EXPENSE TRACKER APP

DATABASE SCHEMA DESIGN AND IMPLEMENTATION

1.INTRODUCTION

The Expense Tracker App is a comprehensive tool designed to assist users in effectively managing their personal finances. By tracking daily expenses, monitoring budgets, and organizing financial reminders, this application plays a key role in promoting financial awareness and responsibility. This documentation outlines the core database design, implementation, and querying strategies used to support the application's functionality. The aim is to provide a robust and normalized structure that ensures data integrity, supports complex queries, and enhances user experience.

2.DATABASE SCHEMA DESIGN

The database for the Expense Tracker App has been carefully designed using the principles of normalization. The schema ensures minimal redundancy, consistent data, and supports complex relationships through the use of primary and foreign keys.

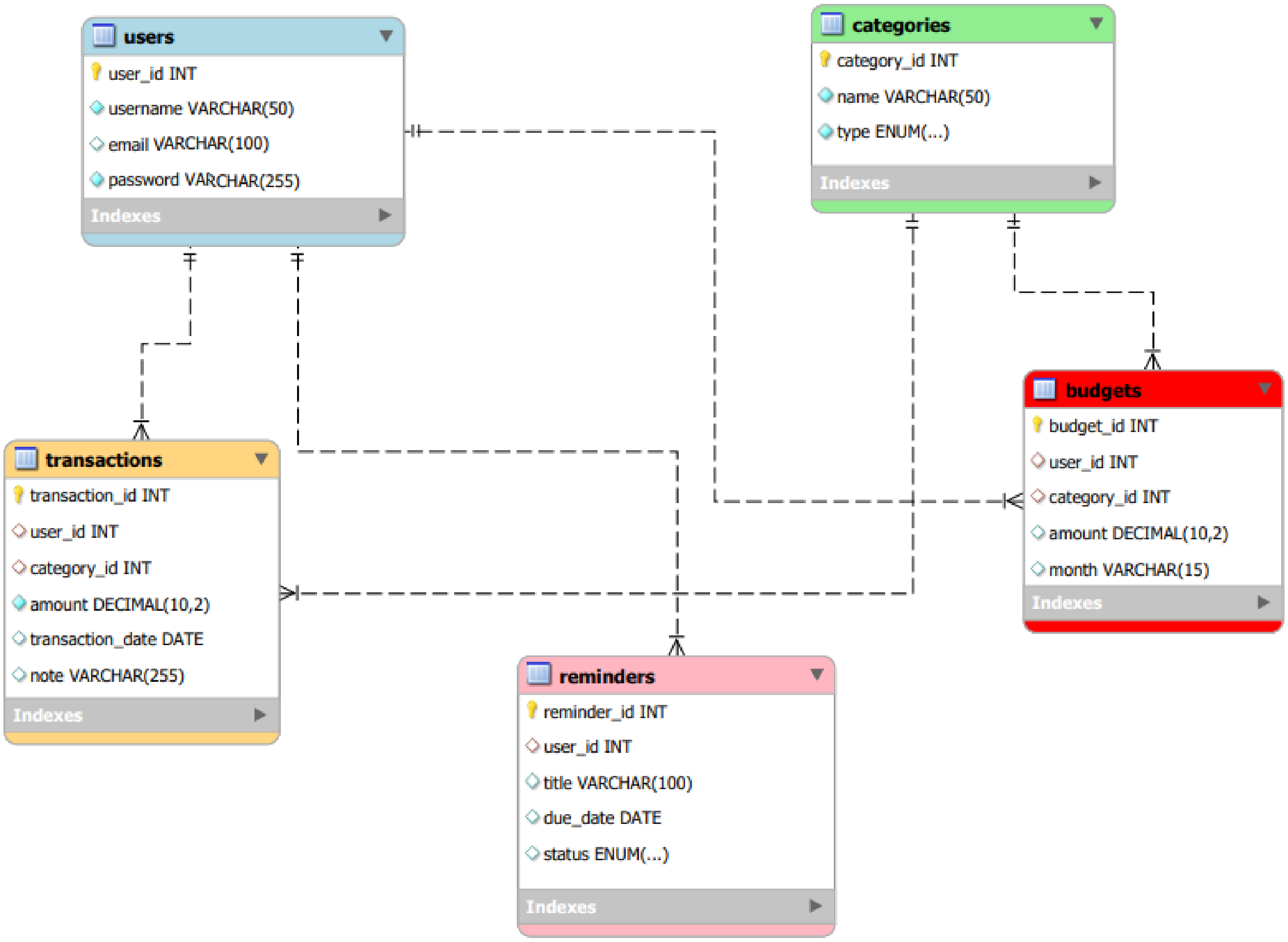
2.1 TABLES AND FIELDS

- **Users Table**
    - user_id (Primary Key)
    - username
    - email
    - password
- **Categories Table**
    - category_id (Primary Key)
    - name
    - type (income/expense)
- **Transactions Table**
    - transaction_id (Primary Key)
    - user_id (Foreign Key)
    - category_id (Foreign Key)
    - amount
    - transaction_date
    - note
- **Budgets Table**
    - budget_id (Primary Key)
    - user_id (Foreign Key)
    - category_id (Foreign Key)
    - amount
    - month

- **Reminders Table**
  - reminder_id (Primary Key)
  - user_id (Foreign Key)
  - title
  - due_date
  - status

2.2 ER DIAGRAM

The ER Diagram illustrates relationships among the five main tables. Foreign key relationships link users to transactions, budgets, and reminders, while categories are connected to both transactions and budgets. This design supports one-to-many relationships and maintains data consistency across tables.



3.DATABASE IMPLEMENTATION (MySQL)

The database was implemented in MySQL. Tables were created using appropriate data types and constraints:

```
CREATE DATABASE expense_tracker;
USE expense_tracker;

CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    email VARCHAR(100),
    password VARCHAR(255) NOT NULL
);

CREATE TABLE categories (
    category_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    type ENUM('income', 'expense') NOT NULL
);

CREATE TABLE transactions (
    transaction_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    category_id INT,
    amount DECIMAL(10,2) NOT NULL,
    transaction_date DATE,
    note VARCHAR(255),
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    FOREIGN KEY (category_id) REFERENCES categories(category_id)
);

CREATE TABLE budgets (
    budget_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    category_id INT,
    amount DECIMAL(10,2),
    month VARCHAR(15),
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    FOREIGN KEY (category_id) REFERENCES categories(category_id)
);

CREATE TABLE reminders (
    reminder_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    title VARCHAR(100),
    due_date DATE,
    status ENUM('pending', 'paid') DEFAULT 'pending',
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

4.SAMPLE DATA IMPLEMENTATION

Sample data was inserted into each table to simulate real-world use:

```
-- Users
INSERT INTO users (username, email, password) VALUES
('pavitra', 'pavitra@email.com', 'pass123'),
('john', 'john@email.com', 'john456'),
('ram', 'ram@email.com', 'ram789'),
('sita', 'sita@email.com', 'sita123'),
('kiran', 'kiran@email.com', 'kiran987');

-- Categories
INSERT INTO categories (name, type) VALUES
('Salary', 'income'),
('Freelance', 'income'),
('Groceries', 'expense'),
('Rent', 'expense'),
('Entertainment', 'expense');

-- Transactions
INSERT INTO transactions (user_id, category_id, amount, transaction_date, note) VALUES
(1, 1, 50000, '2025-07-01', 'Monthly salary'),
(1, 3, 2500, '2025-07-02', 'Groceries shopping'),
(2, 4, 8000, '2025-07-03', 'Paid rent'),
(3, 2, 10000, '2025-07-04', 'Freelance project'),
(4, 5, 1200, '2025-07-05', 'Movie night');

-- Budgets
INSERT INTO budgets (user_id, category_id, amount, month) VALUES
(1, 3, 3000, 'July'),
(1, 4, 8000, 'July'),
(2, 5, 2000, 'July'),
(3, 3, 2500, 'July'),
(4, 5, 1500, 'July');

-- Reminders
INSERT INTO reminders (user_id, title, due_date, status) VALUES
(1, 'Pay electricity bill', '2025-07-10', 'pending'),
(2, 'Recharge mobile', '2025-07-11', 'paid'),
(3, 'Pay water bill', '2025-07-15', 'pending'),
(4, 'Pay Wi-Fi bill', '2025-07-18', 'pending'),
(5, 'Pay credit card', '2025-07-20', 'paid');
```

5.DATA MANIPULATION AND QUERYING

A. DML Operations

```sql
-- Insert a new transaction
INSERT INTO transactions (user_id, category_id, amount, transaction_date, note)
VALUES (1, 3, 1800, '2025-07-06', 'Snacks and supplies');

-- Update budget amount
UPDATE budgets
SET amount = 3500
WHERE user_id = 1 AND category_id = 3;

-- Delete a reminder
DELETE FROM reminders
WHERE reminder_id = 5;
```

B. SELECT Queries

```sql
-- 1. All transactions with username and category
SELECT
    t.transaction_id,
    u.username,
    c.name AS category,
    t.amount,
    t.transaction_date,
    t.note
FROM transactions t
JOIN users u ON t.user_id = u.user_id
JOIN categories c ON t.category_id = c.category_id;

-- 2. Total expenses per user
SELECT
    u.username,
    SUM(t.amount) AS total_expense
FROM transactions t
JOIN users u ON t.user_id = u.user_id
JOIN categories c ON t.category_id = c.category_id
WHERE c.type = 'expense'
GROUP BY u.username;

-- 3. List all reminders by due date
SELECT * FROM reminders
ORDER BY due_date ASC;

-- 4. Budgets for July
SELECT * FROM budgets
WHERE month = 'July';
```

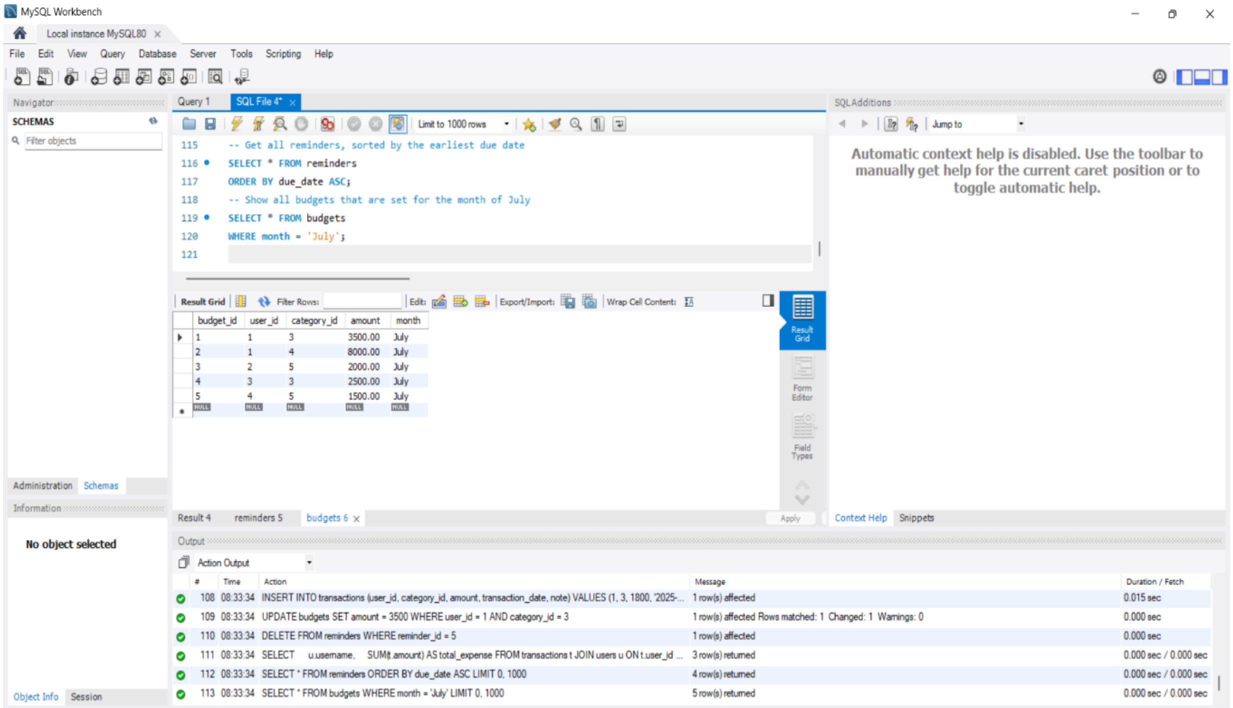
```
-- 5. Transactions above ₹5000
SELECT * FROM transactions
WHERE amount > 5000;
```

C. Complex Queries

```
-- 1. Users who spent more than ₹8000
SELECT username FROM users
WHERE user_id IN (
    SELECT user_id FROM transactions
    GROUP BY user_id
    HAVING SUM(amount) > 8000
);

-- 2. Category with highest total transaction amount
SELECT name, total
FROM (
    SELECT c.name, SUM(t.amount) AS total
    FROM transactions t
    JOIN categories c ON t.category_id = c.category_id
    GROUP BY c.name
) AS totals
ORDER BY total DESC
LIMIT 1;
```

6.SCREENSHOTS OF OUTPUTS

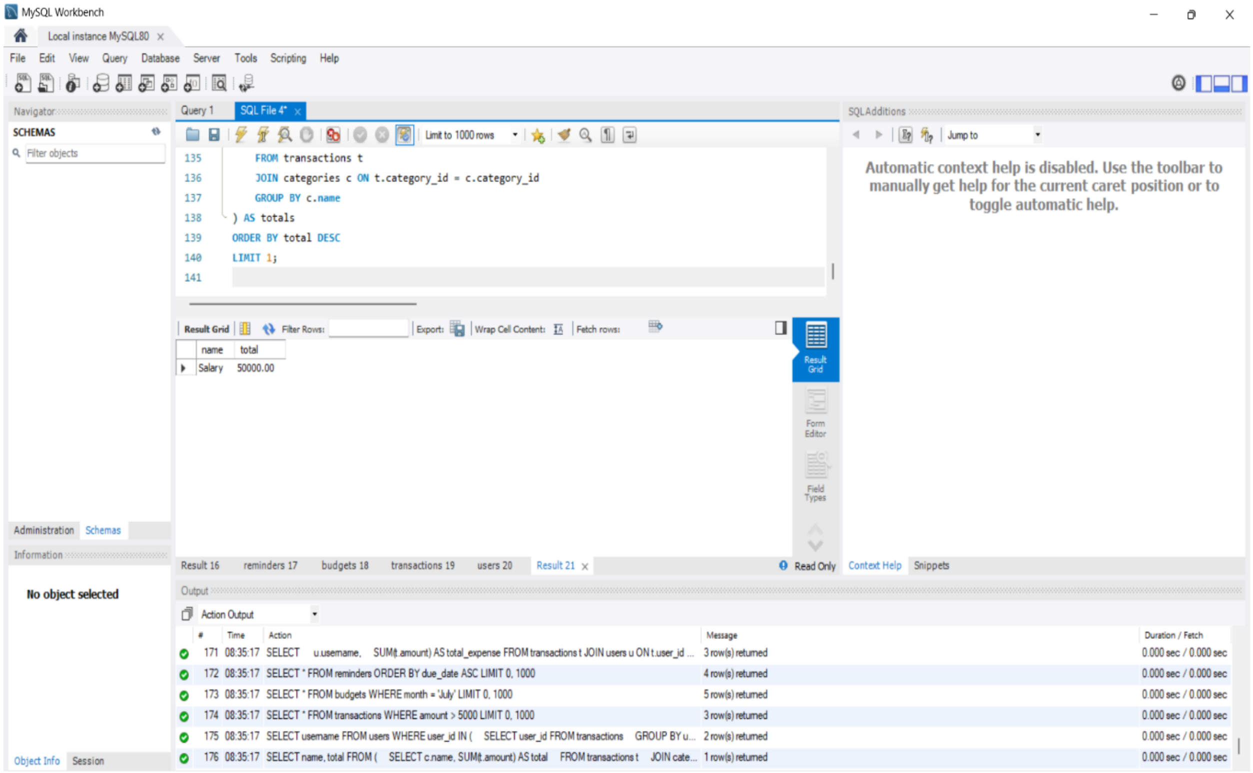


```
-- Get all reminders, sorted by the earliest due date
SELECT * FROM reminders
ORDER BY due_date ASC;
-- Show all budgets that are set for the month of July
SELECT * FROM budgets
WHERE month = 'July';
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

budget_id	user_id	category_id	amount	month
1	1	3	3500.00	July
2	1	4	8000.00	July
3	2	5	2000.00	July
4	3	3	2500.00	July
5	4	5	1500.00	July
NULL	NULL	NULL	NULL	NULL

```
#    Time      Action                                                                                       Message                                                   Duration / Fetch
108  08:33:34  INSERT INTO transactions (user_id, category_id, amount, transaction_date, note) VALUES (1, 3, 1800, '2025-...  1 row(s) affected                          0.015 sec
109  08:33:34  UPDATE budgets SET amount = 3500 WHERE user_id = 1 AND category_id = 3                           1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0  0.000 sec
110  08:33:34  DELETE FROM reminders WHERE reminder_id = 5                                                      1 row(s) affected                          0.000 sec
111  08:33:34  SELECT     u.username,     SUM(t.amount) AS total_expense FROM transactions t JOIN users u ON t.user_id ...  3 row(s) returned                0.000 sec / 0.000 sec
112  08:33:34  SELECT * FROM reminders ORDER BY due_date ASC LIMIT 0, 1000                                      4 row(s) returned                          0.000 sec / 0.000 sec
113  08:33:34  SELECT * FROM budgets WHERE month = 'July' LIMIT 0, 1000                                        5 row(s) returned                          0.000 sec / 0.000 sec
```



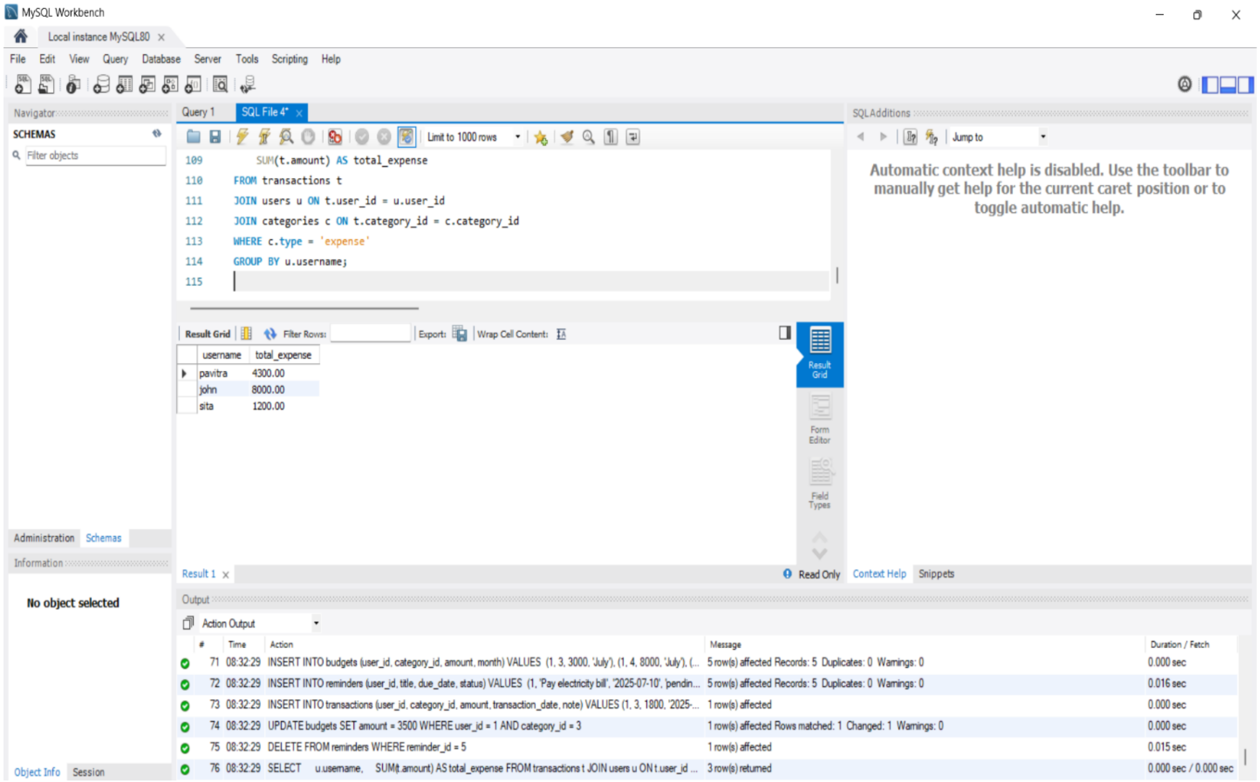
```
        FROM transactions t
        JOIN categories c ON t.category_id = c.category_id
        GROUP BY c.name
    ) AS totals
    ORDER BY total DESC
    LIMIT 1;
```

	name	total
▸	Salary	50000.00

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

No object selected

	#	Time	Action	Message	Duration / Fetch
✓	171	08:35:17	SELECT u.usemame, SUM(t.amount) AS total_expense FROM transactions t JOIN users u ON t.user_id ...	3 row(s) returned	0.000 sec / 0.000 sec
✓	172	08:35:17	SELECT * FROM reminders ORDER BY due_date ASC LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec
✓	173	08:35:17	SELECT * FROM budgets WHERE month = 'July' LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
✓	174	08:35:17	SELECT * FROM transactions WHERE amount > 5000 LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
✓	175	08:35:17	SELECT usemame FROM users WHERE user_id IN (SELECT user_id FROM transactions GROUP BY u...	2 row(s) returned	0.000 sec / 0.000 sec
✓	176	08:35:17	SELECT name, total FROM (SELECT c.name, SUM(t.amount) AS total FROM transactions t JOIN cate...	1 row(s) returned	0.000 sec / 0.000 sec



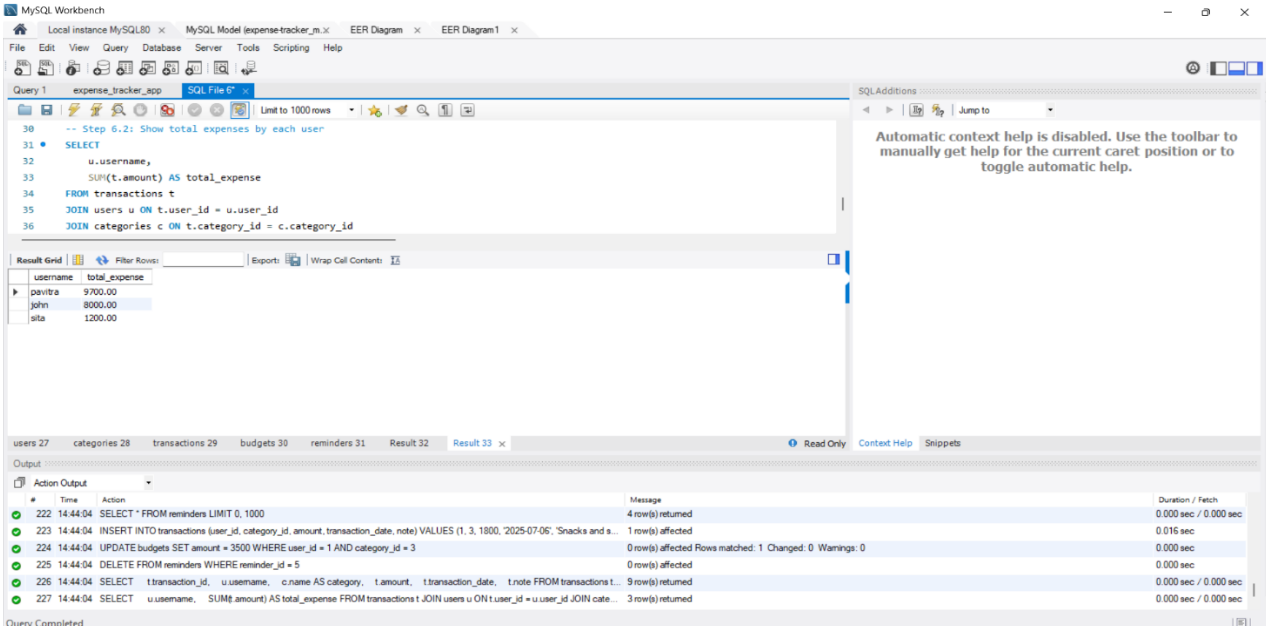
```
        SUM(t.amount) AS total_expense
    FROM transactions t
    JOIN users u ON t.user_id = u.user_id
    JOIN categories c ON t.category_id = c.category_id
    WHERE c.type = 'expense'
    GROUP BY u.username;
```

	username	total_expense
▸	pavitra	4300.00
	john	8000.00
	sita	1200.00

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

No object selected

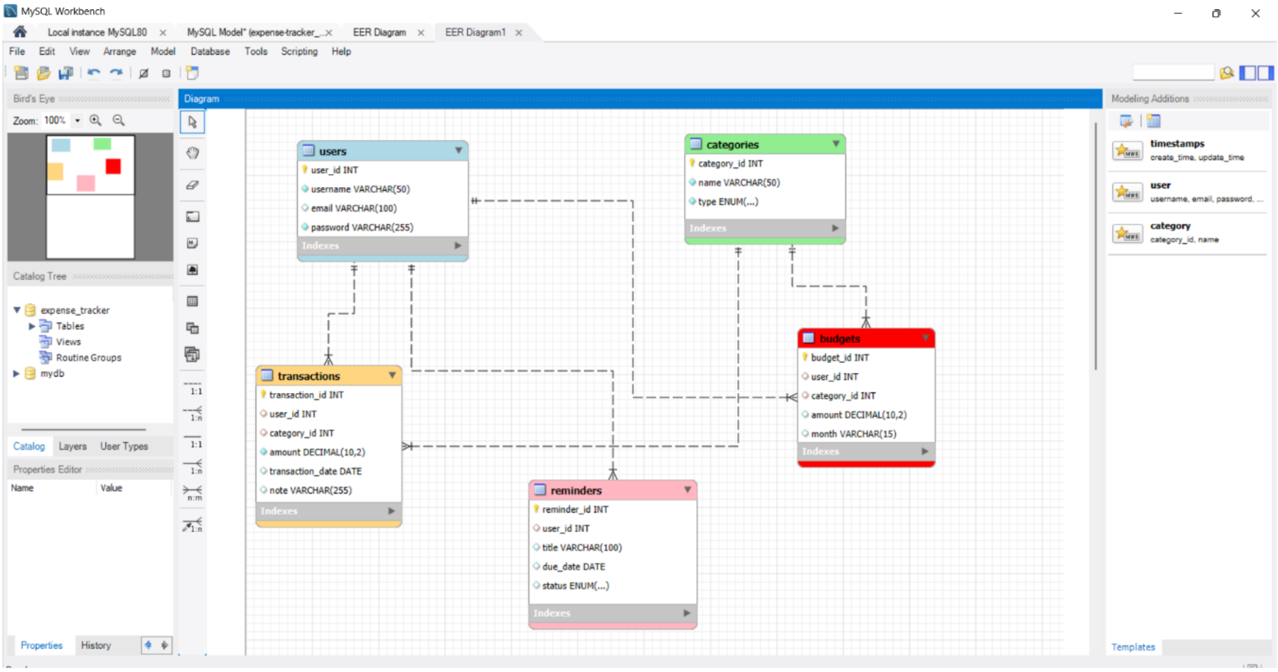
	#	Time	Action	Message	Duration / Fetch
✓	71	08:32:29	INSERT INTO budgets (user_id, category_id, amount, month) VALUES (1, 3, 3000, 'July'), (1, 4, 8000, 'July'), (...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.000 sec
✓	72	08:32:29	INSERT INTO reminders (user_id, title, due_date, status) VALUES (1, 'Pay electricity bill', '2025-07-10', 'pendin...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.016 sec
✓	73	08:32:29	INSERT INTO transactions (user_id, category_id, amount, transaction_date, note) VALUES (1, 3, 1800, '2025-...	1 row(s) affected	0.000 sec
✓	74	08:32:29	UPDATE budgets SET amount = 3500 WHERE user_id = 1 AND category_id = 3	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.000 sec
✓	75	08:32:29	DELETE FROM reminders WHERE reminder_id = 5	1 row(s) affected	0.015 sec
✓	76	08:32:29	SELECT u.usemame, SUM(t.amount) AS total_expense FROM transactions t JOIN users u ON t.user_id ...	3 row(s) returned	0.000 sec / 0.000 sec



```sql
-- Step 6.2: Show total expenses by each user
SELECT
    u.username,
    SUM(t.amount) AS total_expense
FROM transactions t
JOIN users u ON t.user_id = u.user_id
JOIN categories c ON t.category_id = c.category_id
```

username	total_expense
pavitra	9700.00
john	8000.00
sita	1200.00

```
#    Time      Action                                                                                                      Message                                                  Duration / Fetch
222  14:44:04  SELECT * FROM reminders LIMIT 0, 1000                                                                       4 row(s) returned                                        0.000 sec / 0.000 sec
223  14:44:04  INSERT INTO transactions (user_id, category_id, amount, transaction_date, note) VALUES (1, 3, 1800, '2025-07-06', 'Snacks and s...   1 row(s) affected    0.016 sec
224  14:44:04  UPDATE budgets SET amount = 3500 WHERE user_id = 1 AND category_id = 3                                     0 row(s) affected Rows matched: 1  Changed: 0  Warnings: 0   0.000 sec
225  14:44:04  DELETE FROM reminders WHERE reminder_id = 5                                                                 0 row(s) affected                                        0.000 sec
226  14:44:04  SELECT    t.transaction_id,    u.username,    c.name AS category,    t.amount,    t.transaction_date,    t.note FROM transactions t...   9 row(s) returned   0.000 sec / 0.000 sec
227  14:44:04  SELECT    u.username,    SUM(t.amount) AS total_expense FROM transactions t JOIN users u ON t.user_id = u.user_id JOIN cate...   3 row(s) returned   0.000 sec / 0.000 sec
```



7.CONCLUSION

The Expense Tracker App's backend was successfully implemented using a normalized relational database structure. The use of MySQL for table creation, data manipulation, and querying demonstrated the application's capacity to manage user financial data efficiently. The structured schema, validated through queries and ER diagram, ensures flexibility, scalability, and integrity. This project builds foundational knowledge in full-stack development and database design, making it a valuable learning experience for any beginner in Java and SQL development.