**SVKM’s NMIMS**

**School of Technology Management & Engineering (Indore Campus)**

**Computer Engineering Department (B Tech/MBATech CE and B Tech AIDS Sem IV)**

**Database Management System**

**Project Report**

|  |  |  |
| --- | --- | --- |
| Program | BTech C.E, Sec-B | |
| Semester | 4th | |
| Name of the Project: | Expense Tracker | |
|  | | |
| Details of Project Members |  |  |
| Batch | Roll No. | Name |
| 02 | D065 | Pratikshya Dash |
| 02 | D093 | Vaishnavi Gandhi |
| 02 | D094 | Vansh Ray Jain |
| Date of Submission: | | |

**Contribution of each project Members:**

|  |  |  |
| --- | --- | --- |
| Roll No. | Name: | Contribution |
| D065 | Pratikshya Dash | ER Diagram, Creating Database and Performing SQL queries |
| D093 | Vaishnavi Gandhi | Documentation and Performing SQL queries |
| D094 | Vansh Ray Jain | ER Diagram to Relational Schema Conversion and Performing SQL queries |

**Github link of your project:**

<https://github.com/Vishu4648/D065_D093_D094_DBMSProject.git>

**Note:**

1. Create a readme file if you have multiple files
2. All files must be properly named (Example:R004\_DBMSProject)
3. Submit all relevant files of your work ( Report, all SQL files, Any other files)
4. **Plagiarism is highly discouraged (Your report will be checked for plagiarism)**

**Rubrics for the Project evaluation:**

|  |  |
| --- | --- |
| First phase of evaluation:  Innovative Ideas (5 Marks)  Design and Partial implementation (5 Marks) | 10 marks |
| Final phase of evaluation  Implementation, presentation and viva, Self-Learning and Learning Beyond classroom | 10 marks |

**Project Report**

**Expense Tracker**

**by**

**Pratikshya Dash, Roll number: D065**

**Vaishnavi Gandhi, Roll number: D093**

**Vansh Ray Jain, Roll number: D094**

**Course: DBMS**

**AY: 2024-25**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sr no.** | **Topic** | **Page no.** |
| **1** | Storyline |  |
| **2** | Components of Database Design |  |
| **3** | Entity Relationship Diagram |  |
| **4** | Relational Model |  |
| **5** | Normalization |  |
| **6** | SQL Queries |  |
| **7** | Learning from the Project |  |
| **8** | Project Demonstration |  |
| **9** | Self-learning beyond classroom |  |
| **10** | Learning from the project |  |
| **8** | Challenges faced |  |
| **9** | Conclusion |  |

**I. Storyline**

The Expense Tracker System is designed to help users efficiently manage and analyze their daily expenses. It provides a structured approach to financial management by allowing users to record and categorize expenses, set budgets, track recurring costs, and visualize spending patterns through insightful reports and charts. The system integrates essential functionalities such as credit tracking, payment method management, and real-time alerts, ensuring a seamless and intuitive user experience.

Key Features & Functionalities

Expense Recording & Categorization

Users can log expenses with details such as amount, category, and payment method.

Expenses are classified into predefined or custom categories for better analysis.

Budget & Financial Management

Users can set spending limits for different categories.

The system provides real-time alerts when budgets are exceeded.

Recurring Expense Tracking

Automatically tracks periodic expenses such as rent, subscriptions, and loan payments.

Payment Method Integration

Supports various payment options, including cash, credit/debit cards, and digital wallets.

Credit Tracking

Monitors borrowed money, due dates, and interest rates to ensure responsible credit management.

Data Visualization & Insights

Generates reports, charts, and trends to provide a clear understanding of spending patterns.

User Interaction & Experience

Users register and log in to access their personalized dashboard.

Expenses can be added manually or through automated transaction imports.

The system provides alerts for budget limits and upcoming payments.

**II. Components of Database Design**

***ENTITIES ALONG ATTRIBUTES-***

Each **rectangle (entity)** in the ER diagram has **ovals (attributes)** linked to it.

**1. Users**

* user\_id (**Primary Key**)
* name
* email (**Unique**)
* password
* created\_at

**2. Expenses**

* expense\_id (**Primary Key**)
* amount
* date
* description

**3. Categories**

* category\_id (**Primary Key**)
* category\_name
* subcategory\_name (Now inside Category!)

**4. Payment Methods**

* payment\_method\_id (**Primary Key**)
* method\_name
* transaction\_type (Credit/Debit/UPI/Bank Transfer)

**5. Banks**

* bank\_id (**Primary Key**)
* bank\_name
* branch
* account\_number

**6. Budgets**

* budget\_id (**Primary Key**)
* amount\_limit
* start\_date
* end\_date

**7. Savings**

* savings\_id (**Primary Key**)
* goal\_amount
* amount\_saved
* start\_date
* end\_date

**8. Credit**

* credit\_id (**Primary Key**)
* amount
* interest\_rate
* due\_date

**9. Recurring Expenses**

* recurring\_id (**Primary Key**)
* amount
* frequency
* next\_due\_date

**10. Reports**

* report\_id (**Primary Key**)
* total\_income
* total\_expense
* savings\_summary
* generated\_date

**11. Investments**

* investment\_id (**Primary Key**)
* amount
* type (stocks, mutual funds, gold, etc.)
* return\_rate
* start\_date
* maturity\_date

***ASSOCIATION TABLE-***



***CARDINALITY****-*

1. **User Makes Expense** *(One-to-Many: Users → Expenses)*
   * A **user** can have **multiple expenses**, but each **expense** belongs to only **one user**.
2. **Expense Belongs to a Category** *(Many-to-One: Expenses → Categories)*
   * Each **expense** falls into **one category**, but a **category** can have **many expenses**.
3. **Expense Is Paid With a Payment Method** *(Many-to-One: Expenses → Payment Methods)*
   * Each **expense** is made using **one payment method**, but a **payment method** can be used for **many expenses**.
4. **Payment Method Can Be Linked to a Bank** *(Many-to-One: Payment Methods → Banks)*
   * Each **payment method** can be linked to **one bank**, but a **bank** can support **multiple payment methods**.
5. **User Sets a Budget** *(One-to-Many: Users → Budgets)*
   * A **user** can set **multiple budgets**, but each **budget** belongs to **one user**.
6. **Budget Allocates Funds to Category** *(One-to-One: Budgets → Categories)*
   * Each **budget** is assigned to **one category**.
7. **User Saves Money** *(One-to-Many: Users → Savings)*
   * A **user** can have **multiple savings goals**, but each **savings** record belongs to **one user**.
8. **User Takes Credit** *(One-to-Many: Users → Credit)*
   * A **user** can have **multiple credit transactions**, but each **credit** is linked to **one user**.
9. **User Schedules Recurring Expenses** *(One-to-Many: Users → Recurring Expenses)*
   * A **user** can have **many recurring expenses**, but each **recurring expense** belongs to **one user**.
10. **User Generates Reports** *(One-to-One: Users → Reports)*

* Each **user** has **one report at a time** that summarizes their financials.

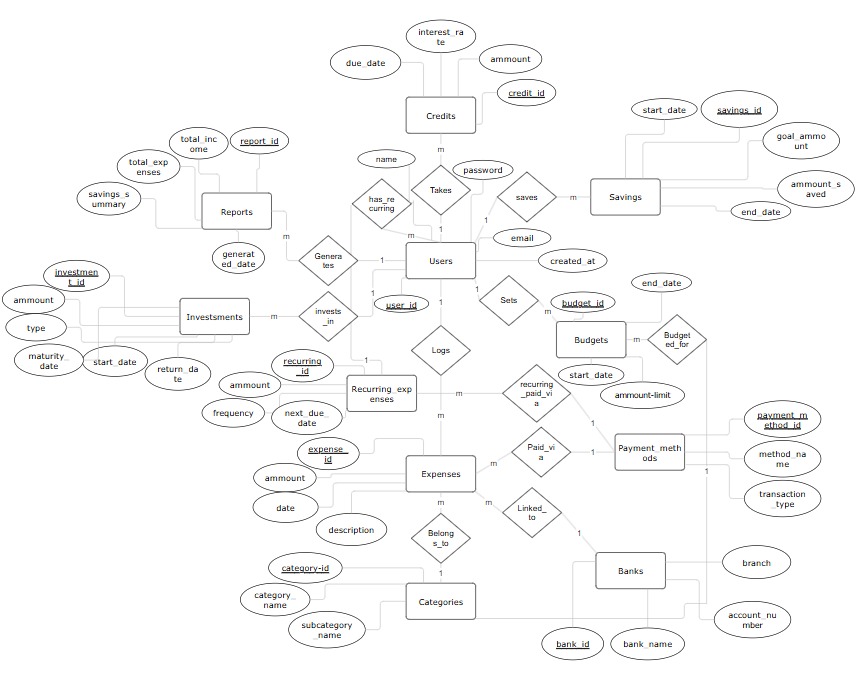
1. **User Invests Money** *(One-to-Many: Users → Investments)*

* A **user** can have **multiple investments**, but each **investment** is linked to **one user**.

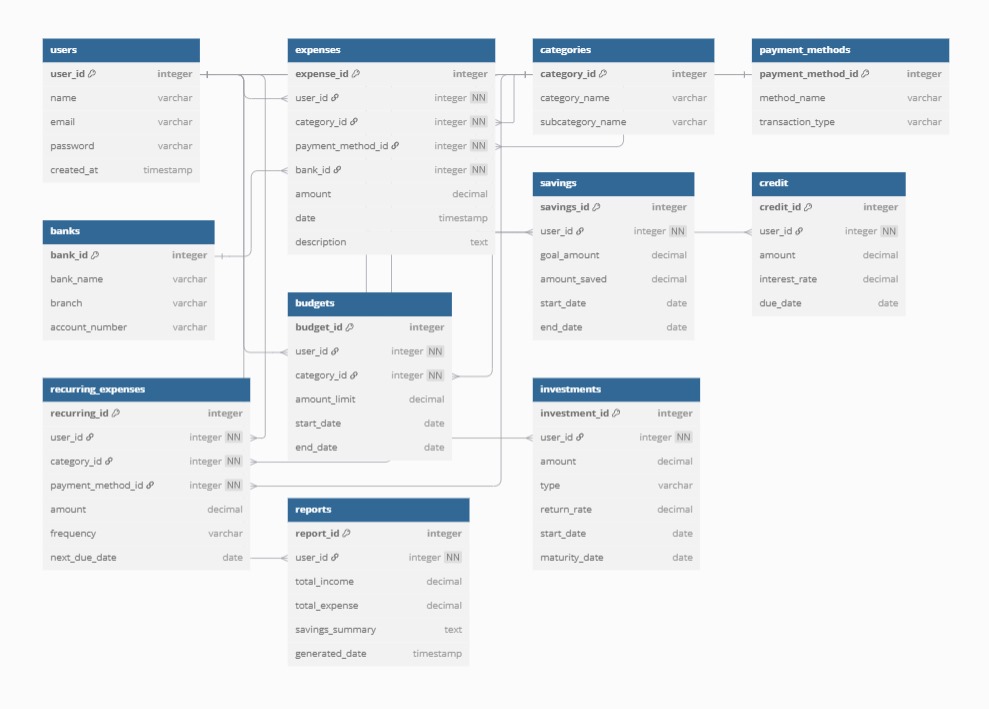
***PARTICIPATION-***

* Users – Yes, it is the total participation. Every user must record an expense.
* Expenses – Yes, it is the total participation. Every expense belongs to a user, category, and payment method.
* Categories – No, it is not the total participation. Some categories may not have expenses yet.
* Subcategories – No, it is not the total participation. Some subcategories may not be used yet.
* Payment Methods – No, it is not the total participation. Some methods may not be linked to expenses yet.
* Budgets – Yes, it is the total participation. Every budget must be assigned to a user.
* Savings – Yes, it is the total participation. Every savings entry must belong to a user.
* Recurring Expenses – Yes, it is the total participation. Every recurring expense is tied to a user and category.
* Credit Transactions – Yes, it is the total participation. Every credit transaction must belong to a user.
* Transaction Type – Yes, it is the total participation. Every transaction type must be linked to a payment method.
* Takes (Relationship Set) – Yes, it is the total participation. Every user must have at least one transaction.

**III. Entity Relationship Diagram**



**IV. Relational Model**

****

***RELATIONAL SCHEMAS-***

1. Users Table

Users (user\_id PK, name, email, password)

1. **Expenses Table**

Expenses (expense\_id PK, user\_id FK, category\_id FK, payment\_method\_id FK, bank\_id FK, amount, date, description)

### **Categories Table**

Categories (category\_id PK, category\_name)

### **Payment Methods Table**

Payment\_Methods (payment\_method\_id PK, method\_name)

1. Banks Table

Banks (bank\_id PK, bank\_name)

### **Budgets Table**

Budgets (budget\_id PK, user\_id FK, category\_id FK, amount, start\_date, end\_date)

1. Savings Table

Savings (savings\_id PK, user\_id FK, amount, date)

1. Credit Table

Credit (credit\_id PK, user\_id FK, credit\_limit, balance, due\_date)

1. Recurring Expenses Table

Recurring\_Expenses (recurring\_expense\_id PK, user\_id FK, category\_id FK, payment\_method\_id FK, amount, frequency)

1. Report Table

Reports (report\_id PK, user\_id FK, report\_type, generated\_date)

1. Investments Table

Investments (investment\_id PK, user\_id FK, investment\_type, amount, date)

**V. Normalization**

Normalization Analysis

First Normal Form (1NF)

Definition:  
A relation is in First Normal Form if:

* All values in each column are atomic (indivisible).
* There are no repeating groups or arrays.
* Each column has a unique name and maintains a consistent data type.

Assessment:  
All tables in the schema conform to the principles of 1NF. Each attribute holds atomic values, there are no repeating groups, and column names are unique with consistent data types.

Conclusion:  
All tables are in First Normal Form (1NF).

Second Normal Form (2NF)

Definition:  
A relation is in Second Normal Form if:

* It is already in 1NF.
* There are no partial dependencies of non-key attributes on a composite primary key.

Assessment:  
As all tables use a single-column primary key, partial dependencies on a composite key do not exist.

Conclusion:  
All tables satisfy the requirements of Second Normal Form (2NF).

Third Normal Form (3NF)

Definition:  
A relation is in Third Normal Form if:

* It is already in 2NF.
* There are no transitive dependencies, i.e., non-key attributes must depend solely on the primary key.

Assessment:  
Most tables satisfy 3NF, but as noted in the previous design review, some tables (e.g., categories, expenses, budgets) may require further normalization to fully eliminate transitive dependencies.

Conclusion:  
Tables are designed with 3NF in mind. However, a few require minor adjustments (e.g., introducing foreign keys or normalization of subcategories) to ensure full compliance.

**Relational Design Review and Normalization Recommendations**

1. Users Table

Observation: All attributes are fully functionally dependent on the primary key user\_id.  
Conclusion: The table conforms to 2NF and 3NF.  
Action: No changes required.

2. Expenses Table

Observation: The table currently assumes expense\_id is the sole determinant. However, the design omits foreign keys that define essential relationships. Each expense should be associated with:

* A user (user\_id)
* A category (category\_id)
* A subcategory (subcategory\_id)
* A payment method (payment\_method\_id)
* A bank (bank\_id) when applicable

Recommendation:

* Add the necessary foreign keys to ensure all dependencies are properly normalized.
* This eliminates transitive dependencies and maintains referential integrity.

Action: Normalize dependencies and include appropriate foreign key constraints.

3. Categories Table

Observation: The attribute subcategory\_name may depend on category\_name, rather than directly on category\_id.  
Recommendation: Normalize subcategories into a separate table to ensure 3NF:

sql

CopyEdit

Subcategories (

subcategory\_id (PRIMARY KEY),

subcategory\_name,

category\_id (FOREIGN KEY)

)

Action: Move subcategories into a new Subcategories table with appropriate foreign key references.

4. Payment Methods Table

Observation: All attributes are fully dependent on payment\_method\_id.  
Conclusion: The table is normalized.  
Action: No changes required.

5. Banks Table

Observation: All attributes are directly dependent on the primary key bank\_id. No transitive dependencies observed.  
Conclusion: The table is in 3NF.  
Action: No changes required.

6. Budgets Table

Observation: Currently assumes budget\_id determines all other fields. However, budgets should be associated with users.  
Recommendation: Include user\_id as a foreign key to link each budget to its respective user.

Action: Add user\_id (FOREIGN KEY) to establish ownership by user.

7. Savings Table

Observation: Similar to Budgets, savings goals are typically specific to individual users.  
Recommendation: Add user\_id as a foreign key.

Action: Update the schema to include user\_id (FOREIGN KEY).

8. Credit Table

Observation: Credit entries must be linked to individual users.  
Recommendation: Introduce user\_id as a foreign key for relational consistency.

Action: Add user\_id (FOREIGN KEY) to establish user ownership.

9. Recurring Expenses Table

Observation: Recurring expenses must be associated with users, and optionally, categories and payment methods.  
Recommendation: Ensure user\_id, category\_id, and payment\_method\_id are all included as foreign keys.

Action: Add the relevant foreign keys to support proper normalization.

10. Reports Table

Observation: Reports are generated per user, but current design does not explicitly enforce this relationship.  
Recommendation: Include user\_id as a foreign key and enforce uniqueness to reflect one report per user.

Action: Add user\_id (FOREIGN KEY, UNIQUE) to define a one-to-one relationship with Users.

11. Investments Table

Observation: Investments should be associated with users. The type attribute appears atomic and valid if treated as a category.  
Recommendation: Add user\_id as a foreign key to indicate ownership.

Action: Include user\_id (FOREIGN KEY) to complete the relationship.

**VI. SQL Queries**

Using a DBMS software (SQLite3 or MySQL or any other of your choice):

* Create the tables
* Populate the tables (insert some meaningful data, at least 10 tuples for each relation)
* Run SQL queries (minimum 20) covering **all concepts** learned in the class

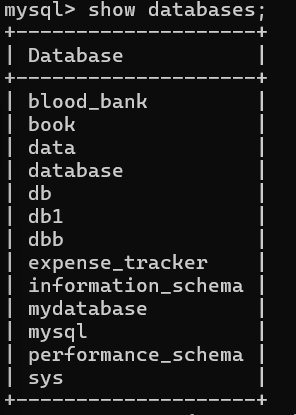
This section should contain the question, SQL code, and the output snapshot for each query.

**Initial query:**

1. create database Expense\_tracker;



1. show databases;



1. use Expense\_tracker;



1. -- Users table

CREATE TABLE users (

user\_id INT PRIMARY KEY,

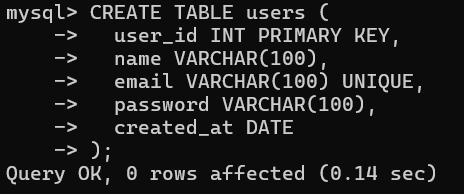
name VARCHAR(100),

email VARCHAR(100) UNIQUE,

password VARCHAR(100),

created\_at DATE

);



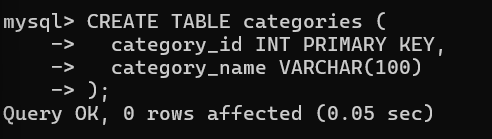
1. -- Categories table

CREATE TABLE categories (

category\_id INT PRIMARY KEY,

category\_name VARCHAR(100)

);



1. -- Subcategories (normalized from categories)

CREATE TABLE subcategories (

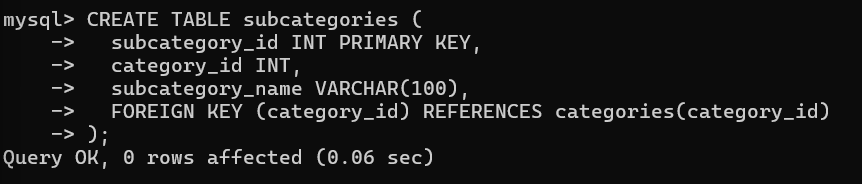
subcategory\_id INT PRIMARY KEY,

category\_id INT,

subcategory\_name VARCHAR(100),

FOREIGN KEY (category\_id) REFERENCES categories(category\_id)

);



1. -- Payment Methods table

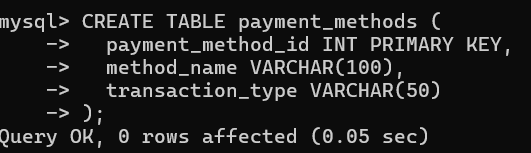
CREATE TABLE payment\_methods (

payment\_method\_id INT PRIMARY KEY,

method\_name VARCHAR(100),

transaction\_type VARCHAR(50)

);



1. -- Banks table

CREATE TABLE banks (

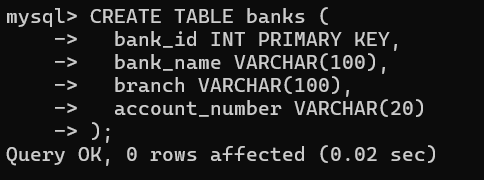
bank\_id INT PRIMARY KEY,

bank\_name VARCHAR(100),

branch VARCHAR(100),

account\_number VARCHAR(20)

);



1. -- Expenses table

CREATE TABLE expenses (

expense\_id INT PRIMARY KEY,

user\_id INT,

category\_id INT,

subcategory\_id INT,

payment\_method\_id INT,

bank\_id INT,

amount DECIMAL(10,2),

date DATE,

description VARCHAR(255),

FOREIGN KEY (user\_id) REFERENCES users(user\_id),

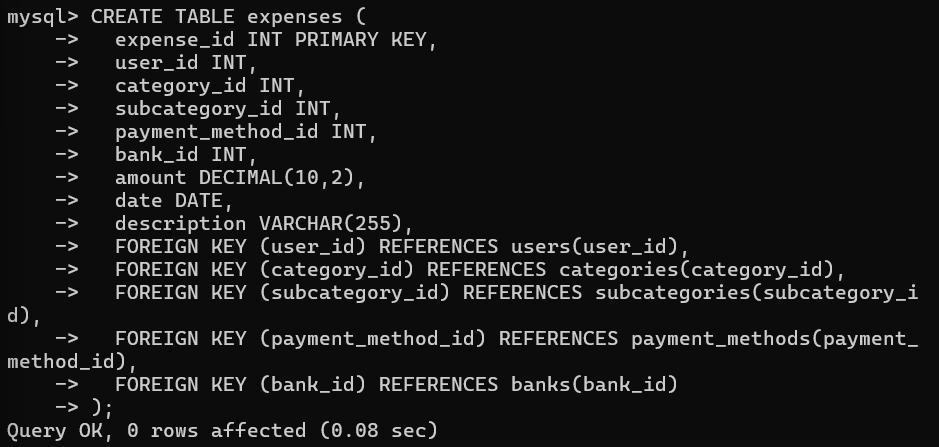
FOREIGN KEY (category\_id) REFERENCES categories(category\_id),

FOREIGN KEY (subcategory\_id) REFERENCES subcategories(subcategory\_id),

FOREIGN KEY (payment\_method\_id) REFERENCES payment\_methods(payment\_method\_id),

FOREIGN KEY (bank\_id) REFERENCES banks(bank\_id)

);



1. -- Budgets table

CREATE TABLE budgets (

budget\_id INT PRIMARY KEY,

user\_id INT,

category\_id INT,

amount DECIMAL(10,2),

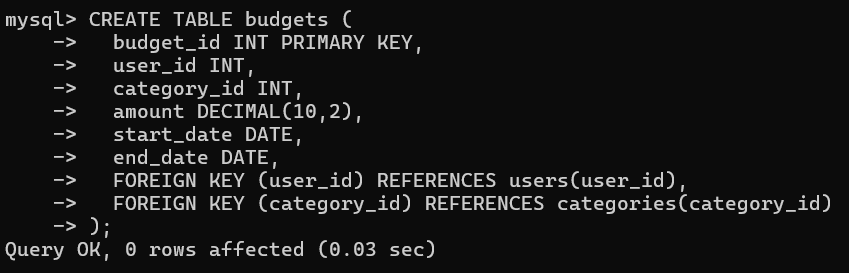
start\_date DATE,

end\_date DATE,

FOREIGN KEY (user\_id) REFERENCES users(user\_id),

FOREIGN KEY (category\_id) REFERENCES categories(category\_id)

);



1. -- Savings table

CREATE TABLE savings (

savings\_id INT PRIMARY KEY,

user\_id INT,

goal\_amount DECIMAL(10,2),

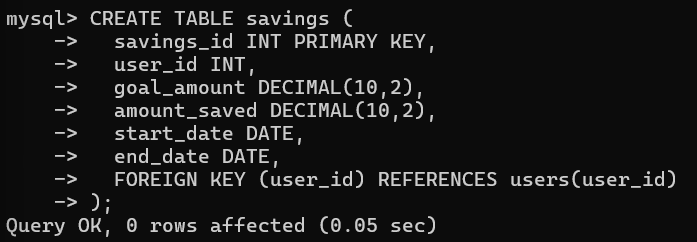
amount\_saved DECIMAL(10,2),

start\_date DATE,

end\_date DATE,

FOREIGN KEY (user\_id) REFERENCES users(user\_id)

);



1. -- Credit table

CREATE TABLE credit (

credit\_id INT PRIMARY KEY,

user\_id INT,

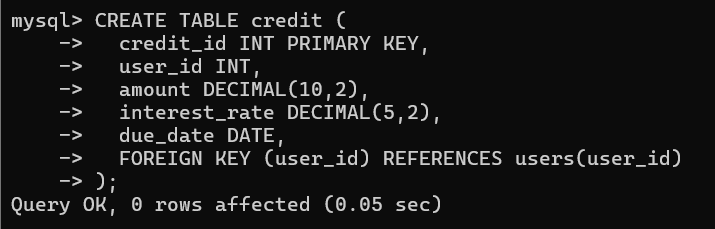
amount DECIMAL(10,2),

interest\_rate DECIMAL(5,2),

due\_date DATE,

FOREIGN KEY (user\_id) REFERENCES users(user\_id)

);



1. -- Recurring Expenses table

CREATE TABLE recurring\_expenses (

recurring\_expense\_id INT PRIMARY KEY,

user\_id INT,

category\_id INT,

subcategory\_id INT,

payment\_method\_id INT,

amount DECIMAL(10,2),

frequency VARCHAR(50),

next\_due\_date DATE,

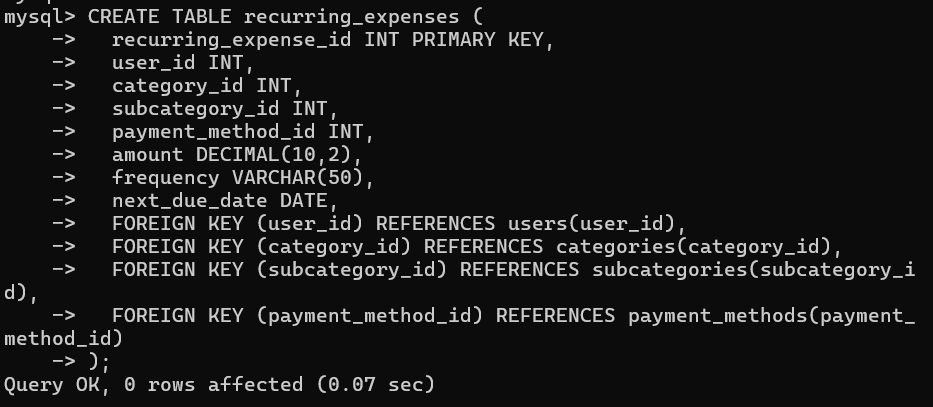
FOREIGN KEY (user\_id) REFERENCES users(user\_id),

FOREIGN KEY (category\_id) REFERENCES categories(category\_id),

FOREIGN KEY (subcategory\_id) REFERENCES subcategories(subcategory\_id),

FOREIGN KEY (payment\_method\_id) REFERENCES payment\_methods(payment\_method\_id)

);



1. -- Reports table

CREATE TABLE reports (

report\_id INT PRIMARY KEY,

user\_id INT UNIQUE,

total\_income DECIMAL(10,2),

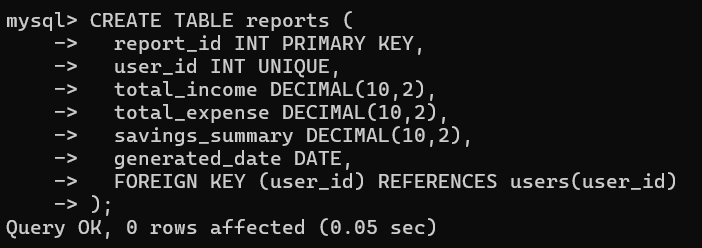
total\_expense DECIMAL(10,2),

savings\_summary DECIMAL(10,2),

generated\_date DATE,

FOREIGN KEY (user\_id) REFERENCES users(user\_id)

);



1. -- Investments table

CREATE TABLE investments (

investment\_id INT PRIMARY KEY,

user\_id INT,

amount DECIMAL(10,2),

type VARCHAR(50),

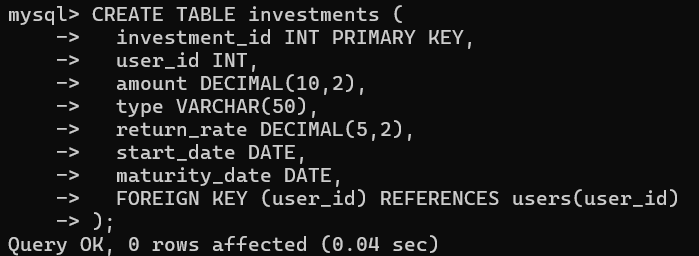
return\_rate DECIMAL(5,2),

start\_date DATE,

maturity\_date DATE,

FOREIGN KEY (user\_id) REFERENCES users(user\_id)

);



1. -- USERS

INSERT INTO users VALUES

(1, 'Alice', 'alice@example.com', 'password123', '2023-01-01'),

(2, 'Bob', 'bob@example.com', 'password456', '2023-01-02'),

(3, 'Charlie', 'charlie@example.com', 'password789', '2023-01-03'),

(4, 'David', 'david@example.com', 'password321', '2023-01-04'),

(5, 'Eva', 'eva@example.com', 'password654', '2023-01-05'),

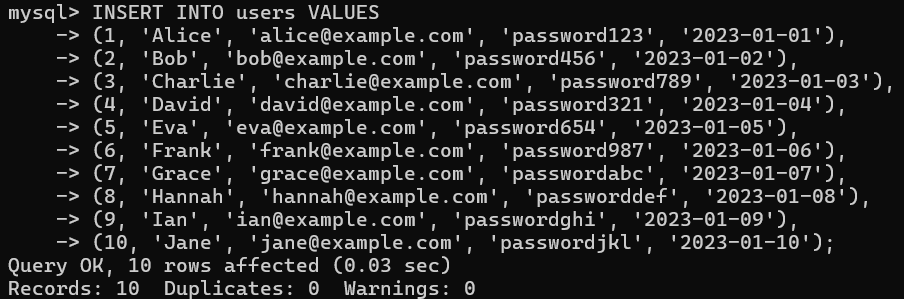
(6, 'Frank', 'frank@example.com', 'password987', '2023-01-06'),

(7, 'Grace', 'grace@example.com', 'passwordabc', '2023-01-07'),

(8, 'Hannah', 'hannah@example.com', 'passworddef', '2023-01-08'),

(9, 'Ian', 'ian@example.com', 'passwordghi', '2023-01-09'),

(10, 'Jane', 'jane@example.com', 'passwordjkl', '2023-01-10');

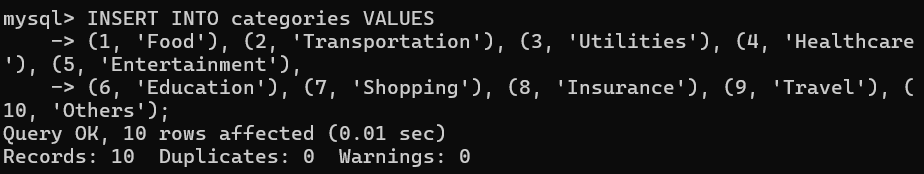


1. -- CATEGORIES

INSERT INTO categories VALUES

(1, 'Food'), (2, 'Transportation'), (3, 'Utilities'), (4, 'Healthcare'), (5, 'Entertainment'),

(6, 'Education'), (7, 'Shopping'), (8, 'Insurance'), (9, 'Travel'), (10, 'Others');



1. -- SUBCATEGORIES

INSERT INTO subcategories VALUES

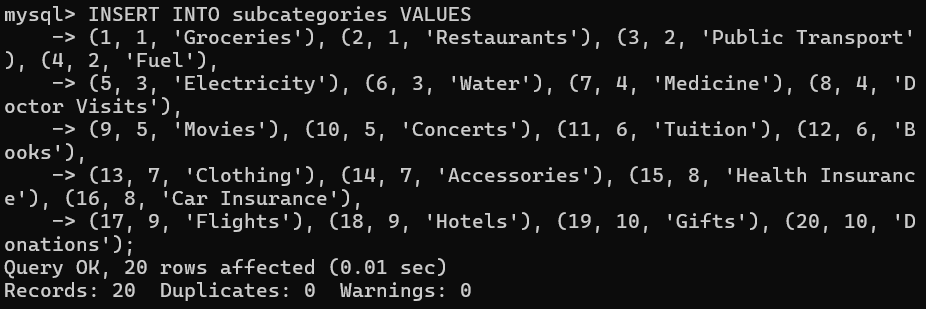
(1, 1, 'Groceries'), (2, 1, 'Restaurants'), (3, 2, 'Public Transport'), (4, 2, 'Fuel'),

(5, 3, 'Electricity'), (6, 3, 'Water'), (7, 4, 'Medicine'), (8, 4, 'Doctor Visits'),

(9, 5, 'Movies'), (10, 5, 'Concerts'), (11, 6, 'Tuition'), (12, 6, 'Books'),

(13, 7, 'Clothing'), (14, 7, 'Accessories'), (15, 8, 'Health Insurance'), (16, 8, 'Car Insurance'),

(17, 9, 'Flights'), (18, 9, 'Hotels'), (19, 10, 'Gifts'), (20, 10, 'Donations');



1. -- PAYMENT METHODS

INSERT INTO payment\_methods (payment\_method\_id, method\_name, transaction\_type) VALUES

(1, 'Cash', 'Expense'),

(2, 'Credit Card', 'Expense'),

(3, 'Debit Card', 'Expense'),

(4, 'UPI', 'Expense'),

(5, 'Net Banking', 'Expense'),

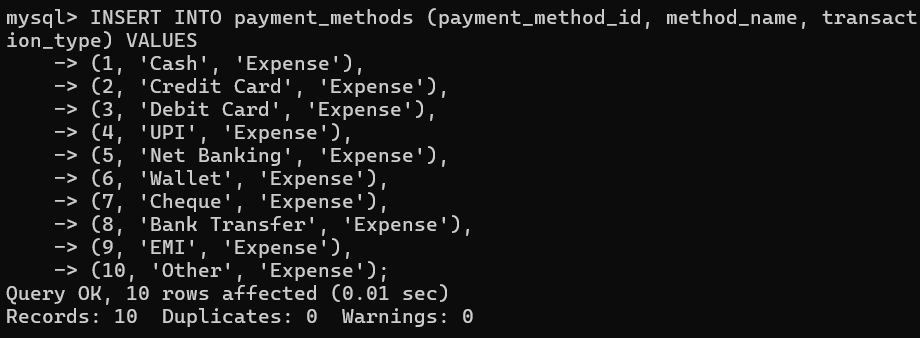
(6, 'Wallet', 'Expense'),

(7, 'Cheque', 'Expense'),

(8, 'Bank Transfer', 'Expense'),

(9, 'EMI', 'Expense'),

(10, 'Other', 'Expense');



1. -- BANKS

INSERT INTO banks (bank\_id, bank\_name, branch, account\_number) VALUES

(1, 'Bank of America', 'USA', '1234567890'),

(2, 'Chase', 'USA', '2345678901'),

(3, 'Wells Fargo', 'USA', '3456789012'),

(4, 'Citibank', 'USA', '4567890123'),

(5, 'HSBC', 'UK', '5678901234'),

(6, 'Barclays', 'UK', '6789012345'),

(7, 'Deutsche Bank', 'Germany', '7890123456'),

(8, 'BNP Paribas', 'France', '8901234567'),

(9, 'ICICI Bank', 'India', '9012345678'),

(10, 'HDFC Bank', 'India', '0123456789');



1. -- EXPENSES

INSERT INTO expenses VALUES

(1, 10, 7, 6, 9, 6, 339.61, '2024-04-29', 'Description 1'),

(2, 4, 6, 18, 9, 10, 87.00, '2024-07-21', 'Description 2'),

(3, 4, 1, 4, 6, 1, 475.56, '2024-03-11', 'Description 3'),

(4, 4, 9, 19, 2, 7, 400.25, '2024-02-01', 'Description 4'),

(5, 2, 1, 19, 4, 7, 332.17, '2024-01-17', 'Description 5'),

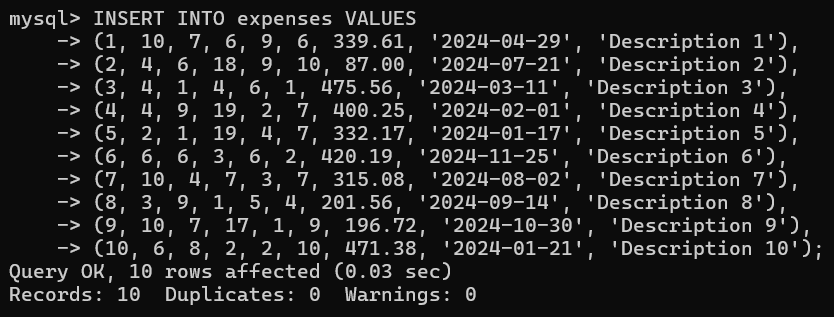
(6, 6, 6, 3, 6, 2, 420.19, '2024-11-25', 'Description 6'),

(7, 10, 4, 7, 3, 7, 315.08, '2024-08-02', 'Description 7'),

(8, 3, 9, 1, 5, 4, 201.56, '2024-09-14', 'Description 8'),

(9, 10, 7, 17, 1, 9, 196.72, '2024-10-30', 'Description 9'),

(10, 6, 8, 2, 2, 10, 471.38, '2024-01-21', 'Description 10');



1. -- BUDGETS

INSERT INTO budgets VALUES

(1, 1, 3, 1712.13, '2024-02-22', '2024-03-23'),

(2, 6, 7, 1017.66, '2024-07-26', '2024-08-25'),

(3, 6, 9, 1013.64, '2024-08-09', '2024-09-08'),

(4, 6, 3, 1854.18, '2024-02-12', '2024-03-13'),

(5, 1, 3, 1660.91, '2024-06-28', '2024-07-28'),

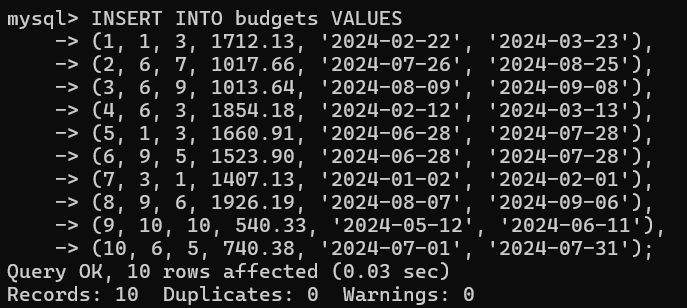
(6, 9, 5, 1523.90, '2024-06-28', '2024-07-28'),

(7, 3, 1, 1407.13, '2024-01-02', '2024-02-01'),

(8, 9, 6, 1926.19, '2024-08-07', '2024-09-06'),

(9, 10, 10, 540.33, '2024-05-12', '2024-06-11'),

(10, 6, 5, 740.38, '2024-07-01', '2024-07-31');



1. -- SAVINGS

INSERT INTO savings VALUES

(1, 1, 4002.44, 3541.35, '2024-09-18', '2024-11-17'),

(2, 2, 2580.81, 1139.05, '2024-05-27', '2024-07-26'),

(3, 3, 2542.50, 2279.69, '2024-06-21', '2024-08-20'),

(4, 4, 1773.76, 1474.13, '2024-02-07', '2024-04-07'),

(5, 5, 3926.60, 3911.38, '2024-04-06', '2024-06-05'),

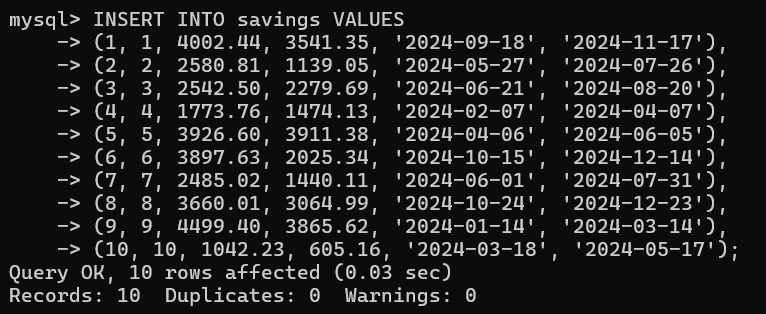
(6, 6, 3897.63, 2025.34, '2024-10-15', '2024-12-14'),

(7, 7, 2485.02, 1440.11, '2024-06-01', '2024-07-31'),

(8, 8, 3660.01, 3064.99, '2024-10-24', '2024-12-23'),

(9, 9, 4499.40, 3865.62, '2024-01-14', '2024-03-14'),

(10, 10, 1042.23, 605.16, '2024-03-18', '2024-05-17');



1. -- CREDIT

INSERT INTO credit VALUES

(1, 1, 4264.11, 11.55, '2024-03-28'),

(2, 2, 1686.12, 12.43, '2024-07-23'),

(3, 3, 2285.46, 5.45, '2024-06-20'),

(4, 4, 4377.52, 4.17, '2024-09-07'),

(5, 5, 1270.98, 12.59, '2024-01-10'),

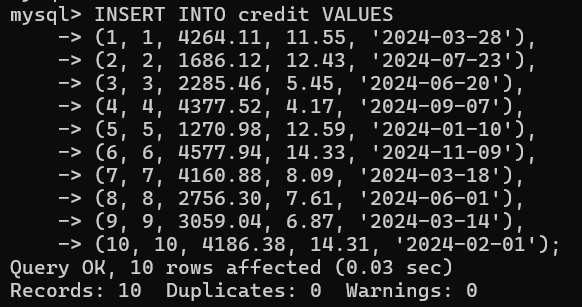
(6, 6, 4577.94, 14.33, '2024-11-09'),

(7, 7, 4160.88, 8.09, '2024-03-18'),

(8, 8, 2756.30, 7.61, '2024-06-01'),

(9, 9, 3059.04, 6.87, '2024-03-14'),

(10, 10, 4186.38, 14.31, '2024-02-01');



1. -- RECURRING EXPENSES

INSERT INTO recurring\_expenses VALUES

(1, 9, 8, 5, 5, 720.14, 'Yearly', '2024-11-11'),

(2, 9, 9, 8, 2, 537.98, 'Weekly', '2024-08-01'),

(3, 1, 6, 18, 4, 179.76, 'Monthly', '2024-07-22'),

(4, 2, 10, 6, 10, 279.83, 'Monthly', '2024-09-07'),

(5, 5, 1, 11, 1, 264.90, 'Yearly', '2024-01-21'),

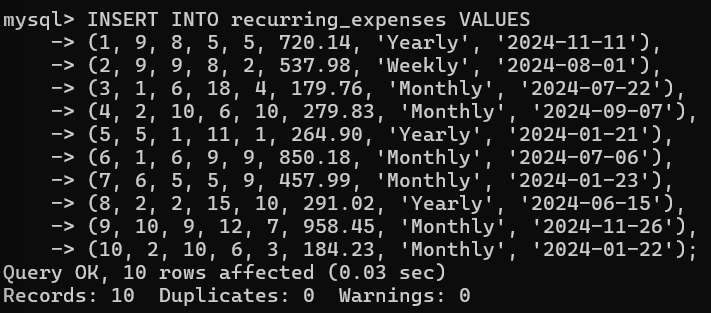
(6, 1, 6, 9, 9, 850.18, 'Monthly', '2024-07-06'),

(7, 6, 5, 5, 9, 457.99, 'Monthly', '2024-01-23'),

(8, 2, 2, 15, 10, 291.02, 'Yearly', '2024-06-15'),

(9, 10, 9, 12, 7, 958.45, 'Monthly', '2024-11-26'),

(10, 2, 10, 6, 3, 184.23, 'Monthly', '2024-01-22');



1. -- REPORTS

INSERT INTO reports VALUES

(1, 1, 6351.73, 2855.26, 3496.47, '2024-06-28'),

(2, 2, 8124.40, 553.48, 7570.92, '2024-08-22'),

(3, 3, 2183.62, 1146.21, 1037.41, '2024-06-22'),

(4, 4, 3237.84, 755.31, 2482.53, '2024-08-04'),

(5, 5, 4733.52, 1804.73, 2928.79, '2024-04-13'),

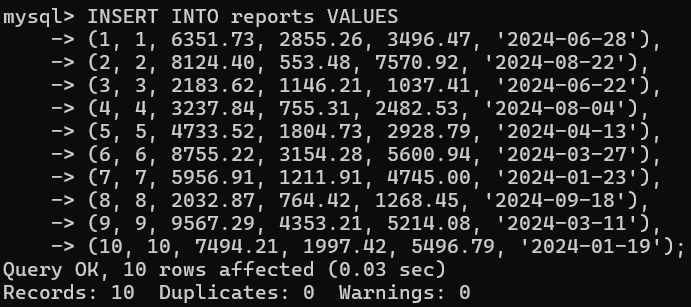
(6, 6, 8755.22, 3154.28, 5600.94, '2024-03-27'),

(7, 7, 5956.91, 1211.91, 4745.00, '2024-01-23'),

(8, 8, 2032.87, 764.42, 1268.45, '2024-09-18'),

(9, 9, 9567.29, 4353.21, 5214.08, '2024-03-11'),

(10, 10, 7494.21, 1997.42, 5496.79, '2024-01-19');



1. -- INVESTMENTS

INSERT INTO investments VALUES

(1, 1, 3435.74, 'Bonds', 5.24, '2024-04-29', '2024-07-28'),

(2, 2, 6464.64, 'Stocks', 5.09, '2024-08-23', '2024-11-21'),

(3, 3, 5990.52, 'Real Estate', 4.47, '2024-01-24', '2024-04-23'),

(4, 4, 9647.88, 'Mutual Funds', 3.62, '2024-07-10', '2024-10-08'),

(5, 5, 8793.61, 'Stocks', 9.42, '2024-03-18', '2024-06-16'),

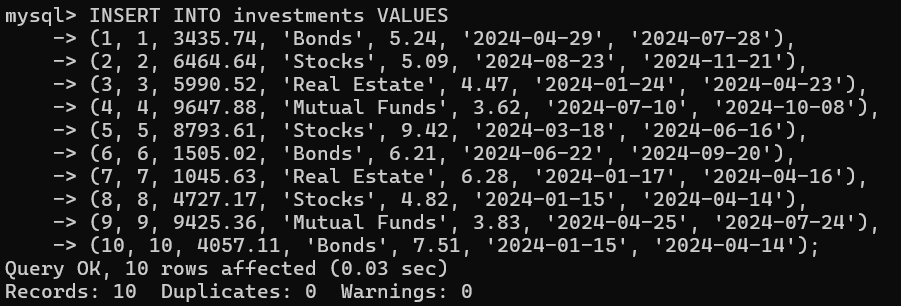
(6, 6, 1505.02, 'Bonds', 6.21, '2024-06-22', '2024-09-20'),

(7, 7, 1045.63, 'Real Estate', 6.28, '2024-01-17', '2024-04-16'),

(8, 8, 4727.17, 'Stocks', 4.82, '2024-01-15', '2024-04-14'),

(9, 9, 9425.36, 'Mutual Funds', 3.83, '2024-04-25', '2024-07-24'),

(10, 10, 4057.11, 'Bonds', 7.51, '2024-01-15', '2024-04-14');



**Question query:**

USE expense\_tracker;

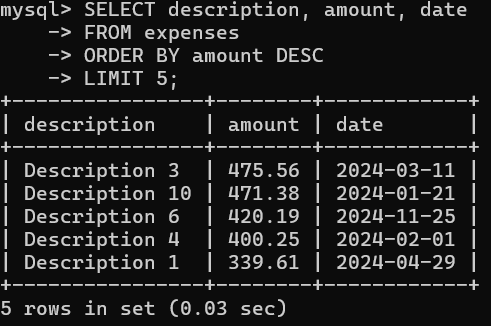
1. -- Top 5 most expensive purchases

SELECT description, amount, date

FROM expenses

ORDER BY amount DESC

LIMIT 5;



1. -- Monthly total expenses (last 6 months)

SELECT

DATE\_FORMAT(date, '%y-%m') AS month,

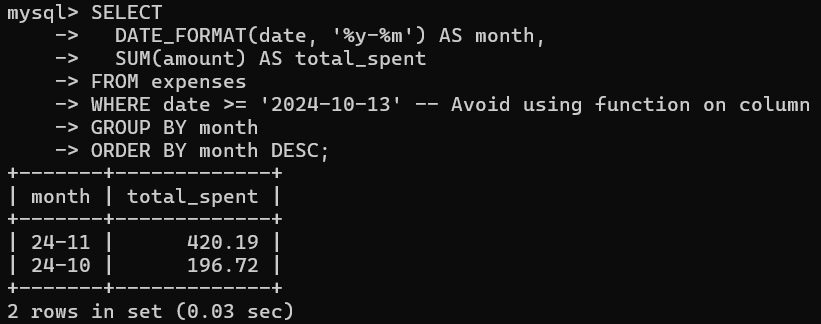
SUM(amount) AS total\_spent

FROM expenses

WHERE date >= '2024-10-13' -- Avoid using function on column

GROUP BY month

ORDER BY month DESC;



1. -- Total expenses by category

SELECT

c.category\_name,

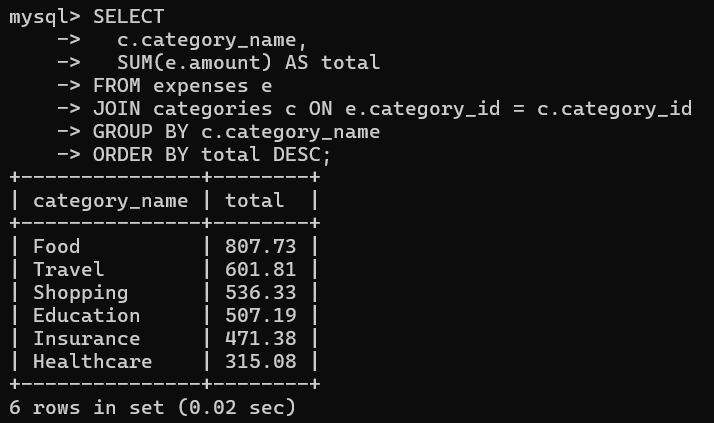
SUM(e.amount) AS total

FROM expenses e

JOIN categories c ON e.category\_id = c.category\_id

GROUP BY c.category\_name

ORDER BY total DESC;



1. -- Average daily spending in the current month

SELECT AVG(daily\_total) AS average\_spent

FROM (

SELECT

date,

SUM(amount) AS daily\_total

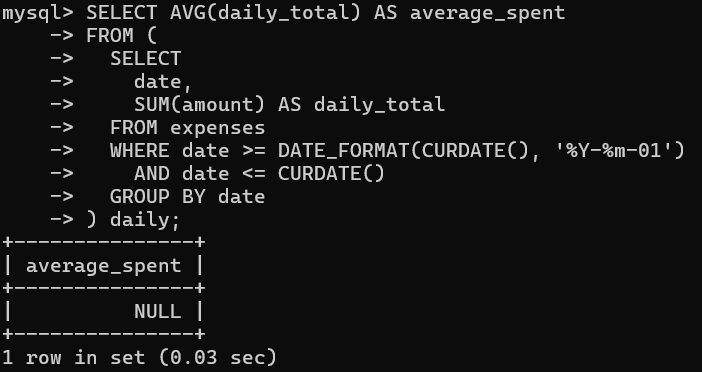
FROM expenses

WHERE date >= DATE\_FORMAT(CURDATE(), '%Y-%m-01')

AND date <= CURDATE()

GROUP BY date

) daily;



1. -- Expenses above budget by category

SELECT

e.expense\_id,

e.amount,

c.category\_name,

b.amount AS budget\_amount

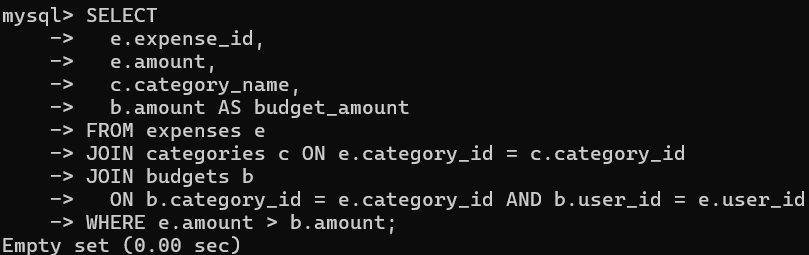
FROM expenses e

JOIN categories c ON e.category\_id = c.category\_id

JOIN budgets b

ON b.category\_id = e.category\_id AND b.user\_id = e.user\_id

WHERE e.amount > b.amount;



1. -- Users who exceeded any category budget

SELECT DISTINCT u.user\_id, u.name

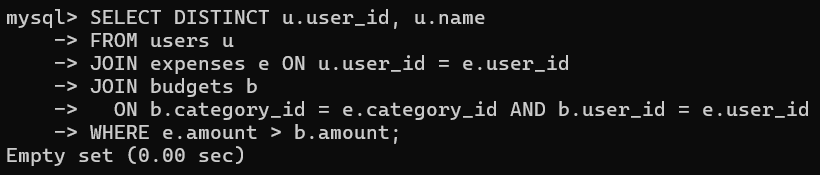
FROM users u

JOIN expenses e ON u.user\_id = e.user\_id

JOIN budgets b

ON b.category\_id = e.category\_id AND b.user\_id = e.user\_id

WHERE e.amount > b.amount;



1. -- Remaining budget per category

SELECT

b.user\_id,

c.category\_name,

b.amount - COALESCE(SUM(e.amount), 0) AS remaining\_budget

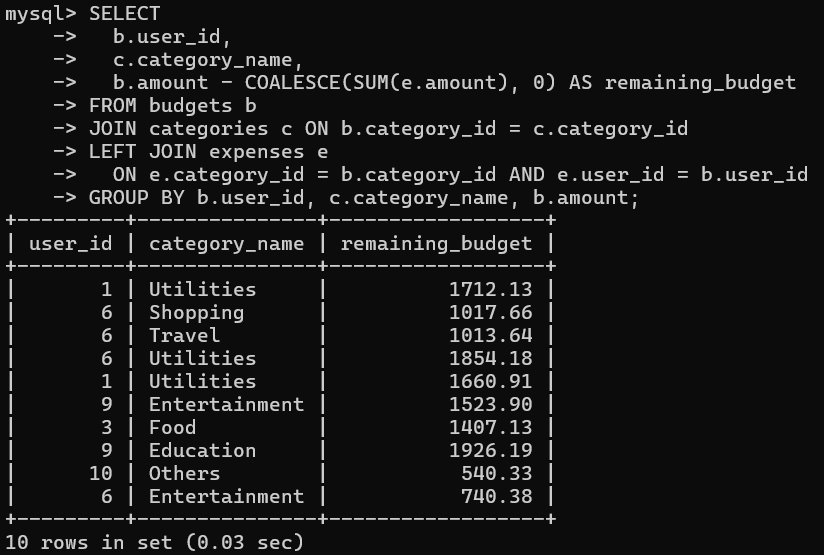
FROM budgets b

JOIN categories c ON b.category\_id = c.category\_id

LEFT JOIN expenses e

ON e.category\_id = b.category\_id AND e.user\_id = b.user\_id

GROUP BY b.user\_id, c.category\_name, b.amount;



1. -- Monthly budget vs actual spending

SELECT

u.user\_id,

DATE\_FORMAT(CURDATE(), '%y-%m') AS month,

SUM(DISTINCT b.amount) AS total\_budget,

SUM(e.amount) AS total\_spent

FROM users u

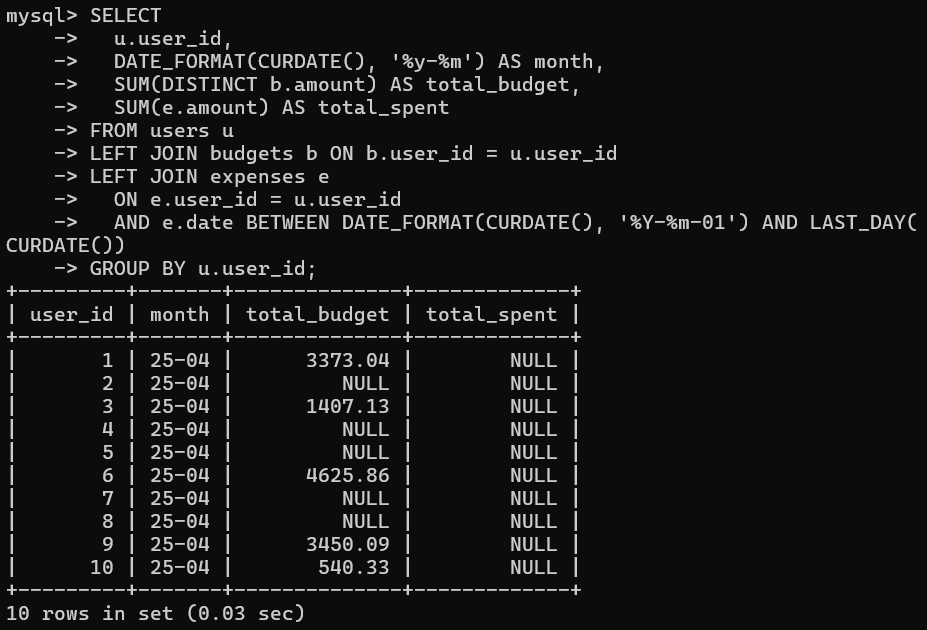
LEFT JOIN budgets b ON b.user\_id = u.user\_id

LEFT JOIN expenses e

ON e.user\_id = u.user\_id

AND e.date BETWEEN DATE\_FORMAT(CURDATE(), '%Y-%m-01') AND LAST\_DAY(CURDATE())

GROUP BY u.user\_id;

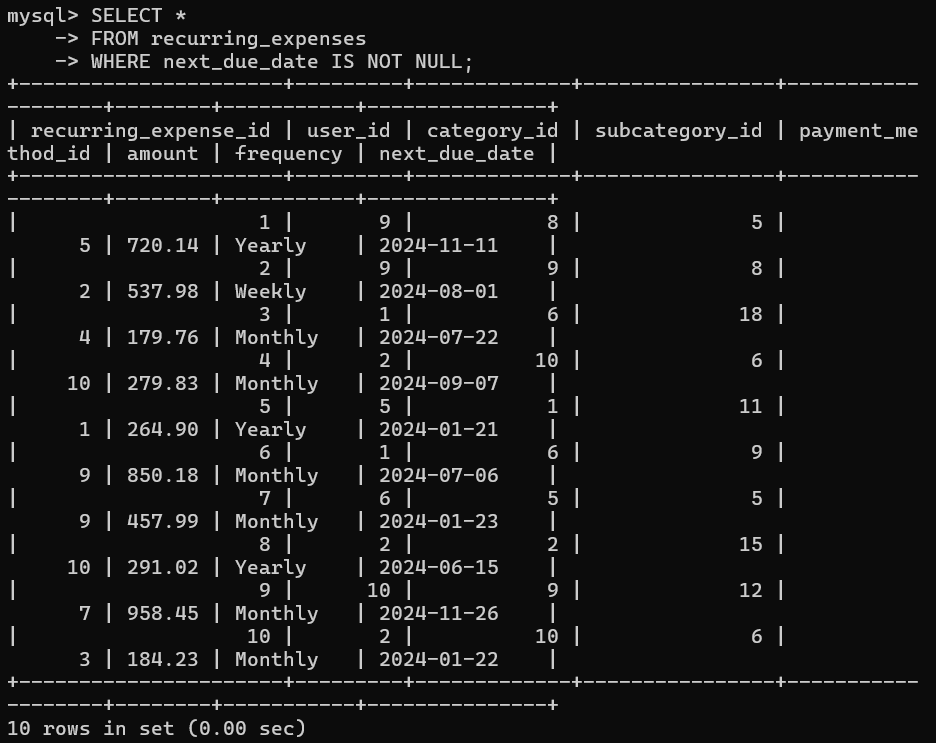


1. -- Active recurring expenses

SELECT \*

FROM recurring\_expenses

WHERE next\_due\_date IS NOT NULL;

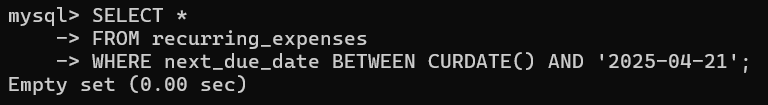


1. -- Upcoming recurring expenses in next 10 days

SELECT \*

FROM recurring\_expenses

WHERE next\_due\_date BETWEEN CURDATE() AND '2025-04-21';



1. -- Total monthly recurring commitment per user

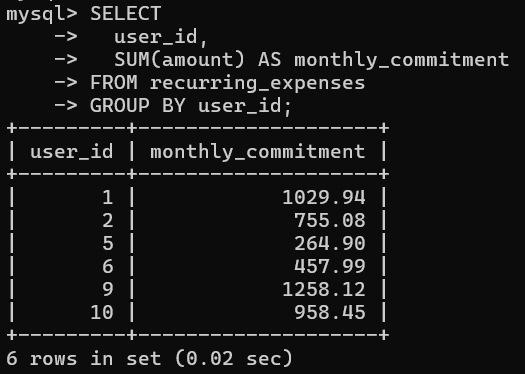
SELECT

user\_id,

SUM(amount) AS monthly\_commitment

FROM recurring\_expenses

GROUP BY user\_id;



1. -- Credit usage percentage per user

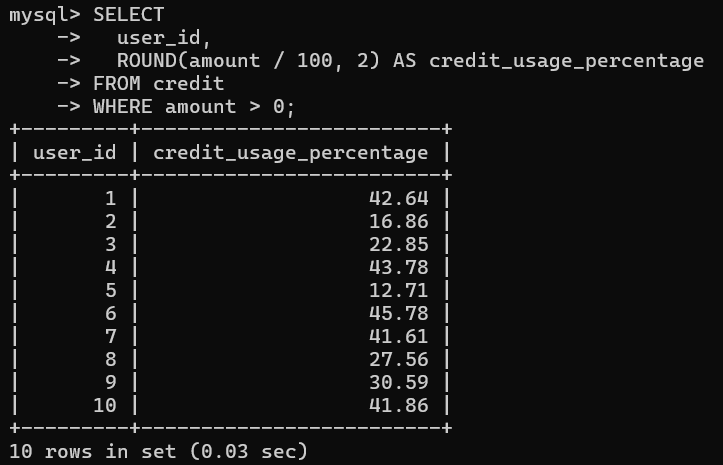
SELECT

user\_id,

ROUND(amount / 100, 2) AS credit\_usage\_percentage

FROM credit

WHERE amount > 0;



1. -- Users nearing credit limit (over 90%)

SELECT

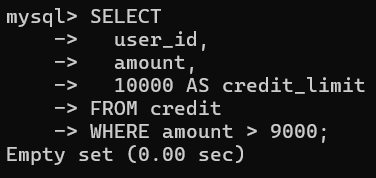
user\_id,

amount,

10000 AS credit\_limit

FROM credit

WHERE amount > 9000;



1. -- Total expenses by payment method

SELECT

p.method\_name,

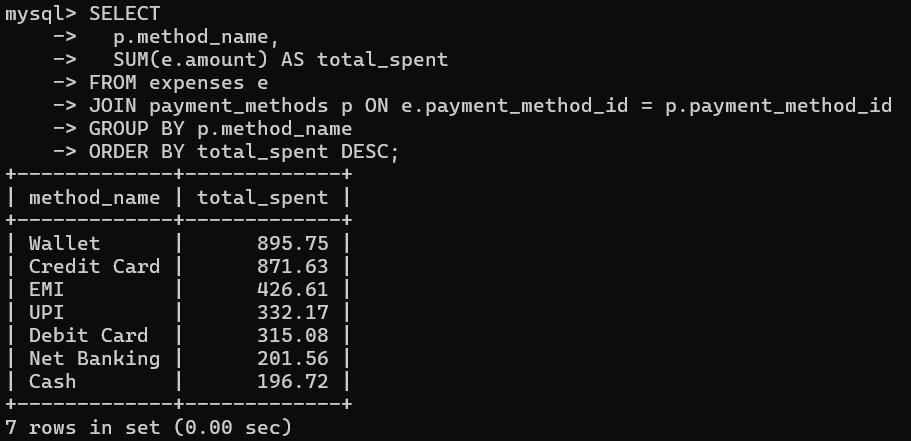
SUM(e.amount) AS total\_spent

FROM expenses e

JOIN payment\_methods p ON e.payment\_method\_id = p.payment\_method\_id

GROUP BY p.method\_name

ORDER BY total\_spent DESC;



1. -- Payment methods used by each user

SELECT DISTINCT

u.user\_id,

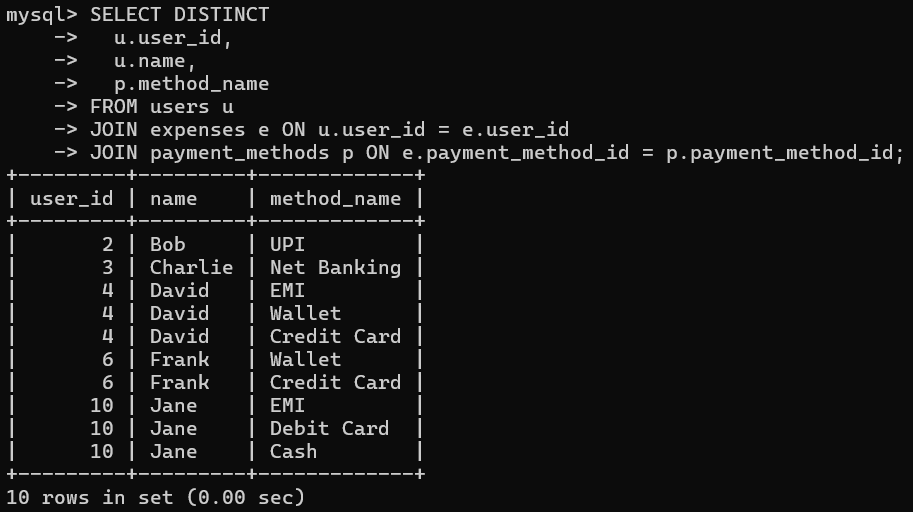
u.name,

p.method\_name

FROM users u

JOIN expenses e ON u.user\_id = e.user\_id

JOIN payment\_methods p ON e.payment\_method\_id = p.payment\_method\_id;



1. -- Bank-wise expense analysis

SELECT

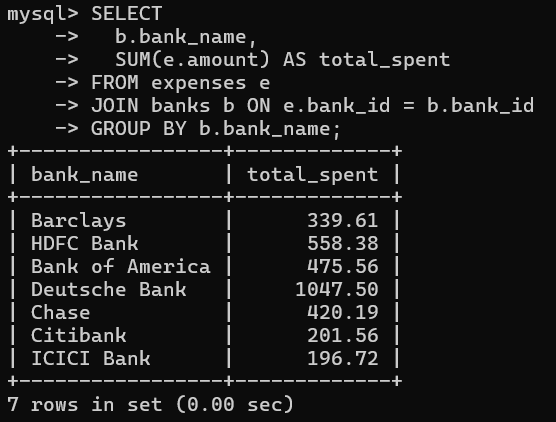
b.bank\_name,

SUM(e.amount) AS total\_spent

FROM expenses e

JOIN banks b ON e.bank\_id = b.bank\_id

GROUP BY b.bank\_name;



1. -- Expenses summary per user

SELECT

u.user\_id,

u.name,

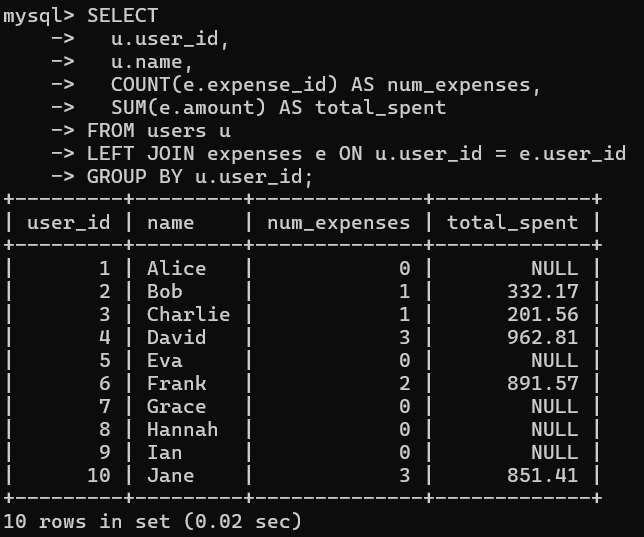
COUNT(e.expense\_id) AS num\_expenses,

SUM(e.amount) AS total\_spent

FROM users u

LEFT JOIN expenses e ON u.user\_id = e.user\_id

GROUP BY u.user\_id;



1. -- Highest spending user

SELECT

u.user\_id,

u.name,

SUM(e.amount) AS total\_spent

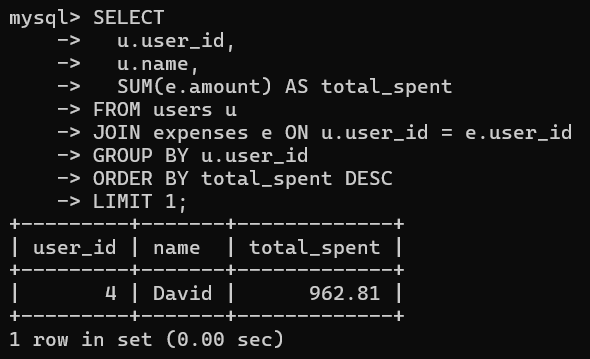
FROM users u

JOIN expenses e ON u.user\_id = e.user\_id

GROUP BY u.user\_id

ORDER BY total\_spent DESC

LIMIT 1;



1. -- User-wise investment amount

SELECT

u.user\_id,

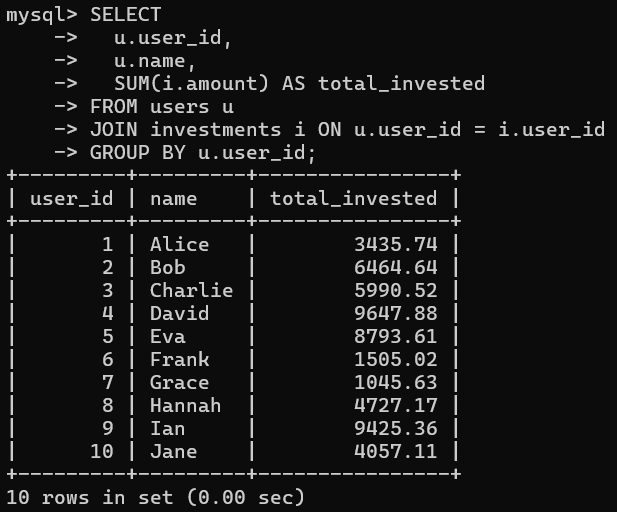
u.name,

SUM(i.amount) AS total\_invested

FROM users u

JOIN investments i ON u.user\_id = i.user\_id

GROUP BY u.user\_id;



1. -- Top 3 expense categories per user

SELECT

user\_id,

category\_id,

total

FROM (

SELECT

user\_id,

category\_id,

SUM(amount) AS total,

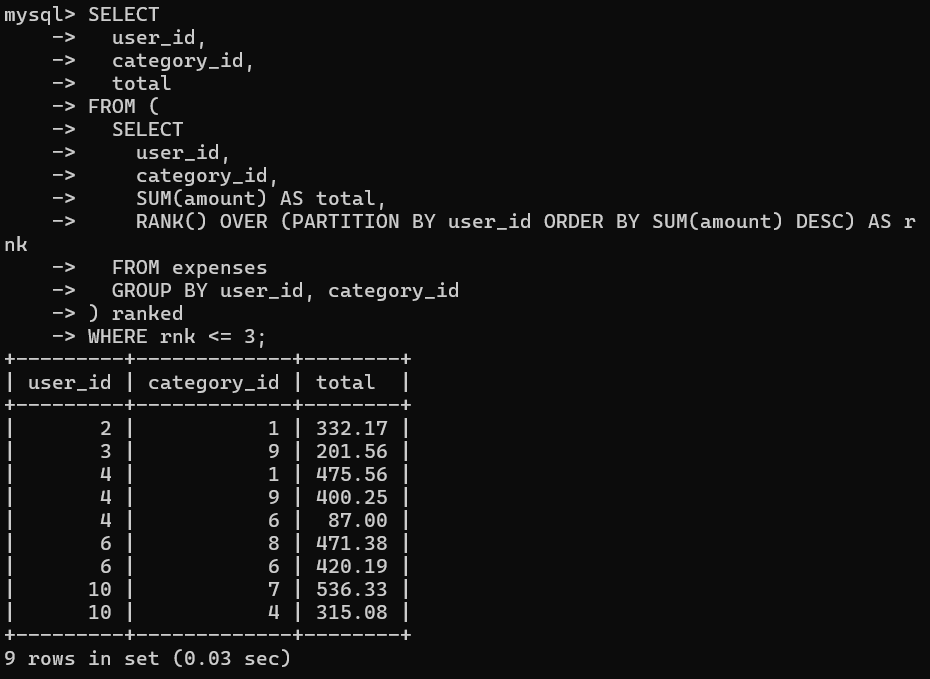
RANK() OVER (PARTITION BY user\_id ORDER BY SUM(amount) DESC) AS rnk

FROM expenses

GROUP BY user\_id, category\_id

) ranked

WHERE rnk <= 3;



1. -- Month-over-month spending change

SELECT

month,

total\_spent,

total\_spent - LAG(total\_spent) OVER (ORDER BY month) AS change

FROM (

SELECT

DATE\_FORMAT(date, '%y-%m') AS month,

SUM(amount) AS total\_spent

FROM expenses

GROUP BY month

) monthly;

-- Total savings per user

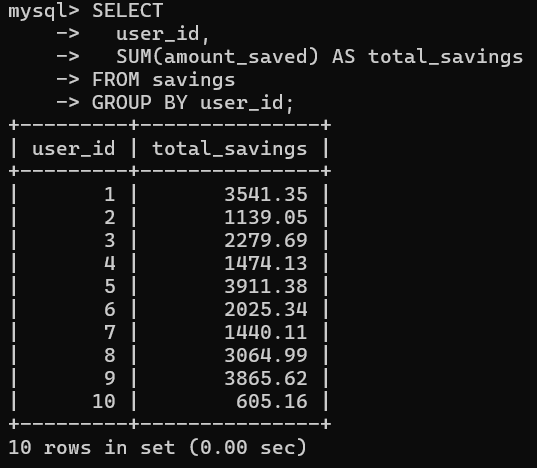
SELECT

user\_id,

SUM(amount\_saved) AS total\_savings

FROM savings

GROUP BY user\_id;



1. -- Savings vs expenses per user

SELECT

s.user\_id,

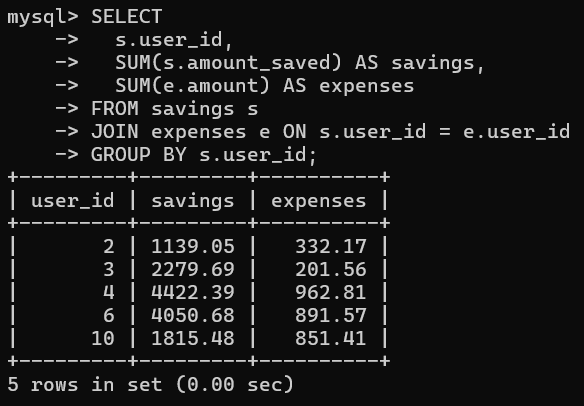
SUM(s.amount\_saved) AS savings,

SUM(e.amount) AS expenses

FROM savings s

JOIN expenses e ON s.user\_id = e.user\_id

GROUP BY s.user\_id;



1. -- Expenses in current year grouped by month

SELECT

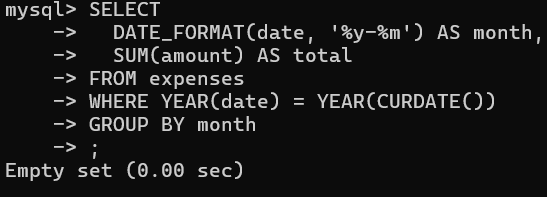
DATE\_FORMAT(date, '%y-%m') AS month,

SUM(amount) AS total

FROM expenses

WHERE YEAR(date) = YEAR(CURDATE())

GROUP BY month;



1. -- Day of week analysis

SELECT

DAYNAME(date) AS weekday,

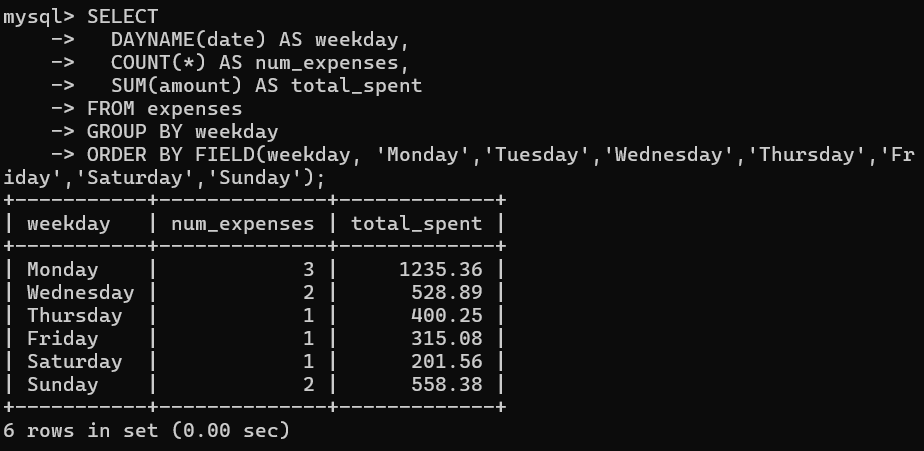
COUNT(\*) AS num\_expenses,

SUM(amount) AS total\_spent

FROM expenses

GROUP BY weekday

ORDER BY FIELD(weekday, 'Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday');



1. -- Expenses between specific dates

SELECT \*

FROM expenses

WHERE date BETWEEN '2024-01-01' AND '2024-03-31';

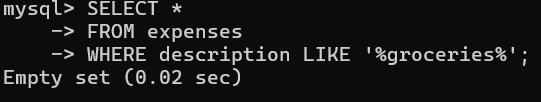


1. -- Filter expenses by keyword in description

SELECT \*

FROM expenses

WHERE description LIKE '%groceries%';

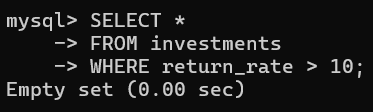


1. -- Investments with return above 10%

SELECT \*

FROM investments

WHERE return\_rate > 10;

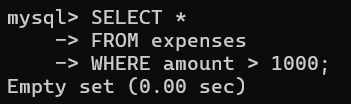


1. -- High-value expenses over 1000

SELECT \*

FROM expenses

WHERE amount > 1000;

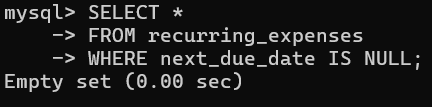


1. -- Inactive recurring expenses

SELECT \*

FROM recurring\_expenses

WHERE next\_due\_date IS NULL;



-- =====================================

-- FULL OUTER JOIN (Expenses + Budgets)

-- =====================================

-- Note: FULL OUTER JOIN is not directly supported in MySQL. Simulated using UNION of LEFT and RIGHT JOINs.

57. SELECT

COALESCE(e.category\_id, b.category\_id) AS category\_id,

c.category\_name,

SUM(e.amount) AS total\_spent,

b.amount AS budget

FROM expenses e

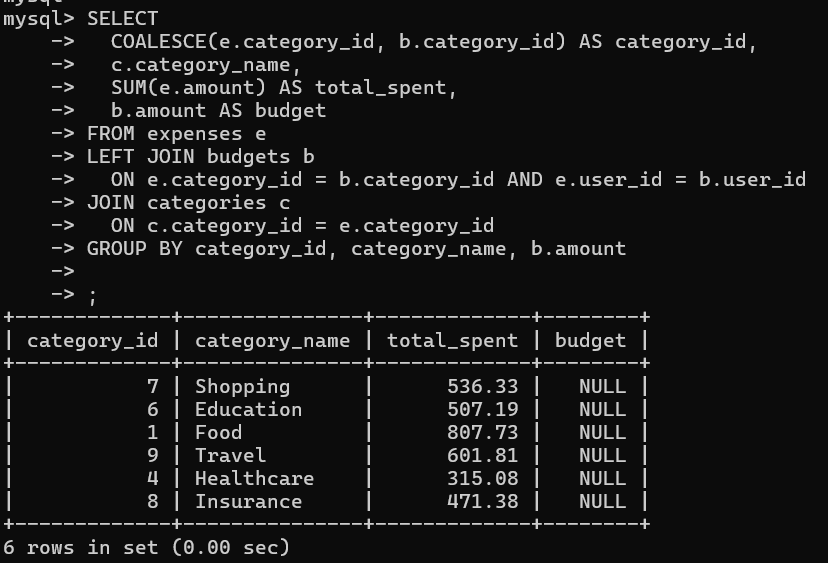
LEFT JOIN budgets b

ON e.category\_id = b.category\_id AND e.user\_id = b.user\_id

JOIN categories c

ON c.category\_id = e.category\_id

GROUP BY category\_id, category\_name, b.amount;



* 1. UNION

SELECT

COALESCE(e.category\_id, b.category\_id) AS category\_id,

c.category\_name,

SUM(e.amount) AS total\_spent,

b.amount AS budget

FROM budgets b

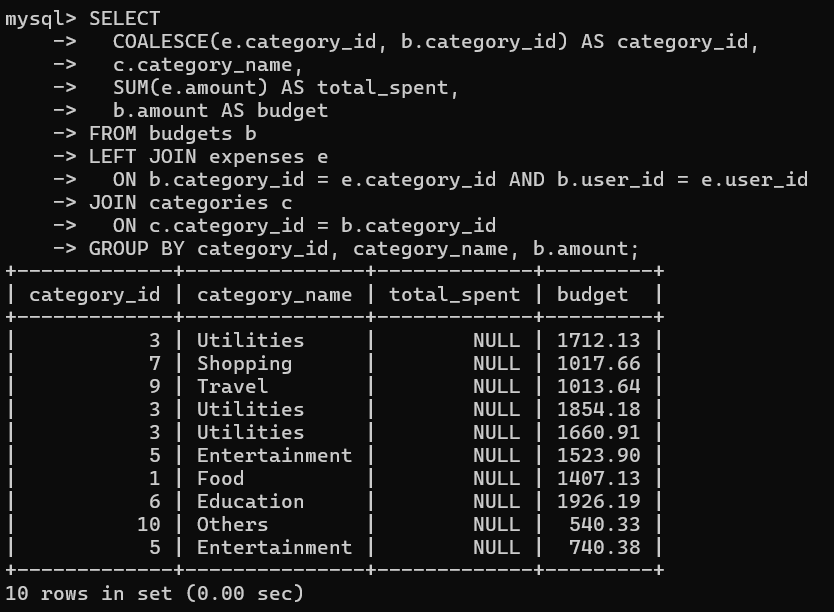
LEFT JOIN expenses e

ON b.category\_id = e.category\_id AND b.user\_id = e.user\_id

JOIN categories c

ON c.category\_id = b.category\_id

GROUP BY category\_id, category\_name, b.amount;



-- =====================================

-- NATURAL JOIN (Categories & Subcategories)

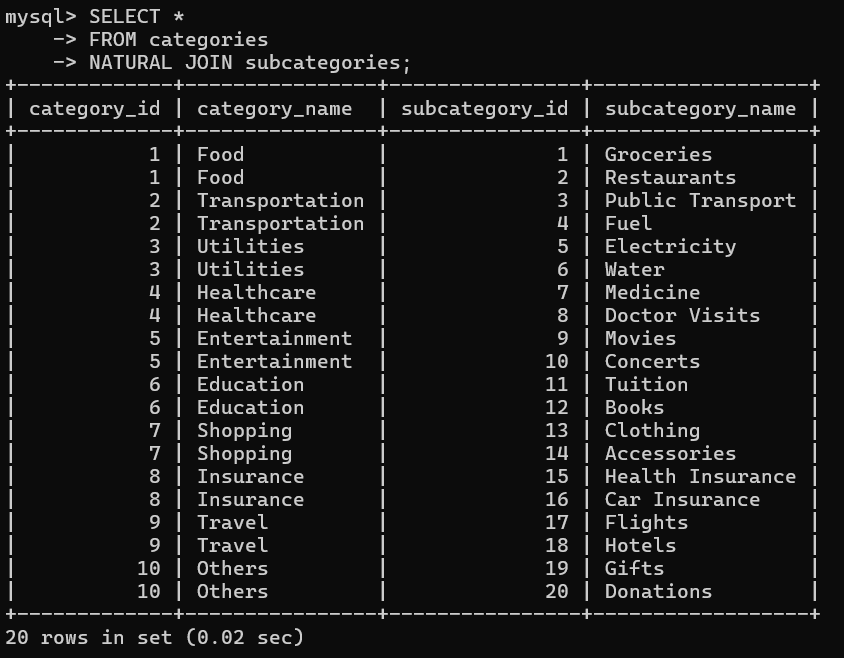
-- =====================================

59. -- Assumes both tables share `category\_id`

SELECT \*

FROM categories

NATURAL JOIN subcategories;



-- =====================================

60. -- THETA JOIN (Custom condition expenses > 2 \* savings)

-- =====================================

SELECT

e.user\_id,

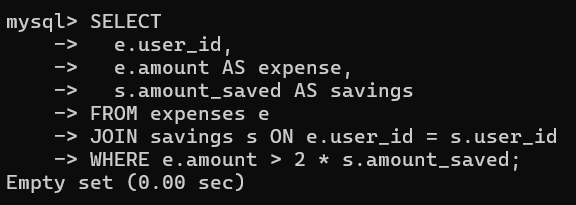
e.amount AS expense,

s.amount\_saved AS savings

FROM expenses e

JOIN savings s ON e.user\_id = s.user\_id

WHERE e.amount > 2 \* s.amount\_saved;



-- =====================================

61. -- MULTI JOIN (Users + Expenses + Categories + Banks + Payment Methods)

-- =====================================

SELECT

u.name,

e.description,

e.amount,

c.category\_name,

p.method\_name,

b.bank\_name,

e.date

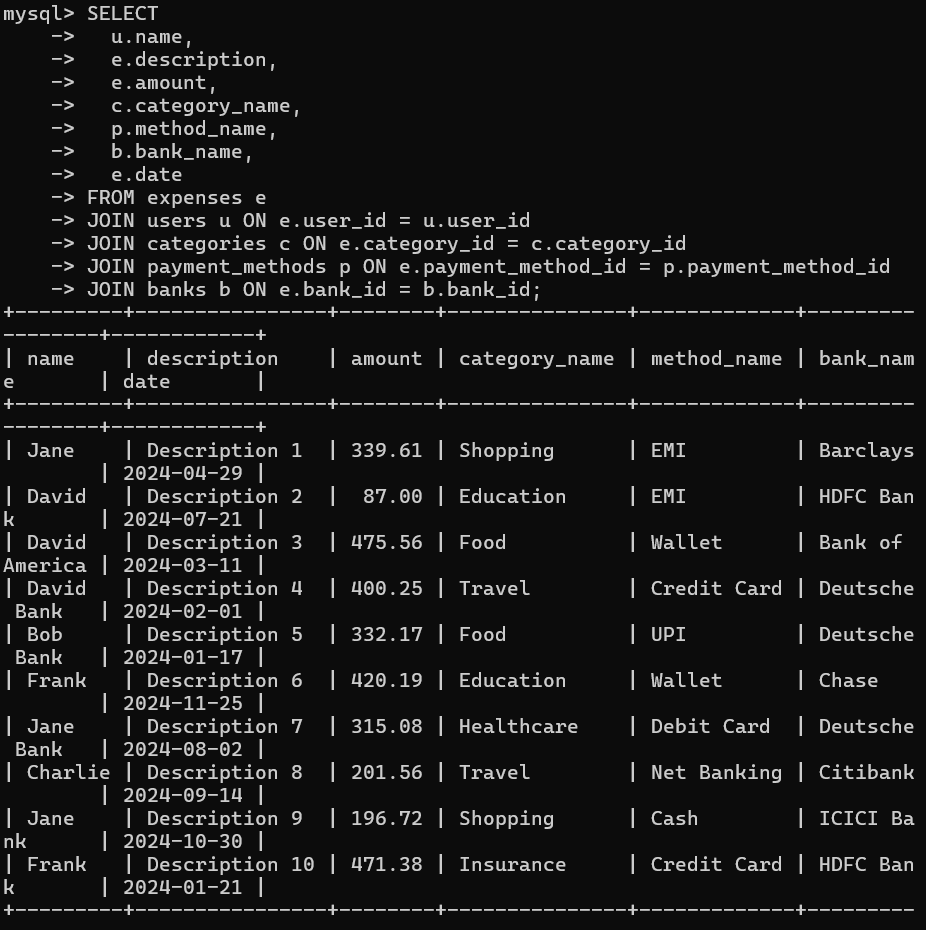
FROM expenses e

JOIN users u ON e.user\_id = u.user\_id

JOIN categories c ON e.category\_id = c.category\_id

JOIN payment\_methods p ON e.payment\_method\_id = p.payment\_method\_id

JOIN banks b ON e.bank\_id = b.bank\_id;

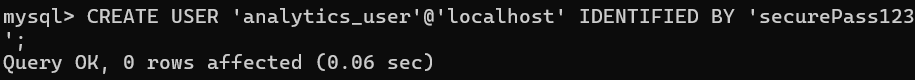
63.

-- =====================================

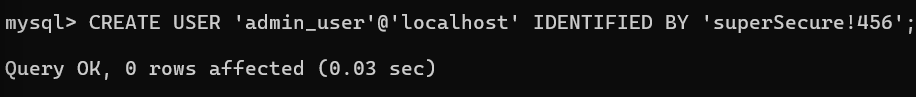
-- CREATING USERS

-- =====================================

62. CREATE USER 'analytics\_user'@'localhost' IDENTIFIED BY 'securePass123';



63. CREATE USER 'admin\_user'@'localhost' IDENTIFIED BY 'superSecure!456';



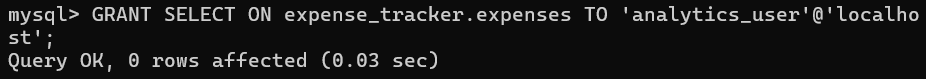
-- =====================================

-- GRANTING PRIVILEGES

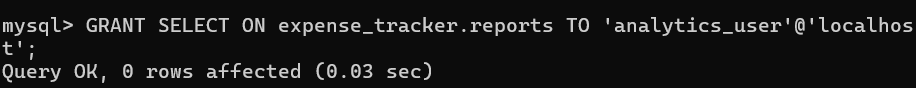
-- =====================================

-- Read-only user

64. GRANT SELECT ON expense\_tracker.expenses TO 'analytics\_user'@'localhost';

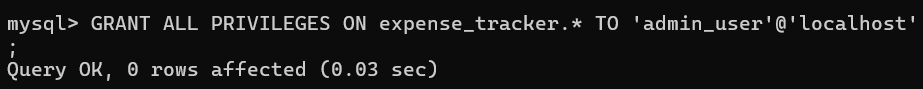


65. GRANT SELECT ON expense\_tracker.reports TO 'analytics\_user'@'localhost';



-- Admin user with full access

66. GRANT ALL PRIVILEGES ON expense\_tracker.\* TO 'admin\_user'@'localhost';

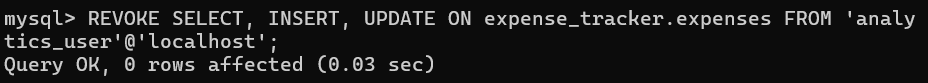


-- =====================================

-- REVOKING ACCESS

-- =====================================

67. REVOKE SELECT, INSERT, UPDATE ON expense\_tracker.expenses FROM 'analytics\_user'@'localhost';



-- =====================================

-- TRANSACTION FOR SENSITIVE INFORMATION

-- =====================================

START TRANSACTION;

-- Credit usage (sensitive)

INSERT INTO credit (user\_id, amount, date\_used)

VALUES (3, 2500.00, 25 Dec 2024);

-- Matching investment entry

INSERT INTO investments (user\_id, investment\_type, amount, return\_rate, investment\_date)

VALUES (3, 'Mutual Fund', 2500.00, 9.5, CURDATE());

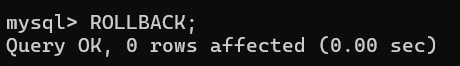
68. -- Commit both only if successful

COMMIT;



-- In case of error

69. -- ROLLBACK;



**VI. Project demonstration**

**Objective of the Demonstration:**

The main objective of this project demonstration is to showcase how the *Expense Tracker* system works, how the database is designed, and how users can easily manage their daily expenses using this system.

**Demonstration Highlights:**

**1. Introduction of the Project**

* Brief explanation of the purpose of the *Expense Tracker*.
* Importance of tracking expenses in daily life.
* Overview of DBMS concepts applied in the project.

**2. Database Design**

* Explanation of Entity-Relationship (ER) Diagram.
* Tables created in the database:
  + User Table
  + Expense Table
  + Category Table
  + Income Table
  + Login Table

**3. Project Modules Demonstrated**

| **Module** | **Features Shown in Demo** |
| --- | --- |
| User Registration/Login | User registration and secure login system with password authentication. |
| Add Expense | Users can add daily expenses with details like date, amount, category, and description. |
| View Expenses | Display of all expenses stored in the database with filters for date/category. |
| Update/Delete Expense | Edit or delete any existing expense record. |
| Add Income | Users can add income details. |
| View Reports | Generate monthly or category-wise reports showing expense summaries and balances. |

**4. SQL Queries Demonstrated**

* Insert, Update, Delete, and Select queries.
* Aggregate functions for generating reports (SUM, COUNT, MAX, MIN).
* Filtering data using WHERE and GROUP BY clauses.

**5. User Interface Demo**

* User-friendly interface to interact with the database.
* Easy navigation for users to add, view, and manage expenses.

**6. Output Display**

* Display of results after each operation (expense added successfully, report generated, error handling, etc.).

**7. Conclusion of Demonstration**

* Final output showing how the Expense Tracker system helps users manage their finances effectively.
* Future enhancements that can be done (like adding charts, mobile app integration, etc.).

**VII. Self -Learning beyond classroom**

:

Working on the *Expense Tracker* project provided us with an opportunity to explore and learn several concepts beyond the classroom syllabus. It enhanced both our technical knowledge and practical skills required for real-world project development.

**Key Areas of Self-Learning:**

**1. Advanced SQL Concepts**

* Learned advanced SQL commands like GROUP BY, HAVING, JOIN, SUBQUERIES, and AGGREGATE FUNCTIONS for better data analysis and report generation.
* Explored the use of triggers and stored procedures for automating tasks in the database.

**2. Database Connectivity**

* Understood how to connect front-end applications (using tools like PHP, Python, Java, etc.) with the back-end database.
* Learned about CRUD operations through user interfaces.

**3. Data Security Techniques**

* Researched techniques for implementing user authentication, password encryption, and data privacy in the system.

**4. User Interface Design**

* Explored how to create user-friendly interfaces to input and display expense data effectively.

**5. Real-Time Data Handling**

* Learned how real-time systems handle user inputs, store them in databases, and reflect changes dynamically.

**6. Project Documentation & Report Writing**

* Developed skills in preparing professional project documentation, including diagrams, modules explanation, challenges faced, and conclusions.

**7. Time Management & Team Collaboration**

* Learned to manage time effectively for completing different phases of the project.
* Understood the importance of collaboration and teamwork while working on a group project.

**8. Exposure to Financial Management Tools**

* Explored existing expense management tools and apps to understand their features and functionalities for inspiration.

**VIII. Learning from the Project**

This *Expense Tracker* project provided us with several valuable learnings, both from technical and practical perspectives. It enhanced our understanding of database systems and improved our problem-solving abilities.

Key Learnings from the Project:

1. Practical Implementation of DBMS Concepts

* Gained hands-on experience in designing and creating databases using SQL.
* Understood the importance of normalization in avoiding data redundancy.
* Learned about primary keys, foreign keys, constraints, and relationships between tables.

2. ER Diagram Design Skills

* Learned how to represent real-world scenarios using Entity-Relationship diagrams.
* Understood how to structure data logically for better storage and retrieval.

3. Query Writing & Optimization

* Improved skills in writing complex SQL queries for various operations like data insertion, updation, deletion, and retrieval.
* Gained knowledge about optimizing queries for faster performance.

4. Data Security Awareness

* Understood the importance of user authentication and secure handling of sensitive data like expenses and personal details.

5. Report Generation Techniques

* Learned how to generate dynamic reports and summaries based on user inputs using SQL queries.

6. Problem Solving & Debugging Skills

* Enhanced the ability to identify errors and debug the database or application logic efficiently.

7. Time & Project Management

* Learned how to plan, design, implement, and test a project within a specific timeline.
* Improved teamwork, task division, and communication skills.

8. Real-World Application Exposure

* Understood how DBMS projects are used in real-world applications like expense management systems, budgeting apps, and financial trackers.

**IX. Challenges Faced**

During the development of the *Expense Tracker* project using DBMS, several challenges were encountered which helped us to learn and improve our problem-solving skills. Some of the key challenges faced are:

**1. Database Design Complexity**

* Designing the correct Entity-Relationship (ER) Diagram and identifying appropriate entities, attributes, and relationships was challenging in the initial phase.
* Ensuring normalization of tables to avoid redundancy and maintain data integrity required careful planning.

**2. Handling Large Data**

* Managing multiple records of daily expenses and categorizing them properly posed a challenge in maintaining database efficiency.

**3. User Authentication & Security**

* Implementing a secure login system and ensuring user data privacy was a critical challenge, especially while storing sensitive financial information.

**4. Query Optimization**

* Writing optimized SQL queries for faster data retrieval, especially when generating reports or filtering expenses over a particular period, required a good understanding of SQL.

**5. Data Validation**

* Validating user inputs like dates, amounts, and categories was necessary to avoid wrong data entry and errors in calculations.

**6. Report Generation**

* Generating detailed expense reports and visual summaries (monthly/weekly expenses, category-wise analysis) required complex queries and logical thinking.

**7. Time Management**

* Balancing the project development with academic schedules and completing all phases like designing, implementation, and testing within the deadline was also a key challenge.

**X. Conclusion**

**Key Takeaways from the Project:**

● The *Expense Tracker* project demonstrated the practical implementation of Database Management System (DBMS) concepts in solving real-life problems related to financial management.

● The project provided hands-on experience in designing and managing databases using SQL, ensuring data storage, retrieval, and manipulation were efficient and secure.

● We learned the importance of database normalization to eliminate redundancy and maintain data integrity.

● The project helped in understanding how to design Entity-Relationship (ER) diagrams for structuring data systematically.

● The implementation of various DBMS operations such as Insert, Update, Delete, and Select queries enhanced the understanding of CRUD operations.

● The project emphasized the significance of data security and user authentication while managing sensitive information like expenses.

● This project helped in building a system that allows users to track their income, expenses, savings, and generate financial reports, which promotes better financial planning.

● It showcased the importance of user-friendly interfaces combined with strong backend database support for a smooth user experience.