BUSINESS EXPENSE TRACKING SYSTEM

Established – 1961 Subject: **OSDBMS**

SEVA SADAN'S

R. K. TALREJA COLLEGE

OF

ARTS, SCIENCE & COMMERCE ULHASNAGAR – 421 003

CERTIFICATE

This is to certify that Mr./Ms. Sahil Sunil Kavalkar of S.Y. Information Technology (SYIT) Roll No. 2542014 has satisfactorily completed the Open Source DataBase Management System Mini Project entitled Business Expense Tracking System during the academic year 2025 – 2026, as a part of the practical requirement. The project work is found to be satisfactory and is approved for submission.

PROF. INCHARGE **HEAD OF DEPT**

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my subject teacher Asst. Prof. Sahil Shukla sir for his valuable guidance, continuous support, and helpful suggestions throughout the completion of this project. His explanations and feedback helped me understand the practical concepts of MySQL and database management in a clear and easy manner.

I am thankful to R.K. Talreja College for providing the opportunity and necessary resources to complete this project as a part of the Open Source Database Management System subject. The laboratory facilities and learning environment helped me work on this project effectively.

I would also like to thank my classmates and friends who supported me whenever I faced difficulties during the project. Their discussions and cooperation helped me solve problems and improve my understanding.

Finally, I would like to express my heartfelt thanks to my parents and family members for their encouragement, patience, and motivation during the preparation of this project. Their support helped me stay focused and complete the work successfully.

This project gave me practical experience in database design and SQL queries, and I am grateful to everyone who directly or indirectly contributed to its completion.

INDEX

1. INTRODUCTION

A database system is a structured way of storing and managing data in an organized format. Instead of keeping records in notebooks or random files, a database stores information in tables so that it can be easily accessed, updated, and managed. A Database Management System (DBMS) helps users to create, store, retrieve, and control data efficiently. It reduces confusion and makes data handling more systematic.

In this project, I have created a Business Expense Tracking Database. This system is designed to store and manage business expense records in a proper and organized way. In many small businesses, expenses are recorded manually in registers or spreadsheets. This method can lead to mistakes, duplication of data, and difficulty in finding old records. Sometimes calculations are wrong, or important data gets lost. Updating records also becomes time-consuming. This project solves these problems by storing all expense-related information in a centralized database system.

The main problem area addressed in this project is improper expense management. When expenses such as travel costs, office supplies, electricity bills, or other payments are not recorded properly, it becomes difficult to track total spending. It also becomes hard to generate monthly or category-wise reports. By using a database system, all expenses are stored in related tables like User, Category, Expense, and Payment Method. This makes searching, updating, and generating reports much faster and more accurate.

For this project, I have used MySQL as the database system. MySQL is an open-source database management system, which means it is free to use and easily available. It is reliable, secure, and widely used in real-world applications. MySQL supports Structured Query Language (SQL), which helps in performing operations like creating tables, inserting data, updating records, and retrieving information. Since this subject is Open Source Database Management System (OSDBMS), MySQL is a suitable choice because it is open-source and practical for learning database concepts.

Overall, this project helped me understand how a database system works in real-life situations. It shows how MySQL can be used to manage business expenses efficiently, reduce errors, and maintain proper financial records in a structured way.

2. PROBLEM DEFINITION

In many small businesses, expenses are recorded manually in notebooks, registers, or basic spreadsheets. This type of system is unstructured and not properly organized. When data is stored manually, it becomes difficult to maintain accuracy and consistency. As the number of records increases, managing and tracking expenses becomes more complicated. There is also a high chance of human errors.

The existing manual system creates several problems such as:

- **Data Redundancy:** The same expense information may be written multiple times in different places, which wastes time and storage.
- **Data Inconsistency:** If data is updated in one place but not in another, it leads to mismatched records and incorrect reports.
- **Calculation Errors:** Manual calculations of total expenses can lead to mistakes.
- **Slow Searching:** Finding old expense records takes a lot of time.
- **Lack of Security:** Anyone can access or modify manual records, and there is no proper protection.
- **No Proper Backup:** Records can be lost due to damage, deletion, or system failure.

Because of these issues, there is a need for a proper database-driven solution. A database system like MySQL helps in storing data in structured tables with relationships. It reduces redundancy, maintains consistency using constraints, improves security through user access control, and allows quick retrieval of information using SQL queries. Therefore, a database system is necessary to manage business expenses efficiently and accurately.

---

3. OBJECTIVES OF THE PROJECT

The main objective of this project is to design and develop a Business Expenses Tracking Database using MySQL. In today's time, managing business expenses properly is very important. If expenses are not recorded in a structured way, it becomes difficult to track spending and control financial activities. Through this project, I want to create a simple and organized database system that can store and manage expense records efficiently.

One of the primary objectives of this project is to design a structured database system. This includes creating different tables such as User, Category, Expense, and Payment Method. Each table stores specific information, and all tables are connected using proper relationships. By designing a structured database, the data becomes organized and easy to manage. It also helps in reducing confusion and maintaining proper records.

Another important objective is to implement SQL queries for data management. SQL (Structured Query Language) is used to perform different operations on the database. In this project, I use SQL commands such as CREATE to create tables, INSERT to add new expense records, UPDATE to modify existing data, DELETE to remove unwanted records, and SELECT to retrieve information. These queries help in managing the data effectively and generating useful reports such as total expenses or category-wise expenses.

Ensuring data integrity is also one of the key objectives of this project. Data integrity means maintaining accuracy and consistency of data. To achieve this, I have used constraints such as PRIMARY KEY, FOREIGN KEY, NOT NULL, and UNIQUE. These constraints prevent duplicate entries, maintain relationships between tables, and ensure that important fields are not left empty. This makes the database more reliable and secure.

Another objective of this project is to gain hands-on experience with MySQL as an open-source database management system. Since this subject focuses on Open Source DBMS, using MySQL helps me understand how real-world databases are created and managed. By working on this project, I have learned how to design tables, create relationships, write queries, and maintain data properly.

4. SCOPE OF THE PROJECT

The Business Expense Tracking Database is designed to manage and store business expense records in an organized way. The system mainly focuses on recording daily expenses, categorizing them properly, and generating useful reports. It can be used wherever expense management is required in a simple and structured format.

This system can be used in small businesses, startups, shops, and even for personal expense tracking. Many small business owners do not use complex accounting software and prefer simple systems. This database provides a basic and effective solution for managing financial records without much technical complexity.

The main users of this system are:

- Admin**:** The admin has full control over the database. The admin can add, update, delete, and view expense records. The admin can also manage categories and payment methods.
- Staff or Employee: Staff members can enter daily expense details and view records related to their work.
- Business Owner: The business owner can check total expenses, category-wise expenses, and monthly reports to understand spending patterns.
- Student (Academic Use): Students can use this system for learning database concepts and understanding how real-world expense systems work.

The scope of this project can be explained in the following areas:

1. Educational Use

This project is very useful for academic learning. It helps students understand database design, table relationships, SQL queries, and constraints. It gives practical knowledge of how a real-life database system works. It is mainly developed as part of an academic requirement for OSDBMS subject.

2. Business Use

The system can be used in small-scale businesses to track daily expenses such as travel, office supplies, electricity bills, and other operational costs. It helps in maintaining proper

records and generating simple reports. However, it is not designed for large companies with complex accounting requirements.

3. Administrative Use

The system can also be used by administrative departments in small organizations to monitor expenses. It helps in reducing manual work and improving record management.

Since this project is developed for academic purposes, it has certain limitations. It does not include advanced features like online payment integration, GST calculation, graphical dashboards, or multi-branch management. It is limited to basic expense tracking and database operations using MySQL.

Overall, the scope of this project is limited to small-scale and academic use, but it clearly demonstrates how a database system can solve real-world problems related to expense management.

5. REQUIREMENT SPECIFICATION

The requirement specification describes the basic hardware and software needed to develop and run the Business Expense Tracking Database system. Since this project is developed using MySQL and SQL queries, it does not require high-end hardware. It can run on a normal computer or laptop.

5.1 Hardware Requirements

The hardware requirements for this project are minimal and easily available. The system can be developed and executed on a standard computer setup.

- **Computer / Laptop:**
  A basic computer or laptop is required to install MySQL and MySQL Workbench. Any modern system that supports Windows or Linux operating system can be used.
- **Minimum RAM:**
  At least 4 GB RAM is recommended for smooth working of MySQL and related tools. However, 8 GB RAM is better for faster performance.
- **Storage:**
  Minimum 20 GB free storage space is sufficient to install the required software and store the database files. Since this project handles small to medium-sized data, it does not require large storage capacity.

Software	Purpose
MySQL Server	Database management
MySQL Workbench	Query execution and design
OS (Windows/Linux)	Platform for system
SQL	Query language

6. SYSTEM DESIGN

System Architecture

The system architecture of the Business Expenses Tracking Database explains how the user interacts with the database and how data is processed inside the system. This project follows a simple database architecture where the user communicates with the MySQL Server through SQL queries.

User Interaction with the Database

In this system, the user interacts with the database using MySQL Workbench. The user writes SQL queries such as INSERT, UPDATE, DELETE, and SELECT to perform different operations. For example, when a user wants to add a new expense, they write an INSERT query. When they want to view total expenses, they use a SELECT query.

The user does not directly access the stored data files. Instead, all communication happens through SQL commands. This makes the system structured and secure.

Role of MySQL Server

MySQL Server is the main component of the system. It manages the entire database. Its main roles are:

- Storing data in tables
- Managing relationships between tables
- Enforcing constraints like PRIMARY KEY and FOREIGN KEY
- Controlling user access and security
- Processing SQL queries

When a query is submitted, MySQL Server checks the syntax, verifies permissions, processes the request, and then returns the result to the user.

Query Execution Flow

The flow of query execution in this system works in the following steps:

1. The user writes an SQL query in MySQL Workbench.
2. The query is sent to the MySQL Server.
3. The MySQL Server parses and validates the query.
4. If the query is correct, the server executes it.
5. The result (such as inserted data or retrieved records) is returned to the user.

For example, when a SELECT query is executed to view expenses, MySQL Server searches the required table, fetches the matching records, and displays the result.

This conceptual system design shows how the database works internally without exposing direct data files to users. It ensures proper data management, security, and efficient query processing.

7. DATABASE DESIGN

7.1 Entity Description

In this project, the database is designed using different tables to store business expense information in a structured way. Each table has a specific purpose, and all tables are connected using relationships.

1. User Table

The User table stores information about the person who is using the system. It includes details such as user name, email, and password. Each user has a unique ID, which helps in identifying records related to that user. This table helps in managing multiple users in the system.

2. Category Table

The Category table stores different types of expense categories such as Travel, Office Supplies, Electricity, and Miscellaneous. Each category has a unique category ID. This table helps in organizing expenses into proper groups, which makes reporting easier.

3. Expense Table

The Expense table is the main table of the system. It stores all expense records. It includes details such as expense amount, date, description, and references to user and category. Each expense is linked to a specific user and category using foreign keys. This table helps in tracking daily business expenses.

4. Payment_Method Table

The Payment_Method table stores information about how the payment was made, such as Cash, Credit Card, Debit Card, or Online Transfer. This helps in tracking payment modes used for expenses.

All these tables are connected logically, which reduces data duplication and maintains proper relationships between data.

7.2 Table Structure

Below are the table structures with attributes and data types used in the project:

1. User Table

Attribute	Data Type
user_id	INT (Primary Key)
user_name	VARCHAR(50)
email	VARCHAR(100)
password	VARCHAR(100)
role	VARCHAR(20)

2. Category Table

Attribute	Data Type
category_id	INT (Primary Key)
category_name	VARCHAR(50)

3. Payment_Method Table

Attribute	Data Type
payment_id	INT (Primary Key)
payment_type	VARCHAR(50)

4. Expense Table

Attribute	Data Type
expense_id	INT (Primary Key)
user_id	INT (Foreign Key)
category_id	INT (Foreign Key)
payment_id	INT (Foreign Key)
amount	DECIMAL(10,2)
expense_date	DATE
description	VARCHAR(255)

The Expense table connects with User, Category, and Payment_Method tables using foreign keys.

7.3 Constraints Used

Constraints are rules applied to table columns to maintain data accuracy and integrity.

PRIMARY KEY:
A Primary Key uniquely identifies each record in a table. For example, user_id in the User table and expense_id in the Expense table are primary keys. It does not allow duplicate or NULL values.

FOREIGN KEY:
A Foreign Key creates a relationship between two tables. For example, user_id in the Expense table is a foreign key that refers to the User table. This ensures that an expense cannot exist without a valid user.

NOT NULL:

The NOT NULL constraint ensures that a column cannot have an empty value. Important fields like amount and expense_date must always have data.

UNIQUE:

The UNIQUE constraint ensures that all values in a column are different. For example, email in the User table can be set as UNIQUE so that no two users can register with the same email address.

Using these constraints helps in maintaining proper relationships between tables and ensures that the database remains accurate and consistent.

---

8. UML DIAGRAMS

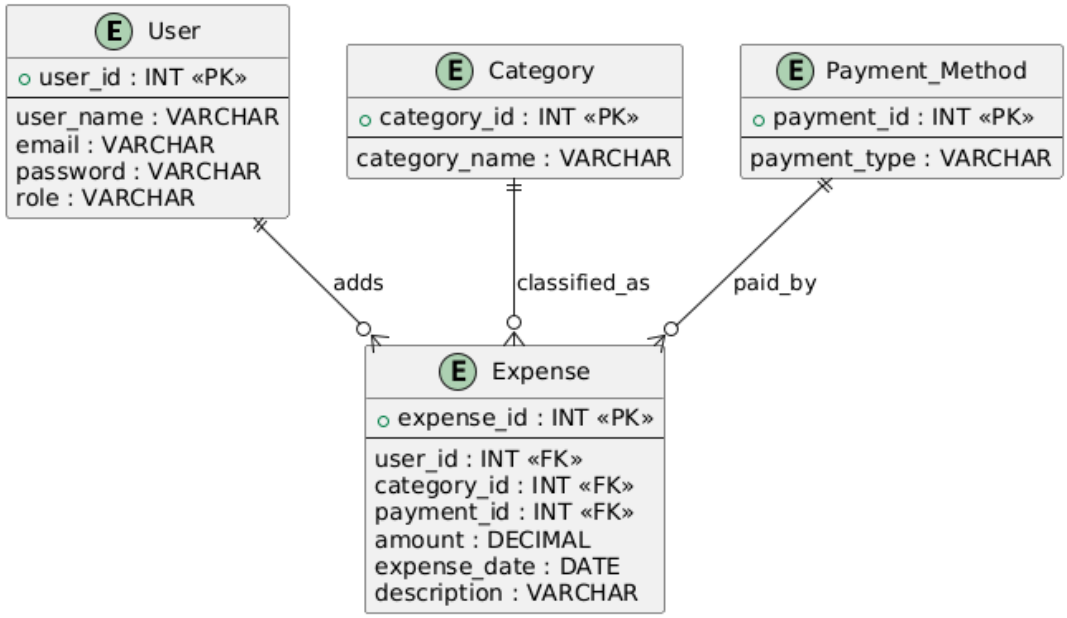

Figure 8.1 ER Diagram

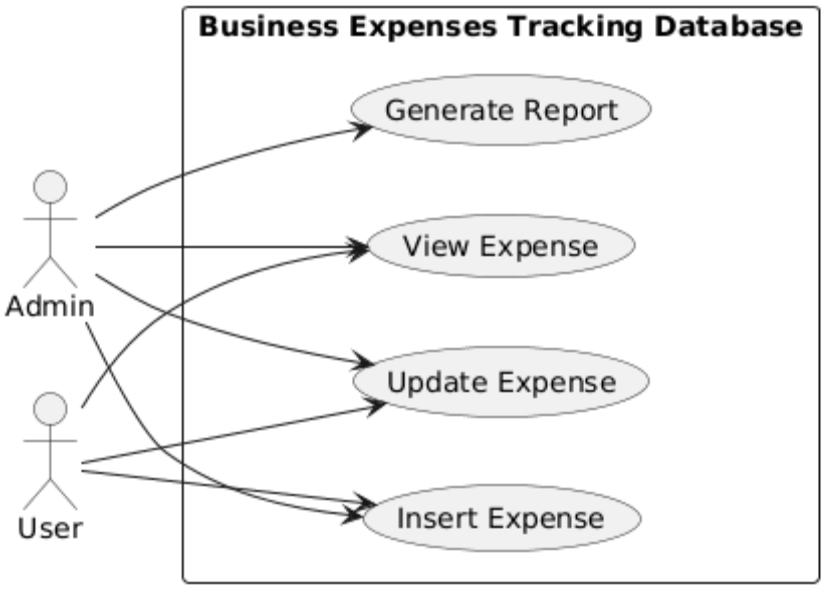

Figure 8.2 Use Case Diagram

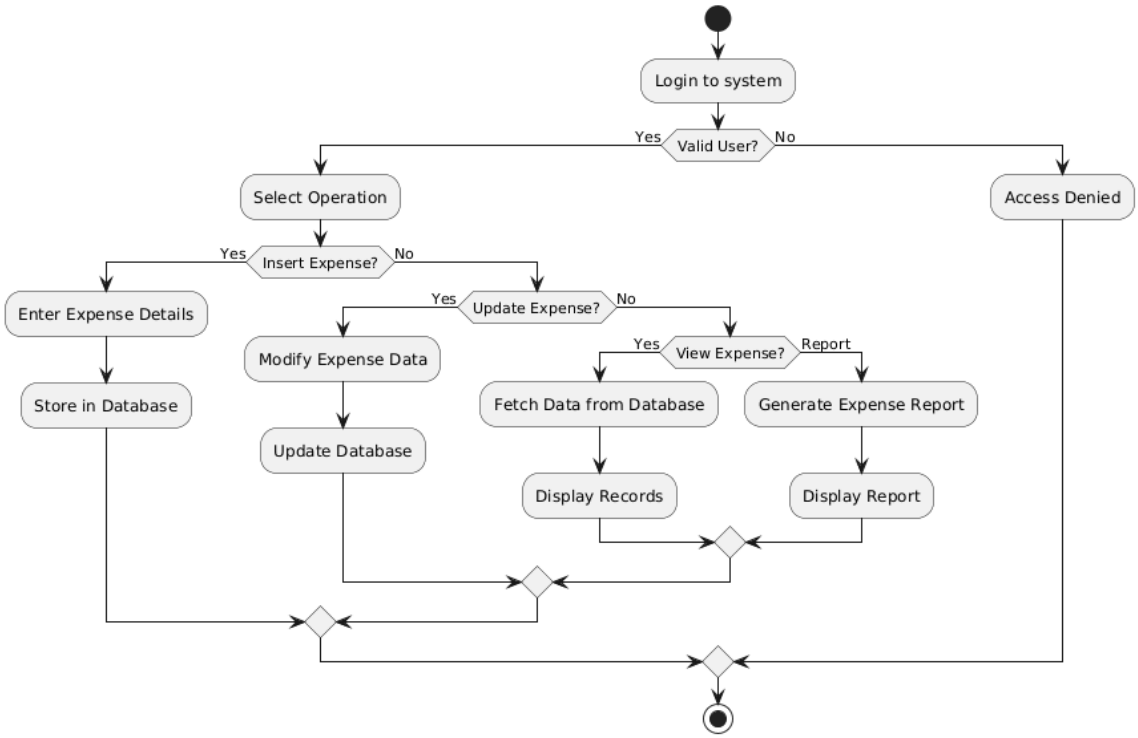

Figure 8.3 Activity Diagram

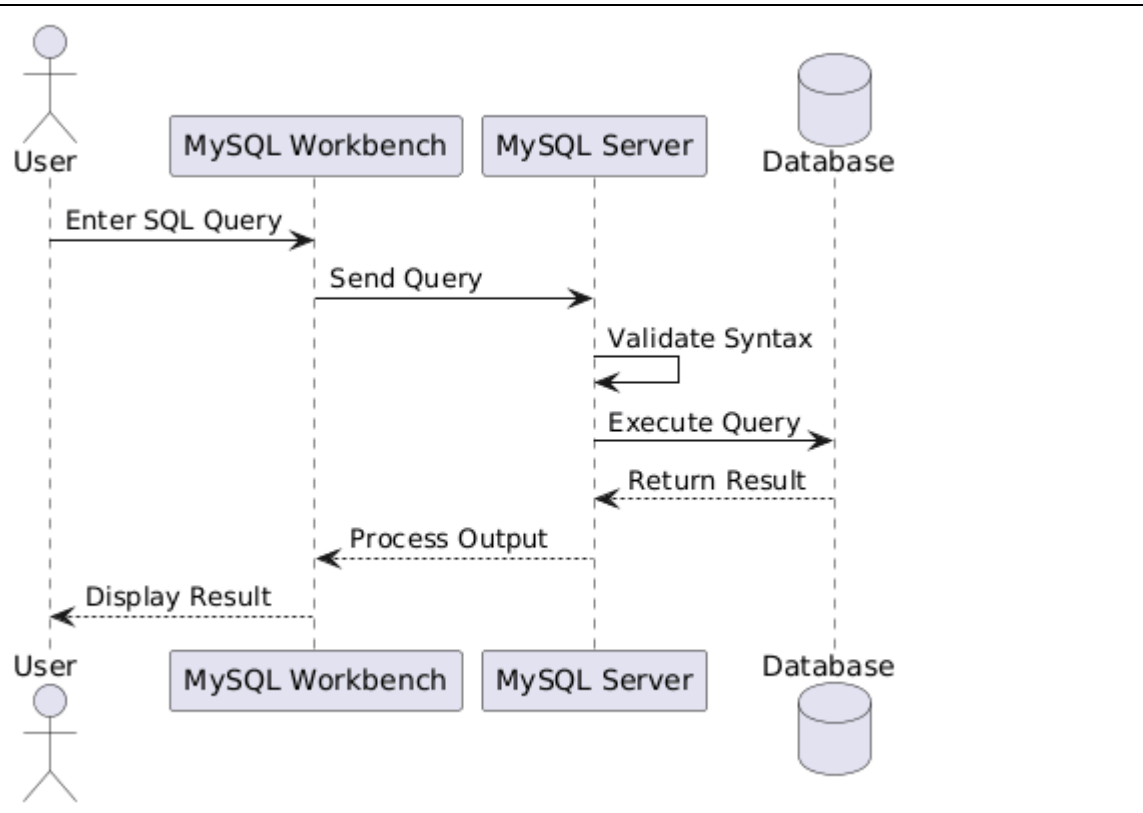

Figure 8.4 Sequence Diagram

9. SQL IMPLEMENTATION

9.1 Database Creation

```
CREATE DATABASE business_expenses_db;
USE business_expenses_db;
```

9.2 Table Creation

1. User Table

```
CREATE TABLE User (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    user_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(100) NOT NULL,
    role VARCHAR(20) NOT NULL
);
```

2. Category Table

```
CREATE TABLE Category (
    category_id INT AUTO_INCREMENT PRIMARY KEY,
    category_name VARCHAR(50) NOT NULL
);
```

3. Payment_Method Table

```
CREATE TABLE Payment_Method (
    payment_id INT AUTO_INCREMENT PRIMARY KEY,
    payment_type VARCHAR(50) NOT NULL
);
```

4. Expense Table

```
CREATE TABLE Expense (
```

```
    expense_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    category_id INT,
    payment_id INT,
    amount DECIMAL(10,2) NOT NULL,
    expense_date DATE NOT NULL,
    description VARCHAR(255),
    FOREIGN KEY (user_id) REFERENCES User(user_id),
    FOREIGN KEY (category_id) REFERENCES Category(category_id),
    FOREIGN KEY (payment_id) REFERENCES
    Payment_Method(payment_id)
);
```

9.3 Data Insertion

Insertion operations can also be executed inside a transaction to ensure safe data entry.

Insert into User

```
INSERT INTO User (user_name, email, password, role) VALUES
('Rahul Sharma', 'rahul@gmail.com', 'rahul123', 'Admin'),
('Deepak Patel', 'deepak@gmail.com', 'deepak123', 'User'),
('Amit Joshi', 'amit@gmail.com', 'amit123', 'User'),
('Keshav Jha, 'keshav@gmail.com', 'keshav123', 'User');
```

Insert into Category

```
INSERT INTO Category (category_name) VALUES
('Travel'),
('Food'),
('Office Supplies'),
('Utilities'),
('Maintenance');
```

Insert into Payment_Method

```
INSERT INTO Payment_Method (payment_type) VALUES
```

```
('Cash'),
('UPI'),
('Credit Card'),
('Debit Card'),
('Net Banking');
```

Insert into Expense

```
INSERT INTO Expense (user_id, category_id, payment_id, amount,
expense_date, description) VALUES
(1, 2, 2, 250.00, '2025-01-05', 'Lunch with client'),
(2, 1, 3, 1200.00, '2025-01-08', 'Taxi fare'),
(3, 3, 4, 850.00, '2025-01-10', 'Printer ink purchase'),
(4, 4, 5, 2300.00, '2025-01-12', 'Electricity bill'),
(1, 5, 1, 600.00, '2025-01-15', 'AC servicing'),
(2, 2, 2, 180.00, '2025-01-18', 'Snacks meeting'),
(3, 1, 3, 950.00, '2025-01-20', 'Fuel expense'),
(4, 3, 4, 400.00, '2025-01-22', 'Stationery purchase'),
(1, 4, 5, 2100.00, '2025-01-25', 'Internet bill'),
(2, 2, 2, 320.00, '2025-01-28', 'Dinner with vendor');
```

9.4 Data Retrieval

Simple SELECT

```
SELECT * FROM Expense;
```

SELECT with WHERE

```
SELECT * FROM Expense
WHERE amount > 1000;
```

JOIN Query

To display expense details with category and payment type:

```
SELECT e.expense_id, u.user_name, c.category_name,
       p.payment_type, e.amount, e.expense_date
FROM Expense e
JOIN User u ON e.user_id = u.user_id
JOIN Category c ON e.category_id = c.category_id
JOIN Payment_Method p ON e.payment_id = p.payment_id;
```

9.5 Advanced Queries

GROUP BY

Total expenses category-wise:

```
SELECT c.category_name, SUM(e.amount) AS total_expense
FROM Expense e
JOIN Category c ON e.category_id = c.category_id
GROUP BY c.category_name;
```

HAVING

Show categories where total expense is more than 2000:

```
SELECT c.category_name, SUM(e.amount) AS total_expense
FROM Expense e
JOIN Category c ON e.category_id = c.category_id
GROUP BY c.category_name
HAVING SUM(e.amount) > 2000;
```

Subquery

Find expenses greater than average expense:

```
SELECT * FROM Expense
WHERE amount > (SELECT AVG(amount) FROM Expense);
```

View Creation

Create a view for expense report:

```
CREATE VIEW expense_report AS
SELECT e.expense_id, u.user_name, c.category_name,
       p.payment_type, e.amount, e.expense_date
FROM Expense e
JOIN User u ON e.user_id = u.user_id
JOIN Category c ON e.category_id = c.category_id
JOIN Payment_Method p ON e.payment_id = p.payment_id;
```

To see the view:

```
SELECT * FROM expense_report;
```

9.6 Transaction Handling

A transaction is a group of SQL statements executed as a single unit. It ensures that either all operations are completed successfully or none of them are applied. This helps in maintaining data consistency in the database.

In the Business Expenses Tracking Database, a transaction is useful when inserting an expense and updating related information together. If an error occurs during execution, the database can be restored to its previous state using ROLLBACK.

Before executing a transaction, autocommit mode is disabled.

```
SET autocommit = 0;
START TRANSACTION;
INSERT INTO Expense (user_id, category_id, payment_id, amount,
expense_date, description)
VALUES (1, 2, 1, 500.00, '2026-02-15', 'Stationery purchase');
UPDATE Category
SET category_name = 'Office Materials'
WHERE category_id = 2;
COMMIT;
```

If any error occurs:

```
ROLLBACK;
```

After execution, autocommit mode can be enabled again.

```
SET autocommit = 1;
```

10. SYSTEM TESTING AND RESULT

10.1 Query Correctness Testing

All SQL queries such as CREATE, INSERT, UPDATE, DELETE, and SELECT were tested one by one.

- The database was successfully created using CREATE DATABASE.
- All tables were created without syntax errors.
- Foreign key relationships were properly established.
- Data was inserted into each table using INSERT queries.
- SELECT queries returned correct records.
- JOIN queries successfully combined data from multiple tables.
- GROUP BY and HAVING queries generated proper summarized reports.

Each query was tested multiple times to ensure it works correctly.

10.2 Data Validation

The following validations were tested:

- PRIMARY KEY prevented duplicate IDs.
- UNIQUE constraint prevented duplicate email entries.
- NOT NULL constraint prevented empty values in important fields.
- FOREIGN KEY constraint prevented inserting expense records without valid user, category, or payment method.

For example, when trying to insert an expense with a non-existing user_id, MySQL showed an error. This confirms that referential integrity is maintained.

10.3 Result Validation

After executing all SQL queries in MySQL CLI, the results were checked to ensure the database works correctly.

Each operation such as inserting records, updating data and retrieving reports produced expected outputs without errors. The data stored in tables was displayed correctly and relationships between tables were properly maintained through foreign keys.

Join queries successfully combined data from User, Category, Payment_Method and Expense tables, showing meaningful reports like expense details along with user name, category and payment type. Aggregate queries like total expense calculation also returned correct values.

Screenshots of outputs are attached below as proof of successful execution. These outputs confirm that the database structure, constraints and queries are functioning as intended.

---

1. SELECT and WHERE

```
mysql> SELECT * FROM Expense;
+------------+---------+-------------+------------+---------+--------------+----------------------+
| expense_id | user_id | category_id | payment_id | amount  | expense_date | description          |
+------------+---------+-------------+------------+---------+--------------+----------------------+
|          1 |       1 |           2 |          2 |  250.00 | 2025-01-05   | Lunch with client    |
|          2 |       2 |           1 |          3 | 1200.00 | 2025-01-08   | Taxi fare            |
|          3 |       3 |           3 |          4 |  850.00 | 2025-01-10   | Printer ink purchase |
|          4 |       4 |           4 |          5 | 2300.00 | 2025-01-12   | Electricity bill     |
|          5 |       1 |           5 |          1 |  600.00 | 2025-01-15   | AC servicing         |
|          6 |       2 |           2 |          2 |  180.00 | 2025-01-18   | Snacks meeting       |
|          7 |       3 |           1 |          3 |  950.00 | 2025-01-20   | Fuel expense         |
|          8 |       4 |           3 |          4 |  400.00 | 2025-01-22   | Stationery purchase  |
|          9 |       1 |           4 |          5 | 2100.00 | 2025-01-25   | Internet bill        |
|         10 |       2 |           2 |          2 |  320.00 | 2025-01-28   | Dinner with vendor   |
+------------+---------+-------------+------------+---------+--------------+----------------------+
10 rows in set (0.00 sec)

mysql>
mysql>
mysql> SELECT * FROM Expense
    -> WHERE amount > 1000;
+------------+---------+-------------+------------+---------+--------------+------------------+
| expense_id | user_id | category_id | payment_id | amount  | expense_date | description      |
+------------+---------+-------------+------------+---------+--------------+------------------+
|          2 |       2 |           1 |          3 | 1200.00 | 2025-01-08   | Taxi fare        |
|          4 |       4 |           4 |          5 | 2300.00 | 2025-01-12   | Electricity bill |
|          9 |       1 |           4 |          5 | 2100.00 | 2025-01-25   | Internet bill    |
+------------+---------+-------------+------------+---------+--------------+------------------+
```

2. JOIN Query

```
mysql> SELECT e.expense_id, u.user_name, c.category_name,
    ->        p.payment_type, e.amount, e.expense_date
    -> FROM Expense e
    -> JOIN User u ON e.user_id = u.user_id
    -> JOIN Category c ON e.category_id = c.category_id
    -> JOIN Payment_Method p ON e.payment_id = p.payment_id;
+------------+-------------+-----------------+--------------+---------+--------------+
| expense_id | user_name   | category_name   | payment_type | amount  | expense_date |
+------------+-------------+-----------------+--------------+---------+--------------+
|          1 | Rahul Sharma | Food            | UPI          |  250.00 | 2025-01-05   |
|          5 | Rahul Sharma | Maintenance     | Cash         |  600.00 | 2025-01-15   |
|          9 | Rahul Sharma | Utilities       | Net Banking  | 2100.00 | 2025-01-25   |
|          2 | Deepak Patel | Travel          | Credit Card  | 1200.00 | 2025-01-08   |
|          6 | Deepak Patel | Food            | UPI          |  180.00 | 2025-01-18   |
|         10 | Deepak Patel | Food            | UPI          |  320.00 | 2025-01-28   |
|          3 | Amit Joshi   | Office Supplies | Debit Card   |  850.00 | 2025-01-10   |
|          7 | Amit Joshi   | Travel          | Credit Card  |  950.00 | 2025-01-20   |
|          4 | Keshav Jha   | Utilities       | Net Banking  | 2300.00 | 2025-01-12   |
|          8 | Keshav Jha   | Office Supplies | Debit Card   |  400.00 | 2025-01-22   |
+------------+-------------+-----------------+--------------+---------+--------------+
```

3. GROUP BY

```
mysql> SELECT c.category_name, SUM(e.amount) AS total_expense
    -> FROM Expense e
    -> JOIN Category c ON e.category_id = c.category_id
    -> GROUP BY c.category_name;
+-----------------+---------------+
| category_name   | total_expense |
+-----------------+---------------+
| Travel          |       2150.00 |
| Food            |        750.00 |
| Office Supplies |       1250.00 |
| Utilities       |       4400.00 |
| Maintenance     |        600.00 |
+-----------------+---------------+
```

4. HAVING

```
mysql> SELECT c.category_name, SUM(e.amount) AS total_expense
    -> FROM Expense e
    -> JOIN Category c ON e.category_id = c.category_id
    -> GROUP BY c.category_name
    -> HAVING SUM(e.amount) > 2000;
+---------------+---------------+
| category_name | total_expense |
+---------------+---------------+
| Travel        |       2150.00 |
| Utilities     |       4400.00 |
+---------------+---------------+
```

5. Subquery

```
mysql> SELECT * FROM Expense
    -> WHERE amount > (SELECT AVG(amount) FROM Expense);
+------------+---------+-------------+------------+---------+--------------+-----------------+
| expense_id | user_id | category_id | payment_id | amount  | expense_date | description     |
+------------+---------+-------------+------------+---------+--------------+-----------------+
|          2 |       2 |           1 |          3 | 1200.00 | 2025-01-08   | Taxi fare       |
|          4 |       4 |           4 |          5 | 2300.00 | 2025-01-12   | Electricity bill |
|          7 |       3 |           1 |          3 |  950.00 | 2025-01-20   | Fuel expense    |
|          9 |       1 |           4 |          5 | 2100.00 | 2025-01-25   | Internet bill   |
+------------+---------+-------------+------------+---------+--------------+-----------------+
```

6. VIEW Creation

```
mysql> CREATE VIEW expense_report AS
    -> SELECT e.expense_id, u.user_name, c.category_name,
    ->        p.payment_type, e.amount, e.expense_date
    -> FROM Expense e
    -> JOIN User u ON e.user_id = u.user_id
    -> JOIN Category c ON e.category_id = c.category_id
    -> JOIN Payment_Method p ON e.payment_id = p.payment_id;
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql>
mysql> SELECT * FROM expense_report;
+------------+--------------+-----------------+--------------+---------+--------------+
| expense_id | user_name    | category_name   | payment_type | amount  | expense_date |
+------------+--------------+-----------------+--------------+---------+--------------+
|          1 | Rahul Sharma | Food            | UPI          |  250.00 | 2025-01-05   |
|          2 | Deepak Patel | Travel          | Credit Card  | 1200.00 | 2025-01-08   |
|          3 | Amit Joshi   | Office Supplies | Debit Card   |  850.00 | 2025-01-10   |
|          4 | Keshav Jha   | Utilities       | Net Banking  | 2300.00 | 2025-01-12   |
|          5 | Rahul Sharma | Maintenance     | Cash         |  600.00 | 2025-01-15   |
|          6 | Deepak Patel | Food            | UPI          |  180.00 | 2025-01-18   |
|          7 | Amit Joshi   | Travel          | Credit Card  |  950.00 | 2025-01-20   |
|          8 | Keshav Jha   | Office Supplies | Debit Card   |  400.00 | 2025-01-22   |
|          9 | Rahul Sharma | Utilities       | Net Banking  | 2100.00 | 2025-01-25   |
|         10 | Deepak Patel | Food            | UPI          |  320.00 | 2025-01-28   |
+------------+--------------+-----------------+--------------+---------+--------------+
```

7. TRANSACTION BLOCK

```
mysql> SET autocommit = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO Expense (user_id, category_id, payment_id, amount, expense_date, description)
    -> VALUES (1, 2, 1, 500.00, '2026-02-15', 'Stationery purchase');
Query OK, 1 row affected (0.00 sec)

mysql> UPDATE Category
    -> SET category_name = 'Office Materials'
    -> WHERE category_id = 2;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

8. ROLLBACK (if error occurs)

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql> SET autocommit = 1;
Query OK, 0 rows affected (0.00 sec)
```

11. SECURITY, BACKUP AND RECOVERY

Security, backup, and recovery are important parts of any database system. In this project, proper measures are taken to ensure that the Business Expenses Tracking Database remains secure and protected from data loss.

11.1 Security

Database security ensures that only authorized users can access and modify data.

User Privileges

MySQL provides user account management and privilege control. Different users can be given different permissions based on their role.

For example:

- Admin can create, insert, update, and delete records.
- Normal users can only insert and view expenses.
- Unauthorized users cannot access the database.

Privileges like SELECT, INSERT, UPDATE, and DELETE can be granted using MySQL commands.

Example:

```
GRANT SELECT, INSERT ON business_expenses_db.* TO
'user1'@'localhost';
```

This ensures that the user can only view and insert data but cannot delete or modify tables.

Access Control

Access control is maintained using:

- Username and password authentication
- Role-based access (Admin/User)

- Limited privileges using GRANT and REVOKE commands

This prevents unauthorized access and protects sensitive financial information stored in the database.

11.2 Backup

Backup is necessary to prevent data loss due to system failure, accidental deletion, or corruption.

In MySQL, database backup can be taken using the **mysqldump** utility.

mysqldump is a command-line tool that creates a copy of the entire database in the form of a SQL file.

Example command:

```
mysqldump -u root -p business_expenses_db > backup.sql
```

This command creates a backup file named backup.sql containing all tables and data.

Backup ensures that data can be restored if any problem occurs.

11.3 Recovery

Recovery is the process of restoring the database from a backup file in case of data loss.

If the database is deleted or corrupted, it can be restored using the backup file created by mysqldump.

Example restore command:

```
mysql -u root -p business_expenses_db < backup.sql
```

This command recreates the database structure and inserts all saved data from the backup file.

Recovery ensures business continuity and prevents permanent data loss.

Overall, security, backup, and recovery mechanisms make the database system reliable and safe. These features ensure that business expense data remains protected, consistent, and recoverable in case of failure.

12. FUTURE SCOPE AND CONCLUSION

12.1 Future Scope

The current project is a basic Business Expenses Tracking Database developed using MySQL. Although it successfully manages expense records in a structured way, there is scope for further improvement and expansion in the future.

One possible future enhancement is web integration. The system can be connected to a web application using technologies like HTML, CSS, and PHP or any backend language. This will allow users to access the system through a browser instead of directly using MySQL Workbench. It will make the system more user-friendly and practical for real business use.

Another improvement can be advanced reporting features. Currently, the system generates reports using SQL queries such as GROUP BY and aggregate functions. In the future, graphical reports like bar charts and pie charts can be added to better understand expense patterns. Monthly and yearly summary reports can also be generated automatically.

The system can also include analytics features. For example, it can analyze spending patterns and identify which category has the highest expense. It can also compare expenses month-wise and provide insights to help businesses control costs. These features will make the system smarter and more useful for decision-making.

Overall, there are many possibilities to enhance the system and make it more powerful and user-friendly.

12.2 Conclusion

The Business Expenses Tracking Database was successfully designed and implemented using MySQL. The project achieved its main goal of creating a structured and organized system for managing business expenses. Tables were created with proper relationships and constraints. SQL queries were implemented to insert, update, delete, and retrieve data efficiently.

Through this project, I gained practical knowledge of database concepts such as primary key, foreign key, joins, constraints, and transactions. I also learned how to create and manage a database using MySQL and how to execute different types of SQL queries.

This project helped me understand the importance of database systems in real-life applications. Manual record keeping can lead to errors and confusion, but a database system ensures accuracy, consistency, and security. MySQL, being an open-source database management system, provides a reliable and cost-effective solution for managing data.

Overall, this project improved my understanding of database management and gave me hands-on experience in working with MySQL. It helped me connect theoretical concepts with practical implementation.

13. REFERENCES

1. MySQL Official Documentation
   Oracle Corporation. *MySQL 8.0 Reference Manual*.
   Available at: https://dev.mysql.com/doc/
2. Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems* (7th Edition). Pearson Education.
3. Murach, J. (2019). *Murach's MySQL* (3rd Edition). Mike Murach & Associates.
4. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database System Concepts* (7th Edition). McGraw-Hill Education.
5. W3Schools. *MySQL Tutorial*.
   Available at: https://www.w3schools.com/mysql/

14. GLOSSARY

DBMS (Database Management System):
A software system that is used to create, manage, and maintain databases. It allows users to store, retrieve, and organize data efficiently.

SQL (Structured Query Language):
A programming language used to communicate with a database. It is used to create tables, insert data, update records, delete data, and retrieve information.

Primary Key:
A column in a table that uniquely identifies each record. It does not allow duplicate or NULL values. For example, expense_id in the Expense table.

Foreign Key:
A column that creates a relationship between two tables. It links one table to another and ensures data consistency. For example, user_id in the Expense table refers to the User table.

MySQL:
An open-source relational database management system used to store and manage data. It uses SQL for database operations and is widely used in real-world applications.

Constraint:
A rule applied to a table column to maintain data accuracy and integrity. Examples include PRIMARY KEY, FOREIGN KEY, NOT NULL, and UNIQUE.

Transaction:

A group of SQL operations executed as a single unit. It ensures that either all operations are completed successfully (COMMIT) or none are applied (ROLLBACK).

Normalization:

The process of organizing data in a database to reduce redundancy and improve data integrity.