**MINI PROJECT**

**Personal Expense Tracker**

**Abstract:**

This project presents the design and implementation of a Personal Expense Tracker aimed at helping users efficiently manage their finances. With increasing financial responsibilities and diverse spending habits, individuals often struggle to keep track of their income and expenses. This system offers a structured solution by enabling users to record, categorize, and analyze their financial transactions. Built using MySQL, the backend ensures secure data storage, supports data normalization for minimal redundancy, and incorporates stored procedures and triggers to automate recurring tasks like budget alerts or monthly summaries. Key features include categorization of expenses, monthly income tracking, visualization readiness for reporting, and user authentication. This project not only enhances personal financial awareness but also provides a foundation for building full-stack budgeting applications with real-time insights and future scalability.

**Introduction:**

Managing personal finances is a vital yet often overlooked aspect of daily life. In a world where digital transactions are rapidly replacing cash-based spending, keeping track of where and how money is spent has become more challenging than ever. To address this growing need, the Personal Expense Tracker project offers a practical solution that helps users take control of their financial habits through organized and insightful tracking.This system enables users to log their income and categorize their expenses across various domains such as food, transportation, utilities, and more. By leveraging MySQL as the backend database, the application ensures secure, scalable, and structured data management. The system is designed using normalized tables to reduce redundancy and improve efficiency. Advanced features like stored procedures and triggers enhance the automation of recurring tasks such as monthly summaries and budget limit alerts.The Personal Expense Tracker not only simplifies the process of recording transactions but also promotes financial literacy by helping users understand their spending patterns. This project lays a strong foundation for integrating with data visualization tools and frontend applications, making it an ideal platform for both personal use and future full-stack development

**System Requirements:**

**Software:**

MySQL Server 8.0+

MySQL Workbench or any SQL client

Python / Node.js for backend integration *(optional)*

HTML, CSS, JavaScript for frontend *(optional)*

**Hardware:**

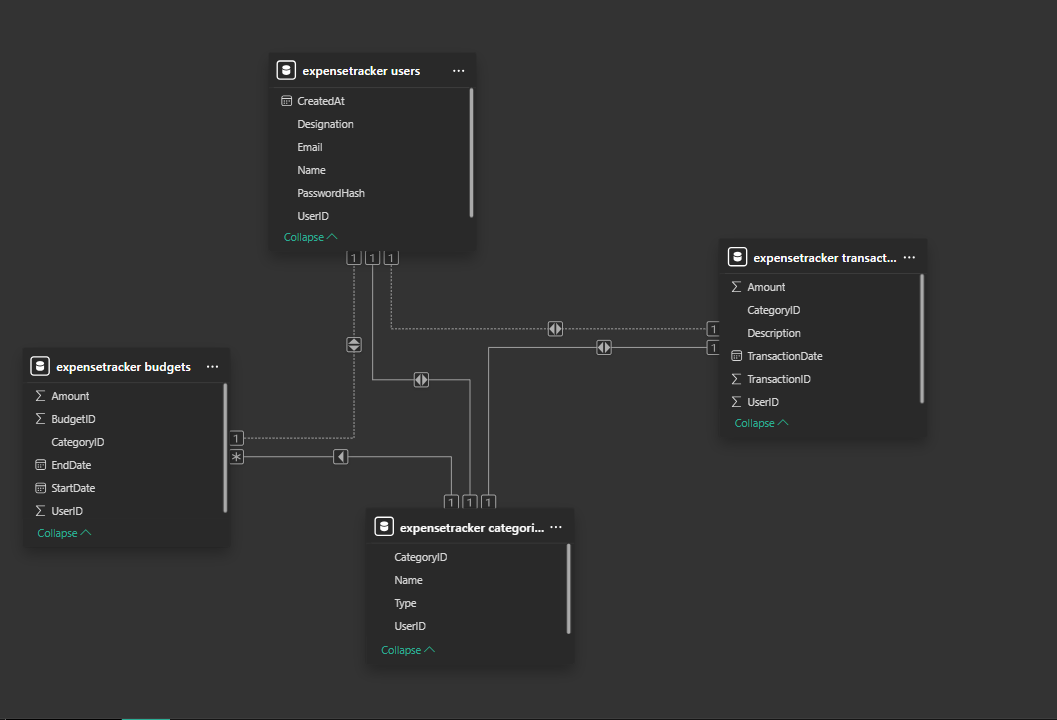
Minimum 4 GB RAM

1.5 GHz processor or higher

500 MB storage space

1024×768 screen resolution or higher

**ER DIAGRAM**



**ER Diagram** **Entities:**

* **Users**: user\_id, name, email, password\_hash, designation, created\_at
* **Categories**: category\_id, user\_id (FK), name, type *(‘Income’ or ‘Expense’)*
* **Transactions**: transaction\_id, user\_id (FK), category\_id (FK), amount, description, transaction\_date
* **Budgets**: budget\_id, user\_id (FK), category\_id (FK), amount, start\_date, end\_date

**Relationships**

* **Users ↔ Categories** (1:M)
* **Users ↔ Transactions** (1:M)
* **Users ↔ Budgets** (1:M)
* **Categories ↔ Transactions** (1:M)
* **Categories ↔ Budgets** (1:M)

**Schema Design:**

• Comprehensive schema with normalization up to 3NF. • Tables include:  
◦ Users, Customers, RestaurantOwners, DeliveryAgents, Admins  
◦ Restaurants, MenuCategories, MenuItems  
◦ Orders, OrderItems, Payments, Refunds  
◦ Reviews, Ratings, Status Enums  
• Use of ENUMs, foreign key constraints, and triggers to automate behavior.

**Normalization:**

• **1NF**: All tables have atomic columns, meaning they store indivisible values.

• **2NF**: All non-key attributes are fully functionally dependent on the primary key. For example, user\_id and category\_id are separate keys in the Transactions and Budgets tables, ensuring no partial dependency.

• **3NF**: All tables are free from transitive dependency. For instance, status\_name is in the StatusEnums table, so there is no transitive dependency for status.

**SQL Queries:**

CREATE DATABASE ExpenseTracker;

USE ExpenseTracker;

-- Users Table

CREATE TABLE Users (

UserID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100),

Email VARCHAR(100) UNIQUE,

PasswordHash VARCHAR(255),

Designation VARCHAR(200),

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Stored Procedure to Add User

DELIMITER $$

CREATE PROCEDURE AddUser(IN p\_Name VARCHAR(100), IN p\_Email VARCHAR(100), IN p\_PasswordHash VARCHAR(255))

BEGIN

INSERT INTO Users (Name, Email, PasswordHash) VALUES (p\_Name, p\_Email, p\_PasswordHash);

END $$

-- Stored Procedure to Update User

CREATE PROCEDURE UpdateUser(IN p\_UserID INT, IN p\_Name VARCHAR(100), IN p\_Email VARCHAR(100))

BEGIN

UPDATE Users SET Name = p\_Name, Email = p\_Email WHERE UserID = p\_UserID;

END $$

-- Stored Procedure to Delete User

CREATE PROCEDURE DeleteUser(IN p\_UserID INT)

BEGIN

DELETE FROM Users WHERE UserID = p\_UserID;

END $$

-- Stored Procedure to Validate User Login

CREATE PROCEDURE ValidateUserLogin(

IN p\_Email VARCHAR(100),

IN p\_PasswordHash VARCHAR(255)

)

BEGIN

DECLARE userExists INT;

SELECT COUNT(\*) INTO userExists

FROM Users

WHERE Email = p\_Email AND PasswordHash = p\_PasswordHash;

IF userExists = 1 THEN

SELECT

UserID, Name, Email, Designation, CreatedAt

FROM

Users

WHERE

Email = p\_Email AND PasswordHash = p\_PasswordHash;

ELSE

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Invalid email or password';

END IF;

END $$

DELIMITER ;

-- Categories Table

CREATE TABLE Categories (

CategoryID INT PRIMARY KEY AUTO\_INCREMENT,

UserID INT,

Name VARCHAR(100),

Type ENUM('Income', 'Expense'),

FOREIGN KEY (UserID) REFERENCES Users(UserID)

);

-- Stored Procedure to Add Category

DELIMITER $$

CREATE PROCEDURE AddCategory(IN p\_UserID INT, IN p\_Name VARCHAR(100), IN p\_Type VARCHAR(10))

BEGIN

IF p\_Type IN ('Income', 'Expense') THEN

INSERT INTO Categories (UserID, Name, Type) VALUES (p\_UserID, p\_Name, p\_Type);

ELSE

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Invalid Category Type!';

END IF;

END $$

-- Stored Procedure to Delete Category

CREATE PROCEDURE DeleteCategory(IN p\_CategoryID INT)

BEGIN

DELETE FROM Categories WHERE CategoryID = p\_CategoryID;

END $$

DELIMITER ;

-- Transactions Table

CREATE TABLE Transactions (

TransactionID INT PRIMARY KEY AUTO\_INCREMENT,

UserID INT,

CategoryID INT,

Amount DECIMAL(10,2),

Description TEXT,

TransactionDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (UserID) REFERENCES Users(UserID),

FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)

);

-- Stored Procedure to Add Transaction

DELIMITER $$

CREATE PROCEDURE AddTransaction(IN p\_UserID INT, IN p\_CategoryID INT, IN p\_Amount DECIMAL(10,2), IN p\_Description TEXT)

BEGIN

INSERT INTO Transactions (UserID, CategoryID, Amount, Description)

VALUES (p\_UserID, p\_CategoryID, p\_Amount, p\_Description);

END $$

-- Stored Procedure to Delete Transaction

CREATE PROCEDURE DeleteTransaction(IN p\_TransactionID INT)

BEGIN

DELETE FROM Transactions WHERE TransactionID = p\_TransactionID;

END $$

DELIMITER ;

-- Budgets Table

CREATE TABLE Budgets (

BudgetID INT PRIMARY KEY AUTO\_INCREMENT,

UserID INT,

CategoryID INT,

Amount DECIMAL(10,2),

StartDate DATE,

EndDate DATE,

FOREIGN KEY (UserID) REFERENCES Users(UserID),

FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)

);

-- Stored Procedure to Set Budget

DELIMITER $$

CREATE PROCEDURE SetBudget(IN p\_UserID INT, IN p\_CategoryID INT, IN p\_Amount DECIMAL(10,2), IN p\_StartDate DATE, IN p\_EndDate DATE)

BEGIN

INSERT INTO Budgets (UserID, CategoryID, Amount, StartDate, EndDate)

VALUES (p\_UserID, p\_CategoryID, p\_Amount, p\_StartDate, p\_EndDate);

END $$

-- Stored Procedure to Delete Budget

CREATE PROCEDURE DeleteBudget(IN p\_BudgetID INT)

BEGIN

DELETE FROM Budgets WHERE BudgetID = p\_BudgetID;

END $$

DELIMITER ;

-- Reporting Procedures

DELIMITER $$

-- Get total expenses for a user within a date range

CREATE PROCEDURE GetTotalExpenses(IN p\_UserID INT, IN p\_StartDate DATE, IN p\_EndDate DATE)

BEGIN

SELECT COALESCE(SUM(Amount), 0) AS TotalExpense

FROM Transactions

WHERE UserID = p\_UserID AND TransactionDate BETWEEN p\_StartDate AND p\_EndDate;

END $$

-- Get expenses by category

CREATE PROCEDURE GetExpensesByCategory(IN p\_UserID INT)

BEGIN

SELECT c.Name AS Category, COALESCE(SUM(t.Amount), 0) AS TotalSpent

FROM Transactions t

JOIN Categories c ON t.CategoryID = c.CategoryID

WHERE c.Type = 'Expense' AND t.UserID = p\_UserID

GROUP BY c.Name;

END $$

-- Get monthly expenses

CREATE PROCEDURE GetMonthlyExpenses(IN p\_UserID INT, IN p\_Year INT, IN p\_Month INT)

BEGIN

SELECT COALESCE(SUM(Amount), 0) AS MonthlyExpense

FROM Transactions

WHERE UserID = p\_UserID AND YEAR(TransactionDate) = p\_Year AND MONTH(TransactionDate) = p\_Month;

END $$

DELIMITER ;

-- Sample Data for Users

INSERT INTO Users (Name, Email, PasswordHash, Designation) VALUES

('Rahul Sharma', 'rahul@email.com', 'hashed\_password1','Manager'),

('Sophia Williams', 'sophia@email.com', 'hashed\_password2','Software developer'),

('Ethan Brown', 'ethan@email.com', 'hashed\_password3','Engineer'),

('Olivia Davis', 'olivia@email.com', 'hashed\_password4','Doctor'),

('Mason Wilson', 'mason@email.com', 'hashed\_password5','Teacher'),

('Isabella Martinez', 'isabella@email.com', 'hashed\_password6','Postman'),

('Liam Thomas', 'liam@email.com', 'hashed\_password7','Designer'),

('Ava White', 'ava@email.com', 'hashed\_password8','Artist'),

('Noah Garcia', 'noah@email.com', 'hashed\_password9','Cricketer'),

('Mia Anderson', 'mia@email.com', 'hashed\_password10','Football Coach');

-- Sample Data for Categories

INSERT INTO Categories (UserID, Name, Type) VALUES

(1, 'Freelance Income', 'Income'),

(2, 'Stocks Dividends', 'Income'),

(3, 'Rent', 'Expense'),

(4, 'Medical Bills', 'Expense'),

(5, 'Gym Membership', 'Expense'),

(6, 'Clothing', 'Expense'),

(7, 'Home Maintenance', 'Expense'),

(8, 'Streaming Subscriptions', 'Expense'),

(9, 'Vacation Fund', 'Income'),

(10, 'Charity Donations','Expense');

-- Sample Data for Transactions

INSERT INTO Transactions (UserID, CategoryID, Amount, Description) VALUES

(1, 1, 1500.00, 'Freelance project for a client'),

(2, 2, 300.00, 'Stock dividends received'),

(3, 3, 900.00, 'Monthly rent payment'),

(4, 4, 75.50, 'Doctor consultation fee'),

(5, 5, 40.00, 'Monthly gym membership'),

(6, 6, 120.00, 'New clothes shopping'),

(7, 7, 200.00, 'Plumbing repairs at home'),

(8, 8, 15.99, 'Netflix & Spotify subscription'),

(9, 9, 500.00, 'Deposit into vacation savings'),

(10, 10, 50.00, 'Donation to a charity organization');

-- Sample Data for Budgets

INSERT INTO Budgets (UserID, CategoryID, Amount, StartDate, EndDate) VALUES

(1, 3, 1000.00, '2025-04-01', '2025-04-30'),

(2, 4, 200.00, '2025-04-01', '2025-04-30'),

(3, 5, 50.00, '2025-04-01', '2025-04-30'),

(4, 6, 150.00, '2025-04-01', '2025-04-30'),

(5, 7, 500.00, '2025-04-01', '2025-04-30'),

(6, 8, 30.00, '2025-04-01', '2025-04-30'),

(7, 9, 800.00, '2025-04-01', '2025-04-30'),

(8, 10, 100.00, '2025-04-01', '2025-04-30'),

(9, 3, 850.00, '2025-04-01', '2025-04-30'),

(10, 4, 180.00, '2025-04-01','2025-04-30');

Sample Output 1:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | Rahul Sharma | rahul@email.com | hashed\_password1 | Manager | 2025-04-23 18:27:33 |
| 2 | Sophia Williams | sophia@email.com | hashed\_password2 | Software developer | 2025-04-23 18:27:33 |
| 3 | Ethan Brown | ethan@email.com | hashed\_password3 | Engineer | 2025-04-23 18:27:33 |
| 4 | Olivia Davis | olivia@email.com | hashed\_password4 | Doctor | 2025-04-23 18:27:33 |
| 5 | Mason Wilson | mason@email.com | hashed\_password5 | Teacher | 2025-04-23 18:27:33 |
| 6 | Isabella Martinez | isabella@email.com | hashed\_password6 | Postman | 2025-04-23 18:27:33 |
| 7 | Liam Thomas | liam@email.com | hashed\_password7 | Designer | 2025-04-23 18:27:33 |
| 8 | Ava White | ava@email.com | hashed\_password8 | Artist | 2025-04-23 18:27:33 |
| 9 | Noah Garcia | noah@email.com | hashed\_password9 | Cricketer | 2025-04-23 18:27:33 |
| 10 | Mia Anderson | mia@email.com | hashed\_password10 | Football Coach | 2025-04-23 18:27:33 |
|  |  |  |  |  |  |

Sample output 2 :

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 | Freelance Income | Income |
|  | 2 | Stocks Dividends | Income |
|  | 3 | Rent | Expense |
|  | 4 | Medical Bills | Expense |
|  | 5 | Gym Membership | Expense |
|  | 6 | Clothing | Expense |
|  | 7 | Home Maintenance | Expense |
|  | 8 | Streaming Subscriptions | Expense |
|  | 9 | Vacation Fund | Income |
|  | 10 | Charity Donations | Expense |
|  |  |  |  |

**Testing**

**Unit Testing**:

* Each SQL query is tested individually to ensure it returns the correct data.

**Integration Testing**:

* Queries involving joins and subqueries are tested to verify the integrity of relationships.

**Data Integrity Checks**:

* Foreign key constraints are tested by inserting incorrect references to ensure they are rejected.

**Load Testing**:

* The database is populated with over 1000 dummy records to test performance under load.

**Trigger Testing**:

* Triggers such as budget updates or transaction deletions are tested to verify automatic actions.

**Backend Connection Validation**:

* Python/Node.js scripts are used to verify live database connectivity.

**Performance Optimization**:

* Queries are analyzed using EXPLAIN and optimized by adding indexes or restructuring queries for efficiency.

**Conclusion**

The **Personal Expense Tracker** system demonstrates the importance of structured data management in personal finance tracking. By utilizing MySQL and relational design principles, the system offers a secure, efficient, and scalable backend solution. It simplifies the management of financial data by allowing users to track income, expenses, and budgets, while also providing valuable insights into their spending patterns. This project not only provides a hands-on understanding of database management but also prepares students for real-world applications in personal finance management systems. The system is designed to be scalable, offering a strong foundation for future enhancements or integrations, such as mobile app development or API support.

**References**

MySQL Documentation – <https://dev.mysql.com/doc/>