Mini project report on

Disaster Data Management System

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering

UE23CS351A – DBMS Project

Submitted by:

Shashwat Solanki	**PES2UG23CS549**
Shivam Anand	**PES2UG23CS549**

under the guidance of

Prof. Shilpa S

Assistant Professor

PES University

AUG - DEC 2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FACULTY OF ENGINEERING

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

CERTIFICATE

This is to certify that the mini project entitled

Disasters data Management System

is a bonafide work carried out by

Shashwat Solanki	**PES2UG23CS545**
Shivam Anand	**PES2UG23CS549**

In partial fulfilment for the completion of fifth semester DBMS Project (UE23CSS351A) in the Program of Study -Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period AUG. 2025 – DEC. 2025. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The project has been approved as it satisfies the 5th semester academic requirements in respect of project work.

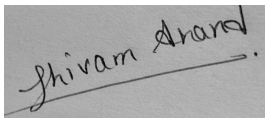
Signature

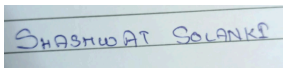
Prof. Shilpa S

Assistant Professor

DECLARATION

We hereby declare that the DBMS Project entitled **Disasters Data Management System** has been carried out by us under the guidance of **Prof. Shilpa s, Assistant Professor** and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester AUG – DEC 2025.

SHIVAM ANAND **PES2UG23CS549** 

Shashwat Solanki **PES2UG23CS545** 

ACKNOWLEDGEMENT

I would like to express my gratitude to Prof. Shilpa S, Department of Computer Science and Engineering, PES University, for her continuous guidance, assistance, and encouragement throughout the development of this UE23CS351A - DBMS Project.

I take this opportunity to thank Dr. Sandesh B J, C, Professor,ChairPerson, Department of Computer Science and Engineering, PES University, for all the knowledge and support I have received from the department.

I am deeply grateful to Prof. Jawahar Doreswamy, Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, PES University for providing to me various opportunities and enlightenment every step of the way. Finally, this DBMS Project could not have been completed without the continual support and encouragement I have received from my family and friends.

ABSTRACT

The **Lost and Found Management System (LFMS)** is a database-driven desktop application developed using **Python (Tkinter GUI)** and **MySQL** to efficiently manage lost and found items within an organization or campus. The system provides an intuitive interface for users to report, track, and claim lost or found belongings, thereby eliminating manual processes like notice boards and word-of-mouth communication.

The backend database, designed in **3NF**, comprises four main entities Users, Locations, Items, and Claims with proper foreign key relationships ensuring data consistency and referential integrity. Core database logic is implemented through **stored procedures**, **functions**, and **triggers**, such as automated claim approvals and timestamped item entries. The front-end application supports complete **CRUD operations**, dynamic data visualization using **Treeviews**, and interactive modules for handling claims and administrative tasks.

Additionally, the system incorporates **aggregate**, **join**, and **nested queries** for analytical insights, such as identifying users with above-average item reports or category-wise item distributions. The implementation highlights secure database connectivity, user feedback mechanisms, and role-based access through Python's GUI framework.

By integrating database design with a functional user interface, the project demonstrates key DBMS concepts, including **procedural SQL, triggers, normalization, and real-time data manipulation**, ensuring efficient, transparent, and scalable management of lost-and-found records.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

The **Lost and Found Management System (LFMS)** is a database-centric desktop application designed to streamline the process of reporting, tracking, and claiming lost or found items within an institution or organization. Traditionally, lost and found processes rely on manual communication or physical notice boards, leading to inefficiency, data loss, and delays in returning items to their rightful owners. This system replaces such manual methods with a **digital, centralized, and interactive platform**.

Developed using **Python (Tkinter)** for the front-end interface and **MySQL** for the back-end database, the system enables users to perform complete **CRUD operations (Create, Read, Update, Delete)** on entities like users, items, locations, and claims. It supports **role-based access**, allowing different privileges for students, staff, and administrators. The integration of **stored procedures, functions, and triggers** enhances automation and ensures data integrity within the database.

The application offers multiple analytical query features such as join, nested, and aggregate queries to generate insights about item trends, user activity, and claim patterns. Through its simple GUI and normalized database structure, the system ensures data consistency, reliability, and ease of management, making it a scalable and practical DBMS mini-project solution.

2. PROBLEM DEFINITION

In most organizations and institutions, managing **lost and found items** remains a manual and inefficient process. Lost belongings are often reported verbally, noted on notice boards, or left untracked, resulting in mismanagement, unclaimed items, and lack of accountability. As the volume of lost and found reports increases, it becomes difficult to maintain accurate records or identify matches between reported and found items.

The absence of a centralized, automated system leads to several key problems:

1. **Lack of traceability** – No structured way to track item status (lost, found, claimed).
2. **Data redundancy** – Repeated manual entries without standardization cause duplication and inconsistency.
3. **Delayed recovery** – Without digital search or filtering mechanisms, matching lost and found records takes significant time.
4. **Limited visibility** – Administrators cannot easily monitor or analyze trends in item loss or recovery.
5. **Error-prone handling** – Manual operations increase the risk of data loss, incomplete updates, or unauthorized modifications.

To address these issues, the proposed **Lost and Found Management System (LFMS)** provides a **database-driven digital platform** that manages users, items, and claims efficiently. The system allows users to:

- Report lost or found items with relevant details.
- View and search for existing reports.
- Submit or approve claims on found items.
- Allow administrators to monitor and manage all operations seamlessly.

3. ER MODEL

The Entity–Relationship (ER) Model of the Lost and Found Management System (LFMS) represents the logical structure of the database and the relationships among its entities. It forms the foundation for understanding how users, items, locations, and claims interact within the system.

1. **User**
    - Attributes: user_id (PK), name, email, phone, role
    - Description: Represents individuals using the system — either as students, staff, or administrators. Each user can report items and claim found ones.

2. **Location**
    - Attributes: location_id (PK), location_name, building, floor_no
    - Description: Represents the physical places where items are lost or found. Helps categorize and localize reports.

3. **Item**
    - Attributes: item_id (PK), item_name, description, category, status, report_date, reported_by (FK), location_id (FK)
    - Description: Core entity of the system that tracks the details of each reported lost or found item. Each item is linked to a user and a location.

4. **Claim**
    - Attributes: claim_id (PK), item_id (FK), claimer_id (FK), claim_date, status, remarks
    - Description: Represents a claim request submitted by a user to claim ownership of a found item. It also records administrative decisions (approved/rejected).

Relationships

- User → Item:
  One user can report multiple items, but each item is reported by exactly one user.
  (One-to-Many relationship)
- User → Claim:
  One user can submit multiple claims for different items.
  (One-to-Many relationship)
- Location → Item:
  Each item is associated with one location, but a location can have multiple items reported there.
  (One-to-Many relationship)

- Item → Claim:
  Each claim is linked to one item, and an item can have multiple claims until one is approved.
  (One-to-Many relationship)

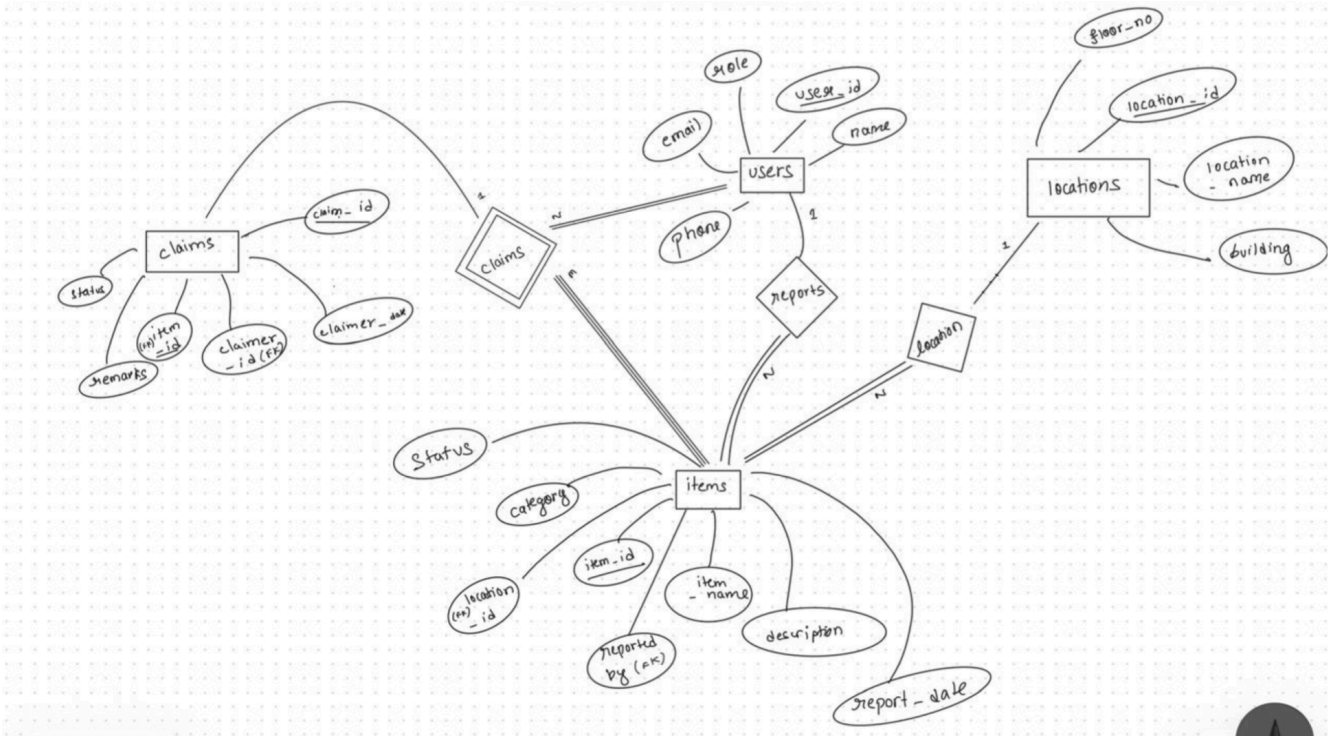
ER Model Summary

The ER model ensures:

- 3NF normalization, avoiding redundancy and maintaining referential integrity.
- Complete data traceability from users to items and claims.
- Efficient retrieval through well-defined relationships and key constraints.

(If you're including diagrams in your report, you can label this as "Figure 1: ER Diagram of Lost & Found Management System.")

Figure 2.1 - ER Diagram



4. ER TO RELATIONAL MAPPING

The **ER to Relational Mapping** process converts the logical design of the Lost and Found Management System (LFMS) into relational database tables. Each entity and relationship identified in the ER model is represented as a table in the MySQL database, ensuring normalization and referential integrity.

Mapping of Entities

1. USER → USERS Table

- **Primary Key:** user_id
- **Attributes:** user_id, name, email, phone, role
- **Description:** Stores information of all users including students, staff, and admins.

```sql
CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(15),
    role ENUM('student','staff','admin') DEFAULT 'student'
);
```

2. LOCATION → LOCATIONS Table

- **Primary Key:** location_id
- **Attributes:** location_id, location_name, building, floor_no
- **Description:** Maintains details of all locations where items can be lost or found.

```sql
CREATE TABLE locations (
    location_id INT AUTO_INCREMENT PRIMARY KEY,
    location_name VARCHAR(100) NOT NULL,
    building VARCHAR(50),
    floor_no INT
);
```

3. ITEM → ITEMS Table

- **Primary Key:** item_id
- **Foreign Keys:** reported_by → users(user_id), location_id → locations(location_id)
- **Attributes:** item_id, item_name, description, category, status, report_date, reported_by, location_id
- **Description:** Records details of all reported items along with their reporter and location.

```
CREATE TABLE items (
    item_id INT AUTO_INCREMENT PRIMARY KEY,
    item_name VARCHAR(100) NOT NULL,
    description TEXT,
    category VARCHAR(50),
    status ENUM('lost','found','claimed') DEFAULT 'lost',
    report_date DATE DEFAULT (CURRENT_DATE),
    reported_by INT,
    location_id INT,
    FOREIGN KEY (reported_by) REFERENCES users(user_id),
    FOREIGN KEY (location_id) REFERENCES locations(location_id)
);
```

4. CLAIM → CLAIMS Table

- **Primary Key:** claim_id
- **Foreign Keys:** item_id → items(item_id), claimer_id → users(user_id)
- **Attributes:** claim_id, item_id, claimer_id, claim_date, status, remarks
- **Description:** Tracks all claims made on found items, including remarks and approval status.

```sql
CREATE TABLE claims (
    claim_id INT AUTO_INCREMENT PRIMARY KEY,
    item_id INT,
    claimer_id INT,
    claim_date DATE DEFAULT (CURRENT_DATE),
    status ENUM('pending','approved','rejected') DEFAULT 'pending',
    remarks VARCHAR(255),
    FOREIGN KEY (item_id) REFERENCES items(item_id),
    FOREIGN KEY (claimer_id) REFERENCES users(user_id)
);
```

Mapping of Relationships

Table 1.1 - Mapping of Relationships

Relationship	Type	Implementation
User – Item	1:M	reported_by foreign key in items
User – Claim	1:M	claimer_id foreign key in claims
Location – Item	1:M	location_id foreign key in items
Item – Claim	1:M	item_id foreign key in claims

4.1 STEPS OF ALGORITHM FOR CHOSEN PROBLEM

The algorithmic workflow of the **Lost and Found Management System (LFMS)** outlines how the system handles item reporting, tracking, and claiming. The steps ensure smooth interaction between the user interface and the database while maintaining data consistency and validation.

Algorithm: Lost and Found Management Process

Step 1:
Initialize the connection between the Python (Tkinter) front end and the MySQL database using connection parameters like host, user, password, and database name.

Step 2:
Display the main application window with navigable tabs Users, Locations, Items, Claims, and Queries.

Step 3:
When a new user registers or is added:
→ Validate input fields.
→ Insert the user record into the users table using an SQL INSERT command.
→ Refresh all dropdown menus that depend on user data.

Step 4:
When a new location is added:
→ Take inputs (location name, building, floor number).
→ Insert into the locations table.
→ Update the location dropdown used in item reporting.

Step 5:
For a lost or found item report:

→ Collect item details from the GUI.

→ Execute the stored procedure add_item() which inserts data into the items table.

→ A trigger trg_item_insert automatically appends a timestamp to the item's description.

→ Refresh the item display table.

Step 6:

For claim processing:

→ User enters the item ID and remarks to claim ownership.

→ The system inserts a record into the claims table.

→ When the admin approves/rejects, the stored procedure update_claim_status() is called.

→ Trigger trg_claim_approve updates the item's status to "claimed" if the claim is approved.

Step 7:

Run analytical or custom queries (Nested, Join, Aggregate) to view system insights, e.g.,

- Items per category
- Users with most reports
- Pending claims

Step 8:

Display query results dynamically using Tkinter's Treeview widget, ensuring user-friendly tabular visualization.

Step 9:

Handle invalid inputs or database errors gracefully using message boxes to ensure robustness and reliability.

4.2 COMPLETE DIAGRAM OF RELATIONAL MAPPING

The **Complete Relational Mapping Diagram** shows how all entities in the Lost and Found Management System are transformed into relational tables, along with their **primary keys (PK)** and **foreign keys (FK)** that define inter-table relationships.

Figure 2.2 - Relational Schema

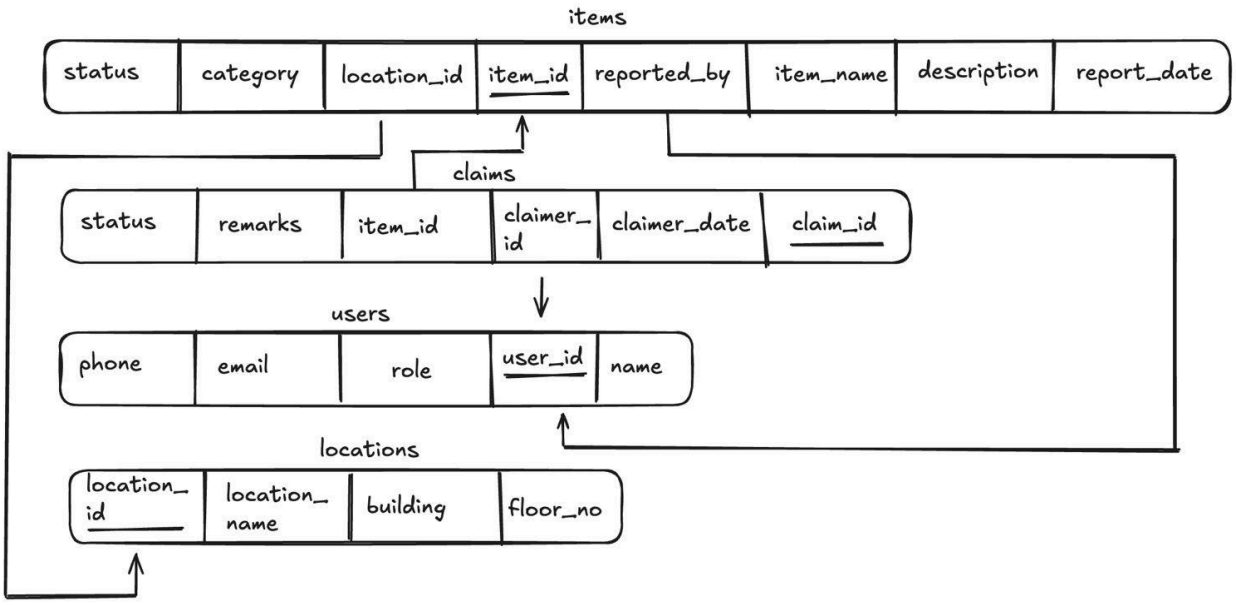


Table 1.2 - Complete Relation Map

Relationship	Cardinality	Description
USERS → ITEMS	1 : M	One user can report multiple items.
USERS → CLAIMS	1 : M	One user can make multiple claims.
LOCATIONS → ITEMS	1 : M	Each location can have several reported items.
ITEMS → CLAIMS	1 : M	An item can have multiple claims, but only one approved.

5. DDL STATEMENTS

STATEMENTS WITH SCREEN SHOTS OF THE TABLE CREATION

The **Data Definition Language (DDL)** statements are used to define and manage the structure of the database schema for the **Lost and Found Management System (LFMS)**.
These statements create the necessary tables, constraints, and relationships to ensure normalized and consistent data storage.

5.1 Database Creation

```
DROP DATABASE IF EXISTS lostfound;
CREATE DATABASE lostfound;
USE lostfound;
```

5.2 Table Creation

a. Users Table

```
CREATE TABLE users (
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
  phone VARCHAR(15),
  role ENUM('student','staff','admin') DEFAULT 'student'
);
```

b. LOCATIONS Table

```
CREATE TABLE locations (
  location_id INT AUTO_INCREMENT PRIMARY KEY,
  location_name VARCHAR(100) NOT NULL,
  building VARCHAR(50),
  floor_no INT
);
```

c. ITEMS Table

```
CREATE TABLE items (
  item_id INT AUTO_INCREMENT PRIMARY KEY,
  item_name VARCHAR(100) NOT NULL,
  description TEXT,
  category VARCHAR(50),
  status ENUM('lost','found','claimed') DEFAULT 'lost',
  report_date DATE DEFAULT (CURRENT_DATE),
  reported_by INT,
  location_id INT,
  FOREIGN KEY (reported_by) REFERENCES users(user_id),
  FOREIGN KEY (location_id) REFERENCES locations(location_id)
);
```

d. CLAIMS Table

```
CREATE TABLE claims (
  claim_id INT AUTO_INCREMENT PRIMARY KEY,
  item_id INT,
  claimer_id INT,
  claim_date DATE DEFAULT (CURRENT_DATE),
  status ENUM('pending','approved','rejected') DEFAULT 'pending',
  remarks VARCHAR(255),
  FOREIGN KEY (item_id) REFERENCES items(item_id),
  FOREIGN KEY (claimer_id) REFERENCES users(user_id)
);
```

Description of Tables -

Figure 2.3 - Table Descriptions

```
mysql> desc claims;
+------------+-----------------------------------------+------+-----+-----------+-------------------+
| Field      | Type                                    | Null | Key | Default   | Extra             |
+------------+-----------------------------------------+------+-----+-----------+-------------------+
| claim_id   | int                                     | NO   | PRI | NULL      | auto_increment    |
| item_id    | int                                     | YES  | MUL | NULL      |                   |
| claimer_id | int                                     | YES  | MUL | NULL      |                   |
| claim_date | date                                    | YES  |     | curdate() | DEFAULT_GENERATED |
| status     | enum('pending','approved','rejected') | YES  |     | pending   |                   |
| remarks    | varchar(255)                            | YES  |     | NULL      |                   |
+------------+-----------------------------------------+------+-----+-----------+-------------------+
6 rows in set (0.01 sec)

mysql> desc items;
+-------------+--------------------------------+------+-----+-----------+-------------------+
| Field       | Type                           | Null | Key | Default   | Extra             |
+-------------+--------------------------------+------+-----+-----------+-------------------+
| item_id     | int                            | NO   | PRI | NULL      | auto_increment    |
| item_name   | varchar(100)                   | NO   |     | NULL      |                   |
| description | text                           | YES  |     | NULL      |                   |
| category    | varchar(50)                    | YES  |     | NULL      |                   |
| status      | enum('lost','found','claimed') | YES  |     | lost      |                   |
| report_date | date                           | YES  |     | curdate() | DEFAULT_GENERATED |
| reported_by | int                            | YES  | MUL | NULL      |                   |
| location_id | int                            | YES  | MUL | NULL      |                   |
+-------------+--------------------------------+------+-----+-----------+-------------------+
8 rows in set (0.00 sec)

mysql> desc locations;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| location_id   | int          | NO   | PRI | NULL    | auto_increment |
| location_name | varchar(100) | NO   |     | NULL    |                |
| building      | varchar(50)  | YES  |     | NULL    |                |
| floor_no      | int          | YES  |     | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

```
mysql> desc users;
+---------+-----------------------------------+------+-----+---------+----------------+
| Field   | Type                              | Null | Key | Default | Extra          |
+---------+-----------------------------------+------+-----+---------+----------------+
| user_id | int                               | NO   | PRI | NULL    | auto_increment |
| name    | varchar(100)                      | NO   |     | NULL    |                |
| email   | varchar(100)                      | NO   | UNI | NULL    |                |
| phone   | varchar(15)                       | YES  |     | NULL    |                |
| role    | enum('student','staff','admin') | YES  |     | student |                |
+---------+-----------------------------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

5.3 Constraints Summary

Table 1.3 - Constraints Summary

Table	Constraint Type	Columns	Description
USERS	PRIMARY KEY	user_id	Unique identification for each user
LOCATIONS	PRIMARY KEY	location_id	Unique location identification
ITEMS	FOREIGN KEY	reported_by → users(user_id)	Maintains link between item and reporter
ITEMS	FOREIGN KEY	location_id → locations(location_id)	Ensures valid location references
CLAIMS	FOREIGN KEY	item_id → items(item_id)	Links claim to corresponding item
CLAIMS	FOREIGN KEY	claimer_id → users(user_id)	Tracks who made the claim

5.4 Normalization

- The schema is designed in Third Normal Form (3NF):
- No repeating groups or multi-valued attributes (1NF)
- All non-key attributes depend on the primary key (2NF)
- No transitive dependencies (3NF)

6. DML STATEMENTS

STATEMENTS WITH SCREEN SHOTS OF THE TABLE WITH INSERTED VALUES

Figure 2.4 - DML Commands

```
mysql> select * from claims
    -> ;
+----------+---------+------------+------------+----------+----------------------+
| claim_id | item_id | claimer_id | claim_date | status   | remarks              |
+----------+---------+------------+------------+----------+----------------------+
|        1 |       2 |          1 | 2025-11-06 | pending  | Looks like my bottle |
|        2 |       1 |          2 | 2025-11-06 | approved | Owner confirmed      |
+----------+---------+------------+------------+----------+----------------------+
2 rows in set (0.01 sec)

mysql> select * from items;
+---------+---------------+-----------------------------+-------------+--------+-------------+-------------+-------------+
| item_id | item_name     | description                 | category    | status | report_date | reported_by | location_id |
+---------+---------------+-----------------------------+-------------+--------+-------------+-------------+-------------+
|       1 | Black Wallet  | Leather wallet with ID card | Accessories | lost   | 2025-11-06  |           1 |           1 |
|       2 | Water Bottle  | Blue Milton bottle          | Daily Use   | found  | 2025-11-06  |           2 |           2 |
|       3 | Laptop Charger | HP charger 65W             | Electronics | lost   | 2025-11-06  |           1 |           3 |
+---------+---------------+-----------------------------+-------------+--------+-------------+-------------+-------------+
3 rows in set (0.00 sec)

mysql> select * from locations;
+-------------+---------------+----------+----------+
| location_id | location_name | building | floor_no |
+-------------+---------------+----------+----------+
|           1 | Library       | Block A  |        1 |
|           2 | Cafeteria     | Block B  |        0 |
|           3 | Main Ground   | Block C  |        0 |
+-------------+---------------+----------+----------+
3 rows in set (0.00 sec)

mysql> select * from users;
+---------+--------------+-----------------------+------------+---------+
| user_id | name         | email                 | phone      | role    |
+---------+--------------+-----------------------+------------+---------+
|       1 | Shivam Anand | shivam@example.com    | 9999999999 | student |
|       2 | Priya Verma  | priya@example.com     | 8888888888 | staff   |
|       3 | Ravi Adram   | ravi.admin@example.com | 7777777777 | admin   |
+---------+--------------+-----------------------+------------+---------+
3 rows in set (0.01 sec)
```

7. QUERIES

7.1 SIMPLE QUERY WITH GROUP BY, AGGREGATE

Figure 2.5 - Aggregate Query

```
mysql> SELECT
    ->     category,
    ->     COUNT(item_id) AS total_items
    -> FROM items
    -> GROUP BY category
    -> ORDER BY total_items DESC;
+-------------+-------------+
| category    | total_items |
+-------------+-------------+
| Accessories |           1 |
| Daily Use   |           1 |
| Electronics |           1 |
+-------------+-------------+
3 rows in set (0.00 sec)
```

7.2 UPDATE OPERATION

Figure 2.6 - Update Query

```
mysql> UPDATE items
    -> SET description = 'Updated description for testing'
    -> WHERE item_id = 1;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> select * from items;
+---------+--------------+---------------------------------+-------------+--------+-------------+-------------+-------------+
| item_id | item_name    | description                     | category    | status | report_date | reported_by | location_id |
+---------+--------------+---------------------------------+-------------+--------+-------------+-------------+-------------+
|       1 | Black Wallet | Updated description for testing | Accessories | lost   | 2025-11-06  |           1 |           1 |
|       2 | Water Bottle | Blue Milton bottle              | Daily Use   | found  | 2025-11-06  |           2 |           2 |
|       3 | Laptop Charger | HP charger 65W                | Electronics | lost   | 2025-11-06  |           1 |           3 |
+---------+--------------+---------------------------------+-------------+--------+-------------+-------------+-------------+
3 rows in set (0.00 sec)
```

7.3 DELETE OPERATION

Figure 2.7 - Delete Query

```
mysql> DELETE FROM claims
    -> WHERE claim_id = 1;
Query OK, 1 row affected (0.00 sec)

mysql> select * from claims;
+----------+---------+------------+------------+----------+-----------------+
| claim_id | item_id | claimer_id | claim_date | status   | remarks         |
+----------+---------+------------+------------+----------+-----------------+
|        2 |       1 |          2 | 2025-11-06 | approved | Owner confirmed |
+----------+---------+------------+------------+----------+-----------------+
1 row in set (0.00 sec)
```

7.4 CORRELATED QUERY

Figure 2.8 - Correlated Query

```
mysql> SELECT
    ->     u.user_id,
    ->     u.name
    -> FROM users u
    -> WHERE EXISTS (
    ->     SELECT 1
    ->     FROM items i
    ->     WHERE i.reported_by = u.user_id
    -> );
+---------+--------------+
| user_id | name         |
+---------+--------------+
|       1 | Shivam Anand |
|       2 | Priya Verma  |
+---------+--------------+
2 rows in set (0.00 sec)
```

7.5 NESTED QUERY

Figure 2.9 - Nested Query

```
mysql> SELECT name, user_id
    -> FROM users
    -> WHERE user_id IN (
    ->     SELECT reported_by
    ->     FROM items
    ->     GROUP BY reported_by
    ->     HAVING COUNT(item_id) > (
    ->         SELECT AVG(item_count)
    ->         FROM (
    ->             SELECT COUNT(item_id) AS item_count
    ->             FROM items
    ->             GROUP BY reported_by
    ->         ) AS sub
    ->     )
    -> );
+--------------+---------+
| name         | user_id |
+--------------+---------+
| Shivam Anand |       1 |
+--------------+---------+
1 row in set (0.00 sec)
```

8. STORED PROCEDURES, FUNCTIONS AND TRIGGERS

8.1 STORED PROCEDURES OR FUNCTIONS

Procedures -

```
mysql> CALL add_item(
    ->     'Red Umbrella',
    ->     'Forgot near cafeteria',
    ->     'Accessories',
    ->     'lost',
    ->     1,
    ->     2
    -> );
Query OK, 1 row affected (0.01 sec)

mysql> select * from items;
+---------+---------------+-----------------------------------------------------+-------------+--------+-------------+-------------+-------------+
| item_id | item_name     | description                                         | category    | status | report_date | reported_by | location_id |
+---------+---------------+-----------------------------------------------------+-------------+--------+-------------+-------------+-------------+
|       1 | Black Wallet  | Updated description for testing                     | Accessories | lost   | 2025-11-06  |           1 |           1 |
|       2 | Water Bottle  | Blue Milton bottle                                  | Daily Use   | found  | 2025-11-06  |           2 |           2 |
|       3 | Laptop Charger | HP charger 65W                                      | Electronics | lost   | 2025-11-06  |           1 |           3 |
|       4 | Red Umbrella  | Forgot near cafeteria [Added on: 2025-11-13 08:49:42] | Accessories | lost   | 2025-11-13  |           1 |           2 |
+---------+---------------+-----------------------------------------------------+-------------+--------+-------------+-------------+-------------+
4 rows in set (0.00 sec)
```

```
mysql> CALL update_claim_status(1, 'approved', 'Verified by admin');
Query OK, 0 rows affected (0.00 sec)

mysql> select * from claims;
+----------+---------+------------+------------+----------+------------------+
| claim_id | item_id | claimer_id | claim_date | status   | remarks          |
+----------+---------+------------+------------+----------+------------------+
|        2 |       1 |          2 | 2025-11-06 | approved | Owner confirmed  |
+----------+---------+------------+------------+----------+------------------+
1 row in set (0.00 sec)
```

Figure 2.10 - Procedures

Function -

Figure 2.11 - Function

```
mysql> SELECT count_items_by_user(1);
+----------------------------+
| count_items_by_user(1) |
+----------------------------+
|                          3 |
+----------------------------+
1 row in set (0.00 sec)
```

```
DELIMITER //
CREATE FUNCTION count_items_by_user(p_user_id INT)
RETURNS INT
DETERMINISTIC
BEGIN
  DECLARE total INT;
  SELECT COUNT(*) INTO total FROM items WHERE reported_by = p_user_id;
  RETURN total;
END //
DELIMITER ;
```

8.2 TRIGGERS

Trigger 1 -

Figure 2.12 - Triggers

```
DELIMITER //
CREATE TRIGGER trg_item_insert
BEFORE INSERT ON items
FOR EACH ROW
BEGIN
  -- Automatically append timestamp info to description
  SET NEW.description = CONCAT(
      COALESCE(NEW.description, ''),
      ' [Added on: ', DATE_FORMAT(NOW(), '%Y-%m-%d %H:%i:%s'), ']'
  );
END //
DELIMITER ;
```

```
mysql> INSERT INTO items(item_name, description, category, status, reported_by, location_id)
    -> VALUES ('Trigger Test Item', 'This is a test description', 'Test', 'found', 1, 1);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT item_id, item_name, description
    -> FROM items
    -> WHERE item_name = 'Trigger Test Item';
+---------+-------------------+-----------------------------------------------------------------+
| item_id | item_name         | description                                                     |
+---------+-------------------+-----------------------------------------------------------------+
|       5 | Trigger Test Item | This is a test description [Added on: 2025-11-13 08:56:47] |
+---------+-------------------+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

Trigger 2 -

```
DELIMITER //
CREATE TRIGGER trg_claim_approve
AFTER UPDATE ON claims
FOR EACH ROW
BEGIN
  IF NEW.status = 'approved' THEN
    UPDATE items
    SET status='claimed',
        description = CONCAT(
            COALESCE(description,''),
            ' [Claim approved on ', DATE_FORMAT(NOW(), '%Y-%m-%d %H:%i:%s'), ']'
        )
    WHERE item_id=NEW.item_id;
  END IF;
END //
DELIMITER ;
```

```
mysql> SELECT item_id, item_name, status, description
    -> FROM items
    -> WHERE item_id = 2;
+---------+--------------+--------+--------------------+
| item_id | item_name    | status | description        |
+---------+--------------+--------+--------------------+
|       2 | Water Bottle | found  | Blue Milton bottle |
+---------+--------------+--------+--------------------+
1 row in set (0.00 sec)

mysql> CALL update_claim_status(3, 'approved', 'Verified');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT item_id, item_name, status, description
    -> FROM items
    -> WHERE item_id = 2;
+---------+--------------+---------+-----------------------------------------------------------------+
| item_id | item_name    | status  | description                                                     |
+---------+--------------+---------+-----------------------------------------------------------------+
|       2 | Water Bottle | claimed | Blue Milton bottle [Claim approved on 2025-11-14 08:35:25] |
+---------+--------------+---------+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

9. FRONT END DEVELOPMENT REFERENCES

• **Python Tkinter –** Standard Python GUI toolkit used for building all user interface components.

• **Tkinter.ttk –** Themed widgets (Treeview, Combobox, Notebook tabs) used for tables, dropdowns, and tabbed layout.

• **messagebox (Tkinter) –** Used for dialog alerts and notifications.

• **Python MySQL Connector (mysql-connector-python) –** Used for connecting the Tkinter GUI to the MySQL database, executing SQL queries, procedures, and functions.

• **Python Standard Library –** Core modules used in the project (datetime, string handling, etc.)

• **MySQL Workbench / MySQL Server –** Backend database used with the Tkinter GUI.

GitHub Link

https://github.com/pes2ug23cs549/dbms-mini-pr