Table of Contents

CHAPTER 01: INTRODUCTION TO EXPENSE TRACKER 2

1.1) Preface 2

1.2) Abstract 2

1.3) Applications 3

1.4) Reliability 4

1.5) Advantages and Limitations 5

CHAPTER 02: TECHNICAL IMPLEMENTATION 7

2.1) System Requirements 7

2.2) Installation Of XAMPP 7

2.3) Testing the connection 9

2.4) List of Files (Python) 10

2.5) Database Schema 11

CHAPTER – 03: SOURCE CODE (DATABASE SCHEMA CREATION) 12

3.1) Key Design Considerations 12

3.2) Source Code for EXPENSE_TRACKER.SQL 12

3.3) Testing The Code by Inserting Records in Each Table 13

3.3.2) Insert Data into Expenses Table 13

3.4) INSTANCE OF RELATION 14

3.5) Entity-Relationship Daigram 16

CHAPTER 04: PYTHON PROGRAM SOURCE CODE 17

4.1. database_connection.py 17

4.2. expense_tracker.py 17

4.3. category_manager.py 17

4.4. budget_manager.py 18

4.5. report_generator.py 18

4.6. user_interface.py 19

4.7. config.py (optional) 20

CHAPTER 05: EXECUTING THE PROGRAM 21

CONCLUSION 25

BIBLIOGRAPHY AND REFERENCES 26

CHAPTER 01: INTRODUCTION TO EXPENSE TRACKER

1.1) Preface

The Expense Tracker project is designed to simplify the process of managing personal finances. In today's fast-paced world, keeping track of daily expenditures can be a tedious and time-consuming task. This project provides an easy-to-use platform for individuals to record their daily expenses, categorize them, and maintain a budget for various categories. By leveraging Python and MySQL, the project aims to offer a seamless experience for users who wish to manage their finances effectively and efficiently.

Python, known for its simplicity and readability, is the chosen programming language for this project. The code is written in a straightforward manner, avoiding the use of complex features such as classes or advanced functions. The primary goal is to ensure that the program remains easy to understand, even for beginners. MySQL is employed as the database system to store and retrieve essential data. The integration of Python with MySQL using the `mysql.connector` library ensures that the data is managed efficiently and securely.

The Expense Tracker has several features aimed at streamlining the expense management process. It allows users to input daily expenses, view detailed reports of their spending, and categorize expenses under different predefined categories. Additionally, users can set budgets for each category, helping them maintain control over their spending habits.

Through this project, individuals can gain valuable insights into their spending behavior and make informed decisions to improve their financial health. The Expense Tracker thus serves as a practical tool for anyone looking to keep track of their finances while also providing an opportunity to explore Python programming and database management.

1.2) Abstract

The Expense Tracker project is a financial management tool developed using Python and MySQL. The primary objective of the project is to enable users to easily track their daily expenses, categorize them, and monitor their spending against predefined budgets. This project provides an intuitive interface for managing personal finances by allowing users to input expenses, generate spending reports, and set budget limits for various categories.

The application includes the following key features:

- **A.** Category Management: Users can define and manage expense categories, such as food, transportation, and entertainment.
- **B.** Expense Recording: Users can input daily expenses along with their categories and dates.
- **C.** Budget Tracking: Users can set budget limits for each category, allowing them to monitor their spending and make adjustments as necessary.

D. Expense Reporting: The system generates reports summarizing spending patterns, offering insights into where money is being spent.

1.3) Applications

The Expense Tracker project has a wide range of applications, particularly in personal finance management. Here are some of the key areas where this system can be effectively used: A. Personal Finance Management:

The Expense Tracker allows individuals to keep track of their daily spending and manage their finances more efficiently. By categorizing expenses and setting budgets for various categories, users can stay within their financial limits and make more informed decisions about their spending habits. B. Budget Monitoring:

One of the key applications of the system is helping users track their monthly budgets. By recording expenses under specific categories (such as groceries, entertainment, transportation, etc.), individuals can quickly determine if they are overspending in certain areas. This feature helps users stay on top of their financial goals and make necessary adjustments in real-time. C. Financial Planning:

With the generated spending reports, users can gain insights into their financial behavior, identifying areas where they are spending more than necessary. This helps users plan their finances better and set realistic goals for saving or cutting down on unnecessary expenses. D. Tracking of Recurring Expenses:

The Expense Tracker can be used to monitor recurring expenses like subscriptions, utilities, and loan payments. By regularly tracking these expenses, users can ensure they are not overlooked and can plan their budget accordingly to accommodate these fixed costs. E. Small Businesses or Freelancers:

Freelancers or small business owners can use the Expense Tracker to manage their operational costs. By categorizing business-related expenses separately from personal expenses, they can get a clearer view of their business financial health. This can be helpful for budgeting, tax filing, and profitability analysis. F. Educational Use:

The system can also be used as an educational tool for teaching the basics of personal finance management. Students can learn about categorizing expenses, setting budgets, and generating financial reports using simple technology. G. Improving Financial Discipline:

The Expense Tracker aids in developing good financial habits by encouraging users to consistently track their spending. This continuous monitoring can lead to improved financial discipline, helping users avoid unnecessary debts and savings goals.

1.4) Reliability

The Expense Tracker project is designed with a focus on reliability to ensure accurate tracking and management of financial data. Below are the key aspects that contribute to the reliability of the sysem:

i. Stable Data Management:

a. The integration of Python with MySQL via the `mysql.connector` library ensures a stable and reliable connection between the application and the database. This allows for smooth data storage, retrieval, and updates without significant risk of data corruption.

b. The use of MySQL as a relational database management system ensures that data is stored in an organized and consistent manner. This minimizes the chances of data loss or inconsistency, particularly when multiple users access or update the database.

ii. Data Integrity:

The project uses basic SQL queries to interact with the database, ensuring that transactions such as adding, updating, or deleting expenses are performed with accuracy. Since the application avoids the use of advanced features like subqueries or transaction processing, it minimizes the risk of complex errors and ensures that operations are straightforward and easy to debug.

Data integrity is maintained through careful design of the tables (Categories, Expenses, and Budgets), which are logically structured to ensure that relationships between the data (e.g., categories and expenses) are clear and reliable.

iii. Error Handling:

The system includes basic error handling to prevent crashes or data corruption in case of unexpected inputs or database connection issues. Users are alerted when there is an issue, allowing them to take corrective actions quickly without disrupting the overall functionality of the system.

The application can handle missing or incorrect data inputs (such as invalid expense amounts or missing categories), ensuring that the data remains accurate and valid throughout the process.

iv. Backup and Recovery:

While the project itself does not implement advanced features such as automatic backups, users can manually back up their MySQL database to ensure that critical financial data is not lost. Additionally, recovery procedures can be easily followed to restore the database in case of an unexpected failure.

v. Performance:

The application is designed to be lightweight and efficient, ensuring that it can handle a reasonable number of daily expense entries without performance degradation. Since the system avoids complex queries and large-scale database operations; it remains responsive even when dealing with moderate amounts of data.

6. User Experience:

   The system's user interface (whether command-line or graphical) is designed to be intuitive and error-proof, making it easy for users to input and track expenses without unnecessary complications. Clear prompts and error messages guide users through the process, reducing the chances of incorrect data entry.

7. Scalability:

   Although designed for personal use or small-scale applications, the structure of the database and the simplicity of the code allow the system to be scaled up if needed. Adding new categories, users, or more extensive reporting features can be done with minimal modifications, ensuring that the system remains reliable as it grows.

1.5) Advantages and Limitations

1.5.1 Advantages

A. Simplicity: The Expense Tracker project is designed with a focus on simplicity, ensuring that even users with basic technical knowledge can operate it easily. The system avoids complex features and advanced programming concepts, making it user-friendly and accessible.

B. Effective Budgeting: By allowing users to set budgets for various categories, the system helps individuals manage their finances better and avoid overspending. This feature is useful in controlling spending and encouraging financial discipline.

C. Expense Categorization: The ability to categorize expenses under different headings, such as food, transportation, and entertainment, provides users with a clear overview of where their money is going. This categorization helps in identifying patterns and making adjustments where necessary.

D. Data Reporting: The system generates reports on spending patterns, offering valuable insights into financial behavior. These reports can help users analyze their spending habits and make informed decisions about where to cut costs or save money.

E. Easy Integration with MySQL: The use of MySQL for data storage ensures that the system can handle moderate amounts of financial data effectively. With the `mysql.connector` library, Python interacts seamlessly with the database, ensuring smooth data retrieval and updates.

F. Minimal System Requirements: The system has modest hardware and software requirements, which means it can be run on most basic computers or laptops. This makes it easily accessible to users with a variety of devices and configurations.

1.5.2 Limitations

A. No Automatic Backup: The system does not implement automatic data backup or recovery mechanisms. Users need to manually back up their data, which increases the risk of data loss if something goes wrong (e.g., system failure or accidental deletion).

B. Limited User Interaction: The project is designed to be used by a single user at a time, and it lacks features such as multi-user support or role-based access, which would be useful in a shared environment or for small business use.

C. Limited Security Features: The project does not include advanced security features such as encryption, authentication, or authorization, which would be necessary if the system were to store sensitive financial information. As a result, the system may not be suitable for use in high-security environments.

D. Manual Data Entry: Users must manually enter each expense, which can be time-consuming. The system does not offer features like receipt scanning or integration with banking APIs, which would streamline data input and make it more efficient.

CHAPTER 02: TECHNICAL IMPLEMENTATION

2.1) System Requirements

To successfully implement and run the "Expense Tracker" the following hardware and software configurations are recommended:

2.1.1) Hardware Requirements

a) Processor: Dual-core processor or higher (Intel Core i3 or equivalent).

b) RAM: Minimum 4 GB (8 GB recommended for better performance).

c) Storage: At least 10 GB of free disk space to store the system files and database.

d) Display: Monitor with a resolution of 1366x768 or higher.

e) Input Devices: Keyboard and mouse for interaction.

f) Other Peripherals: A printer for printing bills and reports (optional).

2.1.2) Software Requirements

a) Operating System:

- o Windows 8, 10 or later o Linux (Ubuntu 20.04 or later)
- o macOS (optional, with Python and MySQL support)

b) Programming Environment:

- o Python 3.8 or later installed on the system.

c) Database Management System: o XAMPP to install MySQL and APACHE server.

d) Required Python Libraries:

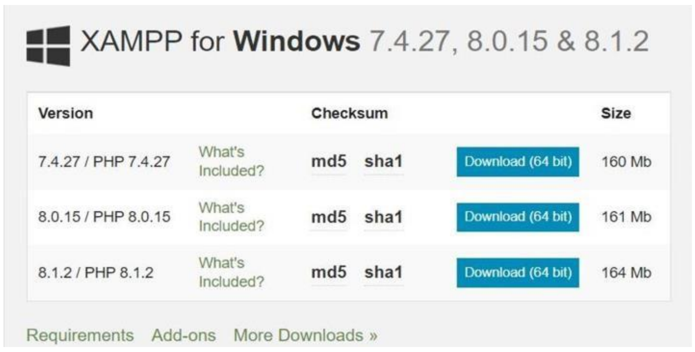
- o `mysql-connector-python` (for connecting Python to MySQL)
- o `math, statistics, csv` and other libraries to implementation of filters. e) Code Editor (Optional):
- o Visual Studio Code, PyCharm, or any text editor for writing and managing Python scripts.

2.2) Installation Of XAMPP

XAMPP is a free, open-source software package that provides a local server environment to run MySQL and other components required for web and database-based projects. Follow the steps below to install and configure XAMPP for the "E-Learning Portal Project":

Step 1: Download XAMPP

1. Visit the official XAMPP website: https://www.apachefriends.org.
2. Download the version compatible with the operating system (Windows, Linux, or macOS).



XAMPP for **Windows** 7.4.27, 8.0.15 & 8.1.2

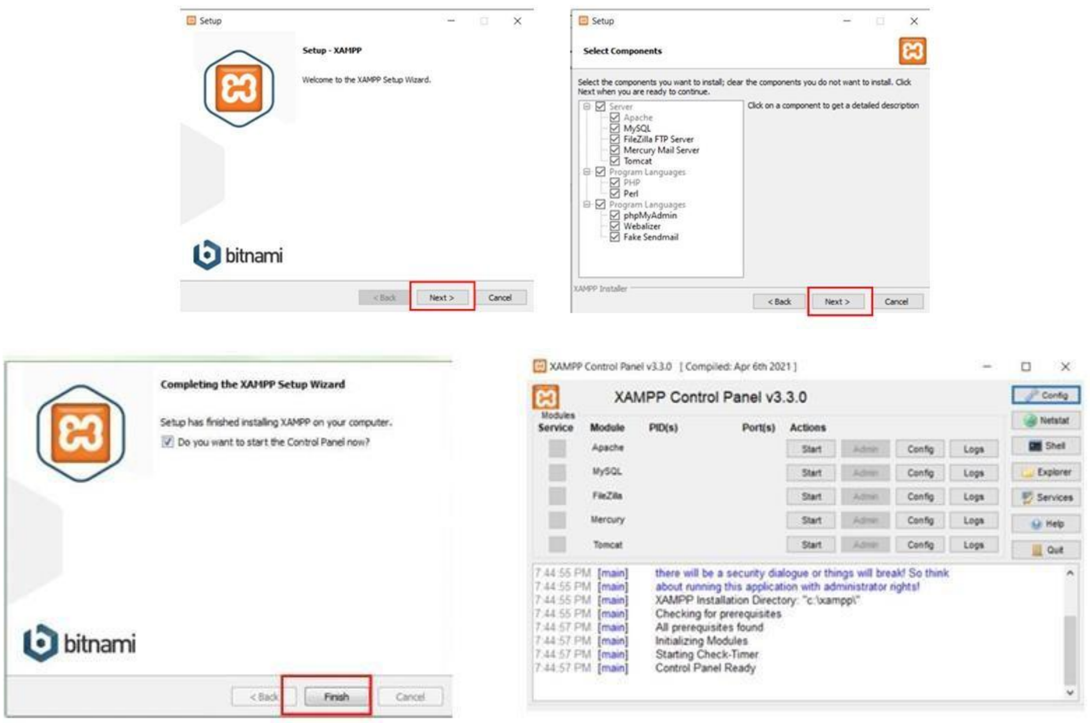
Version		Checksum			Size
7.4.27 / PHP 7.4.27	What's Included?	md5	sha1	Download (64 bit)	160 Mb
8.0.15 / PHP 8.0.15	What's Included?	md5	sha1	Download (64 bit)	161 Mb
8.1.2 / PHP 8.1.2	What's Included?	md5	sha1	Download (64 bit)	164 Mb

Requirements Add-ons More Downloads »

Fig. 2.1: XAMPP for Windows

Step 2: Install XAMPP

1. Run the downloaded installer file.
2. Follow the on-screen instructions in the installation wizard:
    - Select the components to install. For this project, ensure MySQL is selected.
    - Choose the installation directory (default is recommended).
3. Complete the installation by clicking "Finish."



Setup - XAMPP

Welcome to the XAMPP Setup Wizard.

Select Components

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

Click on a component to get a detailed description

- [✓] Server
  - [✓] Apache
  - [✓] MySQL
  - [✓] FileZilla FTP Server
  - [✓] Mercury Mail Server
  - [✓] Tomcat
- [✓] Program Languages
  - [✓] PHP
  - [✓] Perl
- [✓] Program Languages
  - [✓] phpMyAdmin
  - [✓] Webalizer
  - [✓] Fake Sendmail

Completing the XAMPP Setup Wizard

Setup has finished installing XAMPP on your computer.

[✓] Do you want to start the Control Panel now?

XAMPP Control Panel v3.3.0 [Compiled: Apr 6th 2021]

Service	Module	PID(s)	Port(s)	Actions			
	Apache			Start	Admin	Config	Logs
	MySQL			Start	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

```
7:44:55 PM  [main]   there will be a security dialogue or things will break! So think
7:44:55 PM  [main]   about running this application with administrator rights!
7:44:55 PM  [main]   XAMPP Installation Directory: "c:\xampp\"
7:44:55 PM  [main]   Checking for prerequisites
7:44:57 PM  [main]   All prerequisites found
7:44:57 PM  [main]   Initializing Modules
7:44:57 PM  [main]   Starting Check-Timer
7:44:57 PM  [main]   Control Panel Ready
```

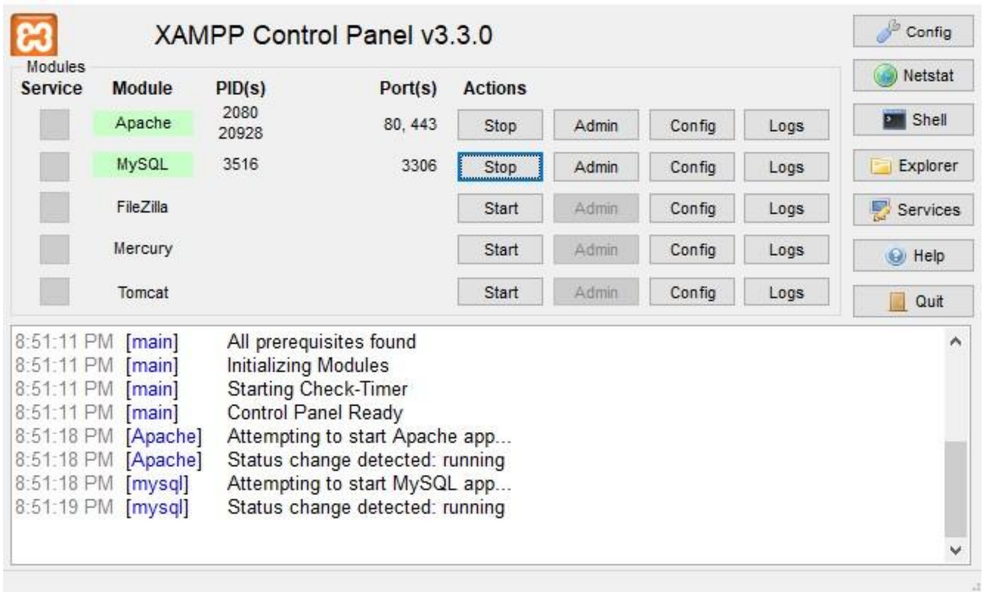
Fig. 2.2: XAMPP installation Process

Step 3: Start XAMPP

1. Open the XAMPP Control Panel.
2. Start the MySQL service by clicking the "Start" button next to it.
3. Verify that MySQL is running (a green status indicator will appear).

Step 4: Access phpMyAdmin

1. Open a web browser and navigate to `http://localhost/phpmyadmin.`
2. Use phpMyAdmin to create and manage the database for the project.



Service	Module	PID(s)	Port(s)	Actions			
	Apache	2080 20928	80, 443	Stop	Admin	Config	Logs
	MySQL	3516	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

```
8:51:11 PM  [main]     All prerequisites found
8:51:11 PM  [main]     Initializing Modules
8:51:11 PM  [main]     Starting Check-Timer
8:51:11 PM  [main]     Control Panel Ready
8:51:18 PM  [Apache]   Attempting to start Apache app...
8:51:18 PM  [Apache]   Status change detected: running
8:51:18 PM  [mysql]    Attempting to start MySQL app...
8:51:19 PM  [mysql]    Status change detected: running
```

Fig. 2.3: Starting APACHE and MySQL sever

2.3) Testing the connection

To test the connection between Python and MySQL, we can use the `mysql-connector-python` library.

2.3.1. Install the MySQL Connector

```
pip install mysql-connector-python
```

2.3.2. Code to Test Connection

```
import mysql.connector from
mysql.connector import Error

try
:
    # Establish a connection
connection = mysql.connector.connect(
host="localhost",
user="root",
password=""
    )      if connection.is_connected():
print("Connection to MySQL was successful!")
        # Print MySQL server details
db_info = connection.get_server_info()
print(f"MySQL Server version: {db_info}")
 except Error as
e:
    print(f"Error connecting to MySQL: {e}") finally:     if
'connection' in locals() and connection.is_connected():
        connection.close()
print("MySQL connection is closed.")
```

2.3.3.

Output If successful:

```
Connection to MySQL was successful!
MySQL Server version: 8.0.1
MySQL connection is closed.
```

If unsuccessful, it prints the error message.

2.4) List of Files (Python)

```
ExpenseTracker/
├── Database/            # Database-related files
├── PythonFunctions/       # Python scripts for functionality
├── expenseReport.csv      # Sample/exported expense report
├── testing.py           # Test scripts
├── user_interface.py      # CLI for interacting with system
├── README.md            # Project README
└── LICENSE             # MIT License file
```

a) `database_connection.py`

- Purpose: Handles the connection between Python and MySQL using the `mysql.connector` library.
- Content: Establishes a connection to the database with credentials like `localhost`, `root`, " ", and `expensetracker`.

b) `expense_tracker.py`

- Purpose: Main program to interact with the user, manage expenses, categories, and budgets.
- Content: Includes functions to add expenses, view reports, and interact with the database.

c) `category_manager.py`

- Purpose: Manages the categories for expenses (adding and viewing categories).
- Content: Includes functions to insert new categories and fetch existing categories from the database.

d) `budget_manager.py`

- Purpose: Manages the budgets for each category.
- Content: Functions to set a budget for a category, and track whether the user is within the budget.

e) `report_generator.py`

- Purpose: Generates reports for the user's expenses.
- Content: Includes functions to generate summaries of expenses and spending patterns.

f) `user_interface.py`

- Purpose: Provides the user interface (CLI or GUI) for interacting with the Expense Tracker.
- Content: Allows users to input expenses, view reports, and manage budgets and categories.

g) `config.py`

- Purpose: Holds configuration variables like database credentials, host, etc.
- Content: A simple configuration file to store variables that can be imported into other files.

2.5) Database Schema

A. Categories Table

This table stores the different categories under which expenses can be classified (e.g., food, transportation, entertainment, etc.).

Field Name	Data Type	Description
`CategoryID`	`INT`	Primary key, auto-incrementing unique identifier for each category.
`CategoryName`	`VARCHAR(255)`	Name of the category (e.g., "Food", "Transport").

B. Expenses Table

This table stores individual expenses, including the amount, associated category, and the date of the expense.

Field Name	Data Type	Description
`ExpenseID`	`INT`	Primary key, auto-incrementing unique identifier for each expense.
`Amount`	`DECIMAL(10, 2)`	The amount of the expense (e.g., 100.50).
`CategoryID`	`INT`	Foreign key linking to `Categories` table. Refers to the category under which the expense falls.
`Date`	`DATE`	The date the expense occurred (e.g., 2025-01-11).

C. Budgets Table

This table stores the budget amounts set for each user in each category. It helps track the budget limits for different categories and users.

Field Name	Data Type	Description
`UserID`	`INT`	Foreign key linking to the user (could be a placeholder for now).
`CategoryID`	`INT`	Foreign key linking to `Categories` table. Refers to the category for which the budget is set.
`BudgetAmount`	`DECIMAL(10,2)`	The set budget amount for the given category (e.g., 500.00).

Entity Relationship Diagram (ER Diagram Overview):

a) Categories: One category can have many expenses, but each expense belongs to only one category.

b) Expenses: Each expense belongs to a specific category.

c) Budgets: A user can have a budget set for each category. The primary key is a combination of `UserID` and

`CategoryID`.

CHAPTER – 03: SOURCE CODE (DATABASE SCHEMA CREATION)

3.1) Key Design Considerations

a) Normalization

o Tables are normalized to avoid data redundancy and maintain consistency.

b) Indexes

o Primary keys ensure quick look up for each table.
o Foreign keys maintain data integrity and relationships.

c) Scalability

o The design can handle increasing data volumes by adding more rows without redesigning the schema.

d) Data Integrity

o Foreign keys and data types ensure data consistency.

3.2) Source Code for EXPENSE_TRACKER.SQL

```
-- Active:736689991548@@127.0.0.1@3306@expensetracker

--creating database
CREATE DATABASE expensetracker;

-- Use the database USE
expensetracker;

--creating table : categories
CREATE TABLE Categories (
    CategoryID INT AUTO_INCREMENT PRIMARY KEY,
    CategoryName VARCHAR(255) NOT NULL
);

--creating table : expenses
CREATE TABLE Expenses (
    ExpenseID INT AUTO_INCREMENT PRIMARY KEY, Amount
    DECIMAL(10, 2) NOT NULL, CategoryID
    INT,
    Date DATE,
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);

--creating table : budgets
CREATE TABLE Budgets (
    UserID INT,
    CategoryID INT,
    BudgetAmount DECIMAL(10, 2) NOT NULL,
    PRIMARY KEY (UserID, CategoryID),
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);
```

3.3) Testing The Code by Inserting Records in Each Table

3.3.1) Insert Data into Categories Table

```
 INSERT INTO Categories (CategoryID, CategoryName) VALUES (1, 'Food'),
(2, 'Transportation'),
(3, 'Utilities'),
(4, 'Entertainment'),
(5, 'Health'),
(6, 'Education'),
(7, 'Shopping'),
(8, 'Travel'),
(9, 'Rent'),
(10, 'Savings'),
(11, 'Insurance'),
(12, 'Gifts'),
(13, 'Dining Out'),
(14, 'Gym Membership'),
(15, 'Internet'),
(16, 'Mobile Recharge'),
(17, 'Fuel'),
(18, 'Miscellaneous'),
(19, 'Home Maintenance'),
(20, 'Subscriptions');
Query OK! Affected Rows: 20
```

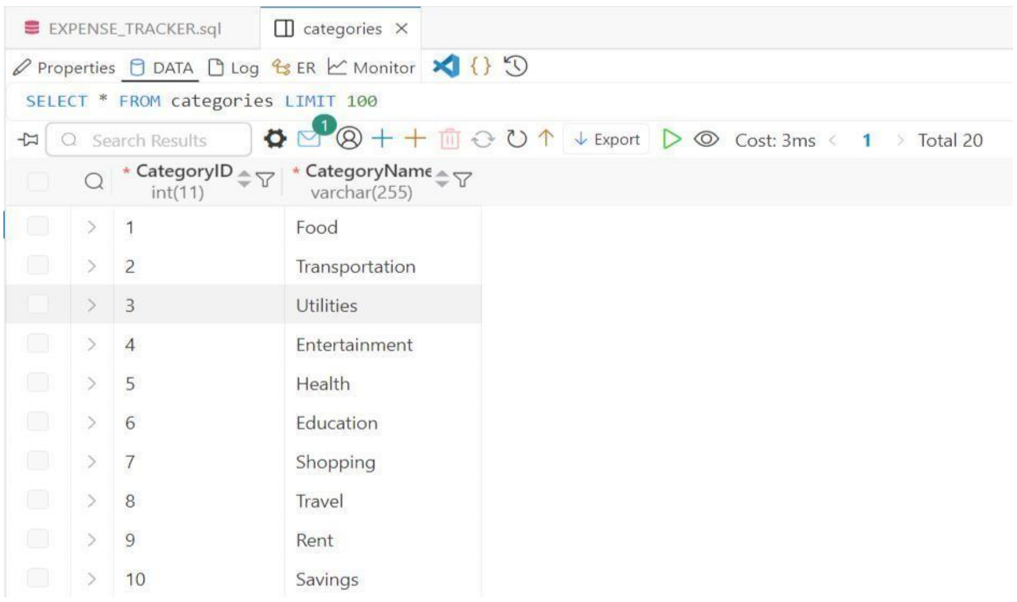
3.3.2) Insert Data into Expenses Table

```
INSERT INTO Expenses (ExpenseID, Amount, CategoryID, Date) VALUES
(1, 200.50, 1, '2025-01-01'),
(2, 50.00, 2, '2025-01-02'),
(3, 300.75, 3, '2025-01-03'),
(4, 150.00, 4, '2025-01-04'),
(5, 500.00, 5, '2025-01-05'),
(6, 600.00, 6, '2025-01-06'),
(7, 250.00, 7, '2025-01-07'),
(8, 800.00, 8, '2025-01-08'),
(9, 1200.00, 9, '2025-01-09'),
(10, 100.00, 10, '2025-01-10'),
(11, 400.00, 11, '2025-01-11'),
(12, 75.00, 12, '2025-01-12'),
(13, 300.00, 13, '2025-01-13'),
(14, 200.00, 14, '2025-01-14'),
(15, 500.00, 15, '2025-01-15'),
(16, 150.00, 16, '2025-01-16'),
(17, 250.00, 17, '2025-01-17'),
(18, 90.00, 18, '2025-01-18'),
(19, 350.00, 19, '2025-01-19'),
(20, 50.00, 20, '2025-01-20');
Query OK! Affected Rows: 20
```

3.3.3) Insert Data into Budgets Table

```
INSERT INTO Budgets (UserID, CategoryID, BudgetAmount) VALUES
(1, 1, 1000.00),
(1, 2, 500.00),
(1, 3, 1500.00),
(1, 4, 800.00),
(1, 5, 2000.00),
(1, 6, 1200.00),
(1, 7, 1000.00),
(1, 8, 3000.00),
(1, 9, 10000.00),
(1, 10, 2000.00),
(1, 11, 1200.00),
(1, 12, 500.00),
(1, 13, 800.00),
(1, 14, 600.00),
(1, 15, 1000.00),
(1, 16, 400.00),
(1, 17, 1500.00),
(1, 18, 300.00),
(1, 19, 700.00),
(1, 20, 100.00);
Query OK! Affected Rows: 20
```

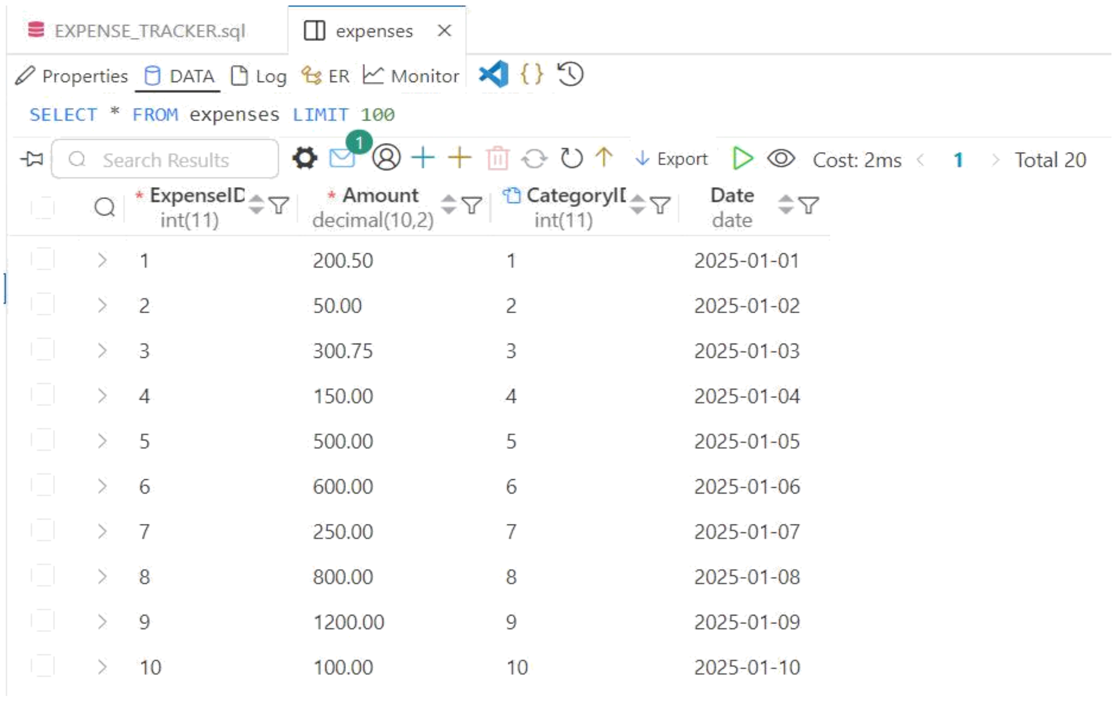
3.4) INSTANCE OF RELATION



```
SELECT * FROM categories LIMIT 100
```

CategoryID int(11)	CategoryName varchar(255)
1	Food
2	Transportation
3	Utilities
4	Entertainment
5	Health
6	Education
7	Shopping
8	Travel
9	Rent
10	Savings

Fig. 3.1: Instance of Categories table

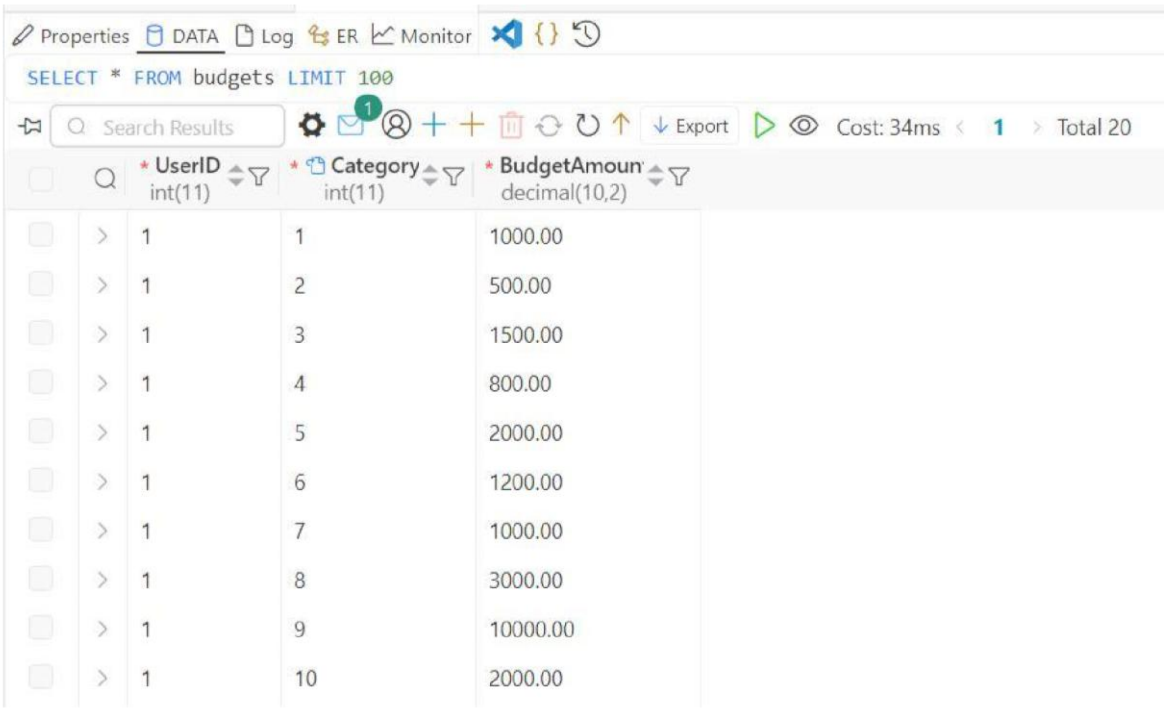


```
SELECT * FROM expenses LIMIT 100
```

Cost: 2ms 1 Total 20

ExpenseID int(11)	Amount decimal(10,2)	CategoryID int(11)	Date date
1	200.50	1	2025-01-01
2	50.00	2	2025-01-02
3	300.75	3	2025-01-03
4	150.00	4	2025-01-04
5	500.00	5	2025-01-05
6	600.00	6	2025-01-06
7	250.00	7	2025-01-07
8	800.00	8	2025-01-08
9	1200.00	9	2025-01-09
10	100.00	10	2025-01-10

Fig. 3.2: Instance of Expenses table



```
SELECT * FROM budgets LIMIT 100
```

Cost: 34ms 1 Total 20

UserID int(11)	Category int(11)	BudgetAmount decimal(10,2)
1	1	1000.00
1	2	500.00
1	3	1500.00
1	4	800.00
1	5	2000.00
1	6	1200.00
1	7	1000.00
1	8	3000.00
1	9	10000.00
1	10	2000.00

Fig. 3.3: Instance of Budget table

3.5) Entity-Relationship Daigram

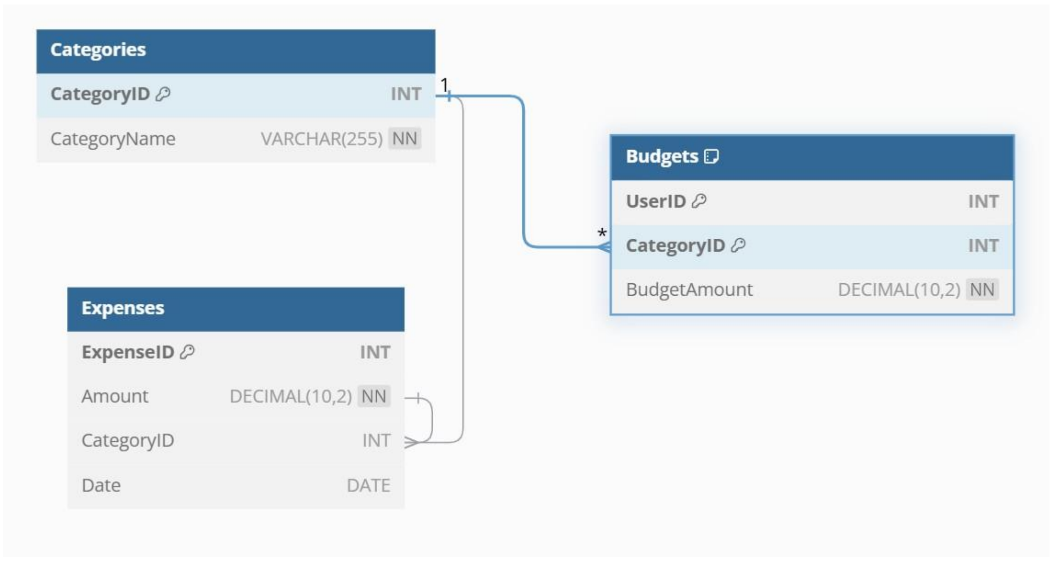


Fig. 3.4: ER Diagram for the relations
Source URL to generate SQL queries into ER Diagram:
https://dbdiagram.io/d/6783fc9e6b7fa355c3a25a10

Relationships:

A. **Categories to Expenses:** One-to-many relationship (A category can have many expenses, but an expense is linked to only one category).

B. Categories to Budgets: One-to-many relationship (A category can have many budgets, but a budget is linked to only one category).

C. Budgets to Users: Many-to-one relationship (Multiple budgets can belong to a single user).

CHAPTER 04: PYTHON PROGRAM SOURCE CODE

4.1. database_connection.py

```
import mysql.connector

# Function to establish connection to the database
def connectToDatabase():     return
mysql.connector.connect(          host="localhost",
user="root",          password="",
database="expensetracker"
    )
```

4.2. expense_tracker.py

```
from database_connection import connectToDatabase

# Function to add a new expense def
addExpense(amount, categoryId, date):
    conn = connectToDatabase()     cursor = conn.cursor()     query
= "INSERT INTO Expenses (Amount, CategoryID, Date) VALUES
(%s, %s, %s)"     cursor.execute(query, (amount,
categoryId, date))     conn.commit()
print("Expenses added Successfully!")
cursor.close()     conn.close()

# Function to view all expenses def
viewExpenses():
    conn = connectToDatabase()     cursor =
conn.cursor()     cursor.execute("SELECT *
FROM Expenses")     expenses =
cursor.fetchall()     cursor.close()
conn.close()     print("Expenses:----------
")     return expenses
```

4.3. category_manager.py

```
from database_connection import connectToDatabase

# Function to add a new category def addCategory(categoryName):
conn = connectToDatabase()     cursor = conn.cursor()     query
= "INSERT INTO Categories (CategoryName) VALUES (%s)"
cursor.execute(query, (categoryName,))     conn.commit()
print(f"Category {categoryName} added successfully!")
cursor.close()     conn.close()

# Function to view all categories def
viewCategories():
    conn = connectToDatabase()     cursor =
conn.cursor()     cursor.execute("SELECT * FROM
Categories")      categories = cursor.fetchall()
print("Following are the Categories: --------")
cursor.close()      conn.close()      return
categories
```

4.4. budget_manager.py

```
from database_connection import connectToDatabase

# Function to set a budget for a category def
setBudget(userId, categoryId, budgetAmount):
    conn = connectToDatabase()
cursor = conn.cursor()
    query = "INSERT INTO Budgets (UserID, CategoryID, BudgetAmount)
VALUES (%s, %s, %s)"      cursor.execute(query, (userId, categoryId,
budgetAmount))      conn.commit()      print("Budget Set
Successfully!")      cursor.close()      conn.close()

# Function to view all budgets for a user
def viewBudgets(userId):
    conn = connectToDatabase()      cursor = conn.cursor()
cursor.execute("SELECT * FROM Budgets WHERE UserID = %s",
(userId,))      budgets = cursor.fetchall()
cursor.close()      conn.close()
print("Following are the budgets: --------")
return budgets
```

4.5. report_generator.py

```
from database_connection import connectToDatabase import
csv
# Function to generate an expense report by category
def generateExpenseReport():      conn =
connectToDatabase()      cursor = conn.cursor()
cursor.execute("""
        SELECT CategoryName, SUM(Amount)
        FROM Expenses
        JOIN Categories ON Expenses.CategoryID =  Categories.CategoryID
        GROUP BY CategoryName
    """)      report = cursor.fetchall()      # Write the rows
into a CSV file      with open("expenseReport.csv", "w",
newline="") as file:
        writer = csv.writer(file)          # Write
the header          writer.writerow(["Category Name",
"Amount"])
        # Write the data rows
writer.writerows(report)
```

```
    print("Data successfully exported to
ExpenseReport.csv")     cursor.close()     conn.close()
return report
```

4.6. user_interface.py

```
from expense_tracker import addExpense, viewExpenses
from category_manager import addCategory, viewCategories
from budget_manager import setBudget, viewBudgets from
report_generator import generateExpenseReport
# Main function to interact with the user
def main():     while True:
        print("Welcome to the Expense Tracker!")
print("1. Add Expense")         print("2. View
Expenses")         print("3. Add Category")
print("4. View Categories")         print("5. Set
Budget")          print("6. View Budgets")
print("7. Generate Report")         print("8. Exit
the Program")         choice = int(input("Enter
your choice: "))
                 if
choice == 1:
            amount = float(input("Enter amount: "))
categoryId = int(input("Enter category ID: "))
date = input("Enter date (YYYY-MM-DD): ")
addExpense(amount, categoryId, date)         elif choice
== 2:
            expenses = viewExpenses()
for expense in expenses:
                print(expense)
elif choice == 3:
            categoryName = input("Enter category name: ")
addCategory(categoryName)          elif choice == 4:
            categories = viewCategories()
for category in categories:
                print(category)
elif choice == 5:
            userId = int(input("Enter user ID: "))
categoryId = int(input("Enter category ID: "))
budgetAmount = float(input("Enter budget amount: "))
setBudget(userId, categoryId, budgetAmount)         elif choice
== 6:
            userId = int(input("Enter user ID: "))
budgets = viewBudgets(userId)              for
budget in budgets:
                print(budget)
elif choice == 7:
            report = generateExpenseReport()
# for entry in report:
            #     print(entry)
        elif choice == 8:
            print("Exiting the Program\nGood Bye!")
break          else:
            print("Invalid Choice Please Try again!!")
# "__main__":
main()
```

4.7. config.py (optional)

```
DB_HOST  = "localhost"
DB_USER  = "root"
DB_PASSWORD     = ""
DB_NAME = "expensetracker"
```

CHAPTER 05: EXECUTING THE PROGRAM

```
PS C:\Users\Sagar\OneDrive\Desktop\ExpenseTracker> python -u
"C:\Users\Sagar\OneDrive\Desktop\ExpenseTracker\user_interface.py"
Welcome to the Expense Tracker!
1. Add Expense
2. View Expenses
3. Add Category
4. View Categories
5. Set Budget
6. View Budgets
7. Generate Report
8. Exit the Program
Enter your choice: 2
Expenses: ----------
(1, Decimal('200.50'), 1, datetime.date(2025, 1, 1))
(2, Decimal('50.00'), 2, datetime.date(2025, 1, 2))
(3, Decimal('300.75'), 3, datetime.date(2025, 1, 3))
(4, Decimal('150.00'), 4, datetime.date(2025, 1, 4))
(5, Decimal('500.00'), 5, datetime.date(2025, 1, 5))
(6, Decimal('600.00'), 6, datetime.date(2025, 1, 6))
(7, Decimal('250.00'), 7, datetime.date(2025, 1, 7))
(8, Decimal('800.00'), 8, datetime.date(2025, 1, 8))
(9, Decimal('1200.00'), 9, datetime.date(2025, 1, 9))
(10, Decimal('100.00'), 10, datetime.date(2025, 1, 10))
(11, Decimal('400.00'), 11, datetime.date(2025, 1, 11))
(12, Decimal('75.00'), 12, datetime.date(2025, 1, 12))
(13, Decimal('300.00'), 13, datetime.date(2025, 1, 13))
(14, Decimal('200.00'), 14, datetime.date(2025, 1, 14))
(15, Decimal('500.00'), 15, datetime.date(2025, 1, 15))
(16, Decimal('150.00'), 16, datetime.date(2025, 1, 16))
(17, Decimal('250.00'), 17, datetime.date(2025, 1, 17))
(18, Decimal('90.00'), 18, datetime.date(2025, 1, 18))
(19, Decimal('350.00'), 19, datetime.date(2025, 1, 19))
(20, Decimal('50.00'), 20, datetime.date(2025, 1, 20))
(21, Decimal('1500.00'), 4, datetime.date(2024, 1, 19))

Welcome to the Expense Tracker!
1. Add Expense
2. View Expenses
3. Add Category
4. View Categories
5. Set Budget
6. View Budgets
7. Generate Report
8. Exit the Program
Enter your choice: 4
Following are the Categories: -------
(1, 'Food')
(2, 'Transportation')
(3, 'Utilities')
(4, 'Entertainment')
```

```
(5, 'Health')
(6, 'Education')
(7, 'Shopping')
(8, 'Travel')
(9, 'Rent')
(10, 'Savings')
(11, 'Insurance')
(12, 'Gifts')
(13, 'Dining Out')
(14, 'Gym Membership')
(15, 'Internet')
(16, 'Mobile Recharge')
(17, 'Fuel')
(18, 'Miscellaneous')
(19, 'Home Maintenance')
(20, 'Subscriptions')

Welcome to the Expense Tracker!
1. Add Expense
2. View Expenses
3. Add Category
4. View Categories
5. Set Budget
6. View Budgets
7. Generate Report
8. Exit the Program
Enter your choice: 6
Enter user ID: 1
Following are the budgets: --------
(1, 1, Decimal('1000.00')) (1,
2, Decimal('500.00'))
(1, 3, Decimal('1500.00')) (1,
4, Decimal('800.00'))
(1, 5, Decimal('2000.00'))
(1, 6, Decimal('1200.00'))
(1, 7, Decimal('1000.00'))
(1, 8, Decimal('3000.00'))
(1, 9, Decimal('10000.00'))
(1, 10, Decimal('2000.00'))
(1, 11, Decimal('1200.00'))
(1, 12, Decimal('500.00'))
(1, 13, Decimal('800.00'))
(1, 14, Decimal('600.00'))
(1, 15, Decimal('1000.00')) (1,
16, Decimal('400.00'))
(1, 17, Decimal('1500.00'))
(1, 18, Decimal('300.00'))
(1, 19, Decimal('700.00'))
(1, 20, Decimal('100.00'))

Welcome to the Expense Tracker!
```

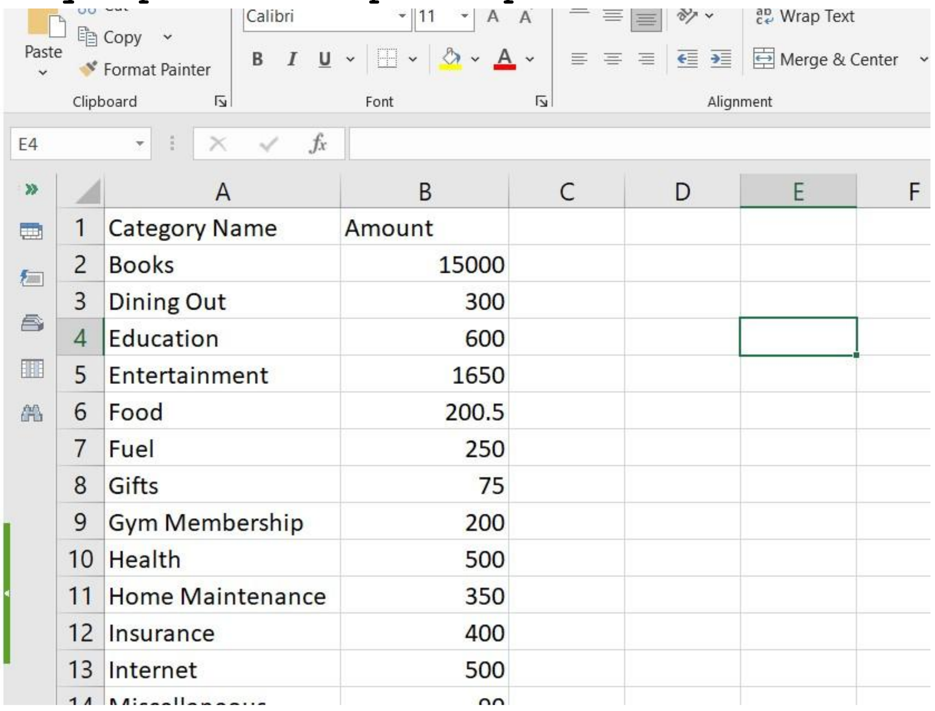
```
1. Add Expense
2. View Expenses
3. Add Category
4. View Categories
5. Set Budget
6. View Budgets
7. Generate Report
8. Exit the Program
Enter your choice: 3
Enter category name: Books Category
Books added successfully!

Welcome to the Expense Tracker!
1. Add Expense
2. View Expenses
3. Add Category
4. View Categories
5. Set Budget
6. View Budgets
7. Generate Report
8. Exit the Program
Enter your choice: 5
Enter user ID: 21
Enter category ID: 21 Enter
budget amount: 15000
Budget Set Successfully!

Welcome to the Expense Tracker!
1. Add Expense
2. View Expenses
3. Add Category
4. View Categories
5. Set Budget
6. View Budgets
7. Generate Report
8. Exit the Program
Enter your choice: 1
Enter amount: 15000
Enter category ID: 21
Enter date (YYYY-MM-DD): 2024-01-20
Expenses added Successfully!

Welcome to the Expense Tracker!
1. Add Expense
2. View Expenses
3. Add Category
4. View Categories
5. Set Budget
6. View Budgets
7. Generate Report
8. Exit the Program
Enter your choice: 7
```

```
Data successfully exported to ExpenseReport.csv
```



Category Name	Amount
Books	15000
Dining Out	300
Education	600
Entertainment	1650
Food	200.5
Fuel	250
Gifts	75
Gym Membership	200
Health	500
Home Maintenance	350
Insurance	400
Internet	500

```
Welcome to the Expense Tracker!
1. Add Expense
2. View Expenses
3. Add Category
4. View Categories
5. Set Budget
6. View Budgets
7. Generate Report
8. Exit the Program
Enter your choice: 8 Exiting
the Program
Good Bye!

PS C:\Users\Sagar\OneDrive\Desktop\Sagar\ExpenseTracker>
```

CONCLUSION

The **Expense Tracker** project successfully implements a simple yet effective solution for tracking and managing daily expenses, setting budgets, and generating spending reports. By leveraging **Python** for application logic and **MySQL** for database management, the project integrates a clear and intuitive interface with a reliable backend to store and retrieve data.

Throughout the project, essential features such as categorizing expenses, setting budget limits, and generating expense reports have been implemented in a straightforward manner. The use of SQL queries to interact with the database ensures efficient management of categories, expenses, and budgets, while Python handles the logic to input, retrieve, and display data.

The **Expense Tracker** project allows users to:

- Record daily expenses in different categories.
- Set budgets for various expense categories.
- View detailed reports of total spending per category.

By using **mysql.connector**, the connection between Python and MySQL is both seamless and efficient, ensuring smooth interaction with the database while maintaining simplicity. The use of camelCase for function and variable names ensures clean and readable code, following best practices for naming conventions.

Key Benefits:

- **User-Friendly Interface**: The project provides an intuitive command-line interface that allows users to easily add and manage their expenses.
- **Efficient Data Management**: By using MySQL, the project ensures organized and efficient storage of data, making it easy to query and update expense records.
- **Budget Tracking**: Users can set budgets for different categories and track their spending against those budgets, helping them manage their finances effectively.
- **Real-Time Reporting**: The system can generate reports summarizing total spending per category, giving users quick insights into their financial habits.
- **Modular Design**: The project's structure is modular, allowing easy future enhancements such as adding new features or expanding functionality.

Overall, the project serves as a solid foundation for anyone looking to manage their finances effectively. With its modular design and clear structure, it can easily be extended with additional features, such as user authentication or more complex reporting. This expense tracking system is reliable, user-friendly, and offers a scalable solution for personal financial management.

BIBLIOGRAPHY AND REFERENCES

Books:

1. **Computer Science with Python** by **Sumita Arora**
   This book is a comprehensive resource for understanding Python programming. It provides a detailed explanation of Python syntax, data structures, and application development concepts, making it a useful reference for the development of projects using Python.
2. **Class XI and XII NCERT Books**
   The NCERT textbooks for **Class XI** and **Class XII** in Computer Science provide foundational knowledge on topics related to Python programming, databases, and system design. These resources helped in understanding the concepts of file handling, database management, and programming constructs.
3. **CBSE Study Materials**
   The official CBSE study materials were used for understanding the theoretical concepts and applying them practically in the project. These materials helped in understanding the requirements and guidelines for the project, especially in terms of database integration and Python programming.

Websites:

1. **Python Official Documentation** (https://docs.python.org/3/)
   The official Python documentation provided in-depth knowledge on the use of Python libraries, including mysql.connector, and helped in solving various implementation challenges during the development of the project.
2. **MySQL Documentation** (https://dev.mysql.com/doc/)
   MySQL documentation was a key reference for understanding SQL queries and their usage in managing the database for this project. It also provided guidelines on setting up and connecting Python with MySQL.
3. **W3Schools** (https://www.w3schools.com/python/)
   W3Schools provided helpful tutorials and examples for basic Python programming, especially for database interaction and web development concepts.

Online Resources:

1. **Stack Overflow** (https://stackoverflow.com)
   Stack Overflow was used to resolve programming-related issues and find solutions to common errors encountered during the development of the project.
2. **GeeksforGeeks** (https://www.geeksforgeeks.org)
   GeeksforGeeks provided a variety of tutorials on Python programming, database management, and MySQL queries that helped in building the project.