''')

+       trigger_sql = """
+       DELIMITER //
+
+       CREATE TRIGGER IF NOT EXISTS check_contribution_before_insert
+       BEFORE INSERT ON shared_goal_contributions
+       FOR EACH ROW
+       BEGIN
+           DECLARE total_contributions DECIMAL(10, 2);
+           DECLARE expected_amount DECIMAL(10, 2);
+
+           -- Get the expected amount for the shared goal
+           SELECT expected_amount INTO expected_amount
+           FROM shared_goals
+           WHERE goal_id = NEW.shared_goal_id;
+
+           -- Calculate the total contributions already made for this goal
+           SELECT SUM(contribution_amount) INTO total_contributions
+           FROM shared_goal_contributions
+           WHERE shared_goal_id = NEW.shared_goal_id;
+
+           -- Ensure the new contribution does not exceed the expected amount
+           IF (total_contributions + NEW.contribution_amount) > expected_amount THEN
+               SIGNAL SQLSTATE '45000'
+               SET MESSAGE_TEXT = 'Contribution exceeds the expected amount for this shared goal';
+           END IF;
+       END;
+       //
+
+       DELIMITER ;
+       """
+
+       cursor.execute(trigger_sql)
        connection.commit()
    except Error as e:
        print(f"Error creating tables: {e}")
```

3. **Describe the database security at the database level**

SQL Commands to Set Privileges:

Creating a New User:

CREATE USER 'finance_user'@'localhost' IDENTIFIED BY 'user_password';

Purpose: This command creates a new user in the MySQL database named finance_user.

Details:

'finance_user' is the username. This user will be used by application to connect to the database.

'localhost' means user can connect only from the same machine where the database is hosted. This is good for security, as it limits external access.

'user_password' is the password for this user. Should set'user_password' with a secure password of choice before deployment and set accordingly.

Use Case: This user will be used by your web application to interact with the database.

Granting Limited Privileges:

GRANT SELECT, INSERT, UPDATE ON finance_app.* TO 'finance_user'@'localhost';

Purpose: This command grants specific privileges to the user finance_user.

Privileges Granted:

SELECT: Allows finance_user to read data from the tables in the finance_app database.

INSERT: Allows finance_user to add new data to the tables.

UPDATE: Allows finance_user to modify existing records in the tables.

Scope:

ON finance_app.: The privileges are granted on all tables () within the database named finance_app.

Limited Access: finance_user cannot delete (DELETE) tables, alter the database structure, or grant permissions to other users. This ensures that the user can only perform actions required for the application to function without the ability to change the structure or delete data.

Applying Privileges:

FLUSH PRIVILEGES;

Purpose: This command ensures that any changes made to user privileges take effect immediately.

Without this command, MySQL might not apply changes to permissions until it is restarted or a similar event occurs.

4. **Describe the database security at the application level**

1. Credential Management

To enhance security, we avoid using the root user directly in the application. Instead, we create a restricted database user (`finance_user`) with limited privileges necessary for the application's operation.

Creating a Restricted User

```
-- Create a new user with a secure password

CREATE USER 'finance_user'@'localhost' IDENTIFIED BY
'your_secure_password';

-- Grant specific privileges to the user

GRANT SELECT, INSERT, UPDATE ON finance_app.* TO
'finance_user'@'localhost';

-- Apply the changes immediately
```

```
FLUSH PRIVILEGES;
```

Purpose:

- Limited Access: By granting only the necessary privileges, we adhere to the principle of least privilege, reducing the risk of unauthorized data manipulation.
- Security: Restricting the user to `localhost` prevents remote connections, minimizing potential attack vectors.

2. SQL Injection Prevention

We protect against SQL injection attacks by using parameterized queries throughout the application. This ensures that user inputs are properly escaped and cannot alter the structure of SQL commands.

Explanation:

- Parameterized Queries: The `%s` placeholder is used in the SQL command, and the variables are passed as a tuple. This ensures that the database driver handles escaping, preventing malicious input from altering the SQL command.
- Security Benefit: Protects the application from SQL injection attacks by not concatenating user inputs directly into SQL statements.

3. Input Validation

We validate and sanitize all user inputs before processing them, reducing the risk of malicious data being entered into the system.

Explanation:

- **Type Checking:** Ensures that inputs are of the expected data type.
- **Value Constraints:** Checks that numeric inputs are within acceptable ranges.
- **Feedback:** Provides immediate feedback to users on invalid inputs.

4. Password Hashing

To protect user credentials, we hash passwords before storing them in the database. This ensures that even if the database is compromised, plaintext passwords are not exposed.

- hashlib