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. Zaara Shaikh of S.Y. Information Technology (SYIT) Roll No. 2542040 has satisfactorily completed the Open Source DataBase Management System Mini Project entitled **Ration Card & Beneficiary Management 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

INDEX

CHAPTER 1 INTRODUCTION

The Public Distribution System (PDS) is an essential government initiative aimed at providing food grains and basic commodities to eligible citizens at subsidized rates. Managing ration cards, beneficiaries, stock, and distribution records manually often leads to errors, duplication, and lack of transparency. With the increasing number of beneficiaries, an efficient and reliable database system is required.

The Ration Card & Beneficiary Management Database project is developed using MySQL as an Open Source Database Management System. The system maintains structured records of ration cards, family members, stock items, officers, and distribution transactions. It ensures accuracy, consistency, and integrity of data through proper database design and OSDBMS concepts.

CHAPTER 2 PROBLEM DEFINITION

The existing manual ration management system suffers from several problems such as duplicate ration cards, improper stock tracking, excess ration distribution, and lack of accountability. Maintaining records on paper makes it difficult to verify beneficiary details and monitor stock availability in real time.

The problem is to design a database system that can efficiently store, manage, and retrieve ration card and distribution data while preventing inconsistencies and misuse of resources.

CHAPTER 3 OBJECTIVES OF THE PROJECT

The objectives of this project are:

1. To design a structured and normalized database for ration card management.
2. To maintain accurate records of beneficiaries and family members.
3. To manage stock details of ration items effectively.
4. To record ration distribution transactions securely.
5. To prevent duplicate and excess ration allocation.
6. To implement OSDBMS concepts such as constraints, triggers, and transactions.

CHAPTER 4 SCOPE OF THE PROJECT

The scope of the project is limited to database design and implementation using MySQL. The system focuses on managing ration card details, stock, and distribution records at a ration distribution center. User interface and web application development are not included. However, the database can be extended and integrated with application software in the future.

CHAPTER 5 REQUIREMENT SPECIFICATION

Hardware Requirements	Software Requirements
Computer System	Operating System: Window /Linux
Minimum 4 GB RAM	Database: MySQL(Open Source)
Hard Disk	Tool: MySQL Workbench

CHAPTER 6 SYSTEM DESIGN

The system design defines the overall structure of the database system. The design is modular and consists of separate tables for ration cards, family members, stock items, officers, and distribution transactions. This modular approach improves data integrity and reduces redundancy.

The system follows a centralized database design where all data is stored in a single database and accessed through SQL queries.

CHAPTER 7 DATABASE DESIGN

7.1 Entity Description

The database is designed using normalization principles up to Third Normal Form (3NF). Each table has a primary key, and relationships between tables are established using foreign keys. This design ensures minimal redundancy and maintains referential integrity.

The main tables used in the database are Ration_Card, Family_Members, Stock, Officer, and Distribution.

7.2 Table Structure

1. Ration_Card

Attribute	Data Type	Constraints / Notes
card_id	INT	PRIMARY KEY
card_type	VARCHAR(20)	CHECK (card_type IN ('APL','BPL','AAY'))
issue_date	DATE	
status	VARCHAR(10)	DEFAULT 'ACTIVE'

2. Family_Members

Attribute	Data Type	Constraints / Notes
member_id	INT	PRIMARY KEY
card_id	INT	FOREIGN KEY → Ration_Card(card_id)
member_name	VARCHAR(50)	
age	INT	
relation	VARCHAR(30)	

3. Officer

Attribute	Data Type	Constraints / Notes
officer_id	INT	PRIMARY KEY
officer_name	VARCHAR(50)	
designation	VARCHAR(30)	

4. Stock

Attribute	Data Type	Constraints / Notes
item_id	INT	PRIMARY KEY
item_name	VARCHAR(30)	
quantity	INT	CHECK (quantity >= 0)

5. Distribution

Attribute	Data Type	Constraints / Notes
dist_id	INT	PRIMARY KEY
card_id	INT	FOREIGN KEY → Ration_Card(card_id)
item_id	INT	FOREIGN KEY → Stock(item_id)
quantity_given	INT	
dist_date	DATE	
officer_id	INT	FOREIGN KEY → Officer(officer_id)

7.3 Constraints Used

Constraints are used in the Water Supply Complaint Management System to maintain data accuracy, consistency, and integrity. They ensure that only valid and meaningful data is stored in the database. The following constraints are used extensively in this project:

PRIMARY KEY

The PRIMARY KEY constraint is used to uniquely identify each record in a table. In this project, primary keys are used in all major tables such as Area, User_Details, and Complaint. Each table has a unique identifier like area_id, user_id, or complaint_id. This ensures that every record can be uniquely accessed and prevents duplicate entries within the same table.

FOREIGN KEY

The FOREIGN KEY constraint is used to establish relationships between different tables. In this project, foreign keys are used to link user records with complaints and area records with users. This constraint ensures that records in one table always refer to valid records in another table. It prevents invalid or orphan records and maintains logical connections between complaint data.

NOT NULL

The NOT NULL constraint is used to ensure that important fields always contain values. In this project, attributes such as user_name, phone_no, complaint_type, and status are defined as NOT NULL. This ensures that incomplete or meaningless records are not stored in the database and that essential information is always available.

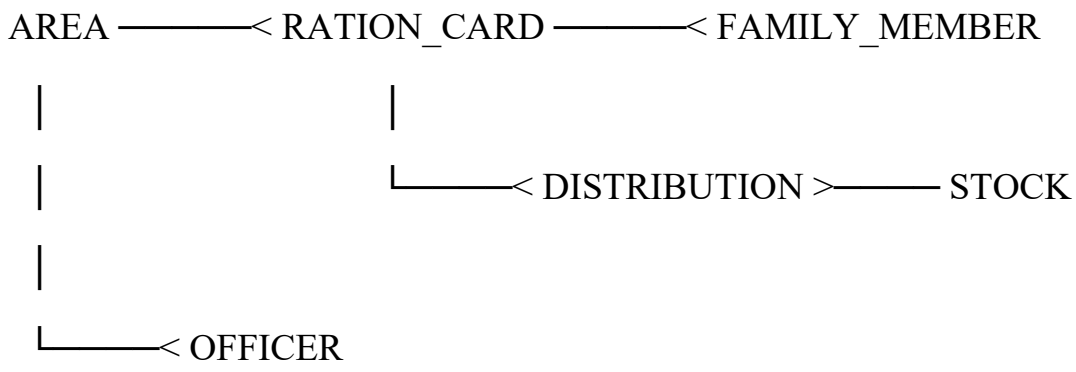
UNIQUE

The UNIQUE constraint is used to prevent duplicate entries in specific fields. In this project, it is mainly used for phone numbers to ensure that the same user contact number is not stored multiple times in the database. This reduces redundancy and maintains data accuracy.

CHAPTER 8 UML DAIGRAM

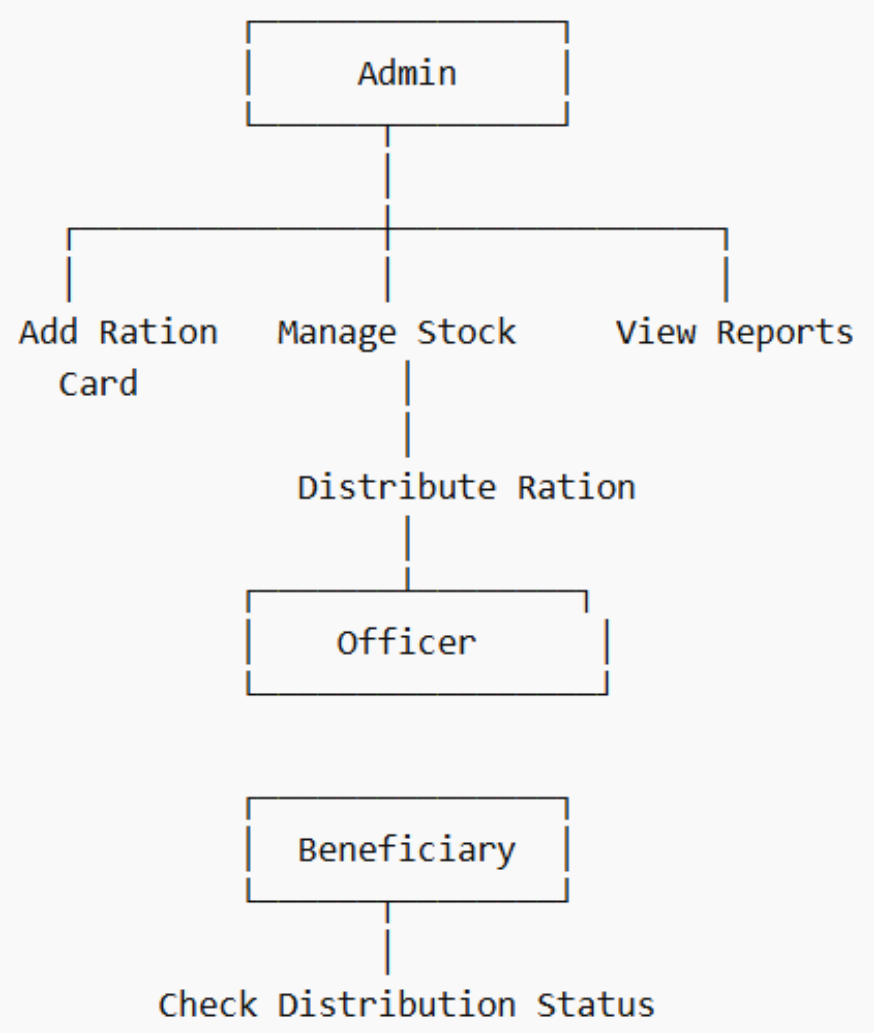
The ER diagram represents the logical structure of the database. The main entities in the system are Ration_Card, Family_Members, Stock, Officer, and Distribution.
Each ration card can have multiple family members. The Distribution entity links ration cards, stock items, and officers, ensuring proper tracking of ration distribution. Primary keys uniquely identify records, while foreign keys establish relationships between entities. This design ensures data integrity and avoids redundancy.

8.1 ER Diagram of Ration Card & Beneficiary Management System



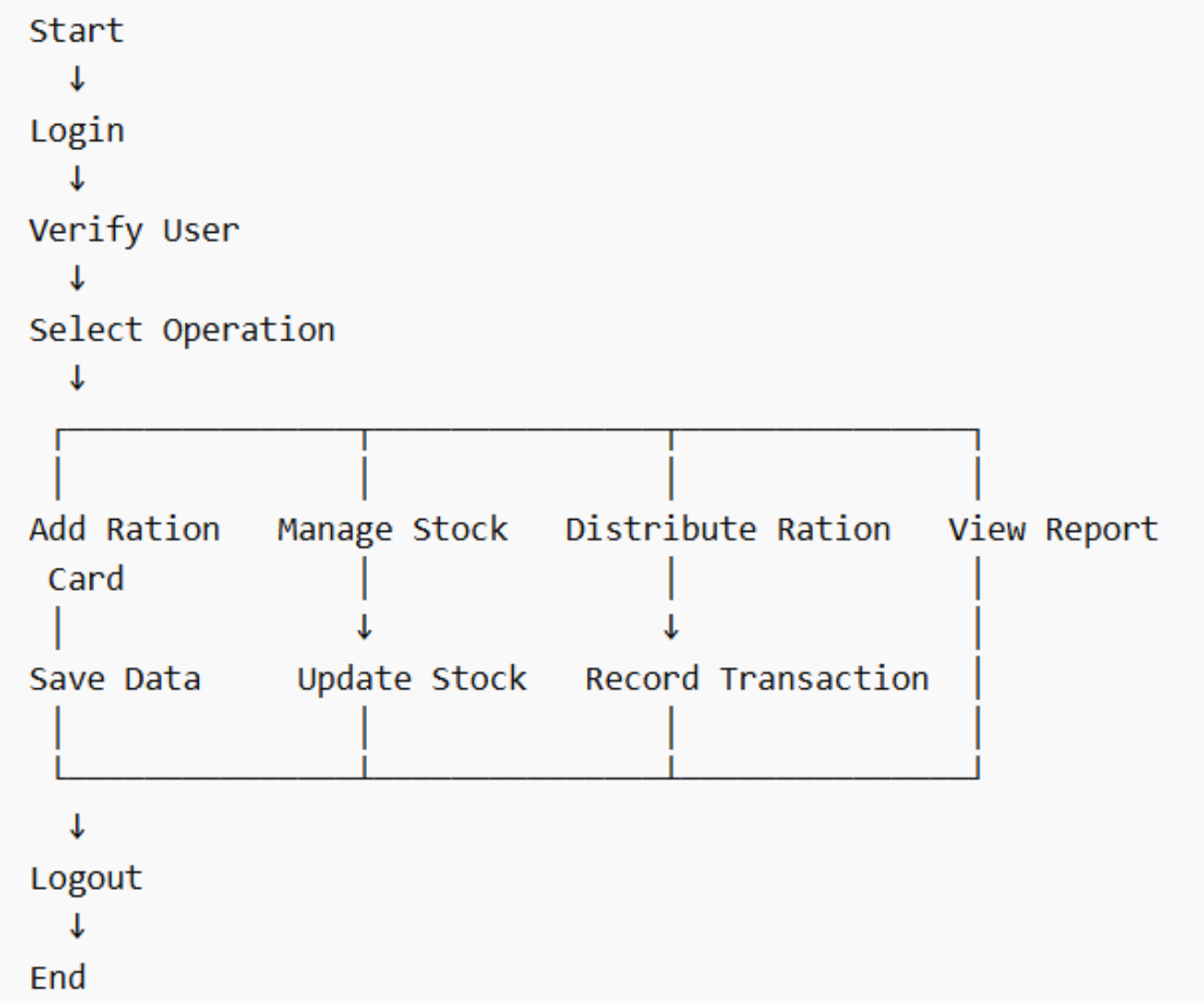
- **Area** entity stores location details and is related to multiple ration cards and officers.
- **Stock** entity keeps information about ration items and their quantity.
- **Distribution** entity records ration transactions between ration cards and stock.
- **Officer** entity represents officials responsible for managing distribution in an area.
- The diagram helps maintain data integrity and shows how all entities are connected.

8.2 Use Case Diagram of Ration Card & Beneficiary Management System



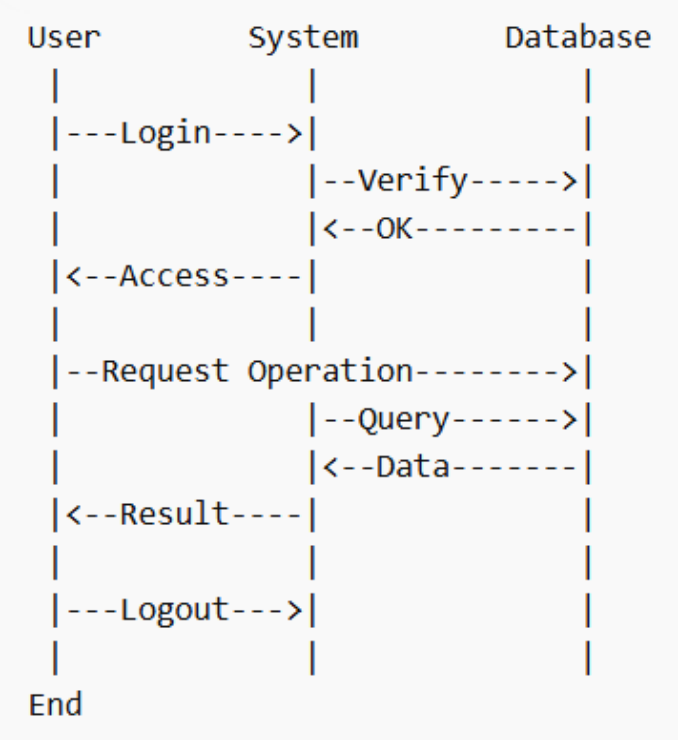
- Use Case Diagram shows interactions between users and the system.
- The main actors are Admin, Officer, and Beneficiary.
- Admin manages ration cards, stock, and reports.
- Officer handles ration distribution activities.
- Beneficiary can check distribution status.
- The diagram helps understand system functionality and user roles.

8.3 Activity Diagram of Ration Card & Beneficiary Management System



- Activity Diagram shows the workflow of the system.
- The process starts with user login and verification.
- User selects an operation such as adding ration card, managing stock, distributing ration, or viewing reports.
- The system processes the selected activity and updates the database.
- After completing tasks, the user logs out and the process ends.
- It helps understand the step-by-step working of the system.

8.4 Sequence Diagram of Ration Card & Beneficiary Management System



- Sequence Diagram shows interaction between User, System, and Database.
- The process starts when the user logs into the system.
- The system verifies user credentials from the database.
- After successful login, the user performs operations like viewing or updating data.
- The system sends requests to the database and retrieves results.
- Finally, the user logs out and the process ends.

CHAPTER 9 SQL IMPLEMENTATION

The SQL implementation is used to create and manage the database for the Ration Card and Beneficiary Management System. It includes database creation, table structures, constraints, triggers, and transactions to ensure data integrity and proper functioning of the system.

9.1 Database Creation

The database named **RationCardDB** is created to store all system data.

```
CREATE DATABASE RationCardDB;

USE RationCardDB;
```

9.2Table Creation

- **Ration_Card Table**

This table stores information about ration cards including card type, issue date, and status.

```
CREATE TABLE Ration_Card (
   card_id INT PRIMARY KEY,
   card_type VARCHAR(20) CHECK (card_type IN ('APL','BPL','AAY')),
   issue_date DATE,
   status VARCHAR(10) DEFAULT 'ACTIVE'
);
```

- **Family_Members Table**

Stores details of family members linked to a ration card.

```
CREATE TABLE Family_Members (
```

```
    member_id INT PRIMARY KEY,
    card_id INT,
    member_name VARCHAR(50),
    age INT,
    relation VARCHAR(30),
    FOREIGN KEY (card_id) REFERENCES Ration_Card(card_id)
);
```

- **Officer Table**

Stores details of officers responsible for ration distribution.

```
CREATE TABLE Officer (
    officer_id INT PRIMARY KEY,
    officer_name VARCHAR(50),
    designation VARCHAR(30)
);
```

- **Stock Table**

Stores ration item details and available quantity.

```
CREATE TABLE Stock (
    item_id INT PRIMARY KEY,
    item_name VARCHAR(30),
    quantity INT CHECK (quantity >= 0)
);
```

- **Distribution Table**

Records ration distribution transactions linking ration cards, stock items, and officers.

```
CREATE TABLE Distribution (
   dist_id INT PRIMARY KEY,
   card_id INT,
   item_id INT,
   quantity_given INT,
   dist_date DATE,
   officer_id INT,
   FOREIGN KEY (card_id) REFERENCES Ration_Card(card_id),
   FOREIGN KEY (item_id) REFERENCES Stock(item_id),
   FOREIGN KEY (officer_id) REFERENCES Officer(officer_id)
);
```

9.2 Trigger Implementation

A trigger is created to ensure that distribution cannot occur if sufficient stock is not available.

```
DELIMITER //

CREATE TRIGGER check_stock_before_distribution
BEFORE INSERT ON Distribution
FOR EACH ROW
BEGIN
```

```
    DECLARE available_stock INT;

    SELECT quantity INTO available_stock
    FROM Stock
    WHERE item_id = NEW.item_id;

    IF available_stock < NEW.quantity_given THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Not enough stock available';
    END IF;
END;
//

DELIMITER ;
```

9.3 Data Insertion

Sample data is inserted into tables to test system functionality.

```
INSERT INTO Ration_Card VALUES
(1001,'BPL','2023-01-10','ACTIVE'),
(1002,'APL','2022-05-12','ACTIVE'),
(1003,'AAY','2021-08-15','ACTIVE'),
(1004,'BPL','2023-03-21','ACTIVE'),
(1005,'APL','2022-07-11','ACTIVE'),
```

(1006,'BPL','2023-02-18','ACTIVE'),

(1007,'APL','2022-09-05','ACTIVE'),

(1008,'AAY','2021-11-22','ACTIVE'),

(1009,'BPL','2023-04-01','ACTIVE'),

(1010,'APL','2022-10-19','ACTIVE'),

(1011,'BPL','2023-06-25','ACTIVE'),

(1012,'AAY','2021-12-30','ACTIVE');

INSERT INTO Family_Members VALUES

(1,1001,'Zaara Shaikh',21,'Self'),

(2,1001,'Ahmed Shaikh',50,'Father'),

(3,1002,'Priya Sharma',30,'Self'),

(4,1003,'Ravi Patel',45,'Self'),

(5,1004,'Ayesha Khan',28,'Self'),

(6,1005,'Rahul Verma',35,'Self'),

(7,1006,'Fatima Shaikh',25,'Self'),

(8,1007,'Arjun Singh',40,'Self'),

(9,1008,'Meena Kumari',55,'Self'),

(10,1009,'Imran Khan',38,'Self');

INSERT INTO Officer VALUES

(201,'Mr. Khan','Food Inspector'),

(202,'Mr. Sharma','Supervisor'),

(203,'Ms. Patel','Clerk'),

(204,'Mr. Singh','Inspector'),

(205,'Ms. Verma','Officer'),

(206,'Mr. Gupta','Supervisor'),

(207,'Ms. Shaikh','Clerk'),

(208,'Mr. Khan','Inspector'),

(209,'Mr. Yadav','Officer'),

(210,'Ms. Sharma','Supervisor');

INSERT INTO Stock VALUES

(101,'Rice',500),

(102,'Wheat',400),

(103,'Sugar',300),

(104,'Oil',200),

(105,'Salt',150),

(106,'Dal',250),

(107,'Tea',180),

(108,'Milk Powder',120),

(109,'Maize',220);

Transaction Processing

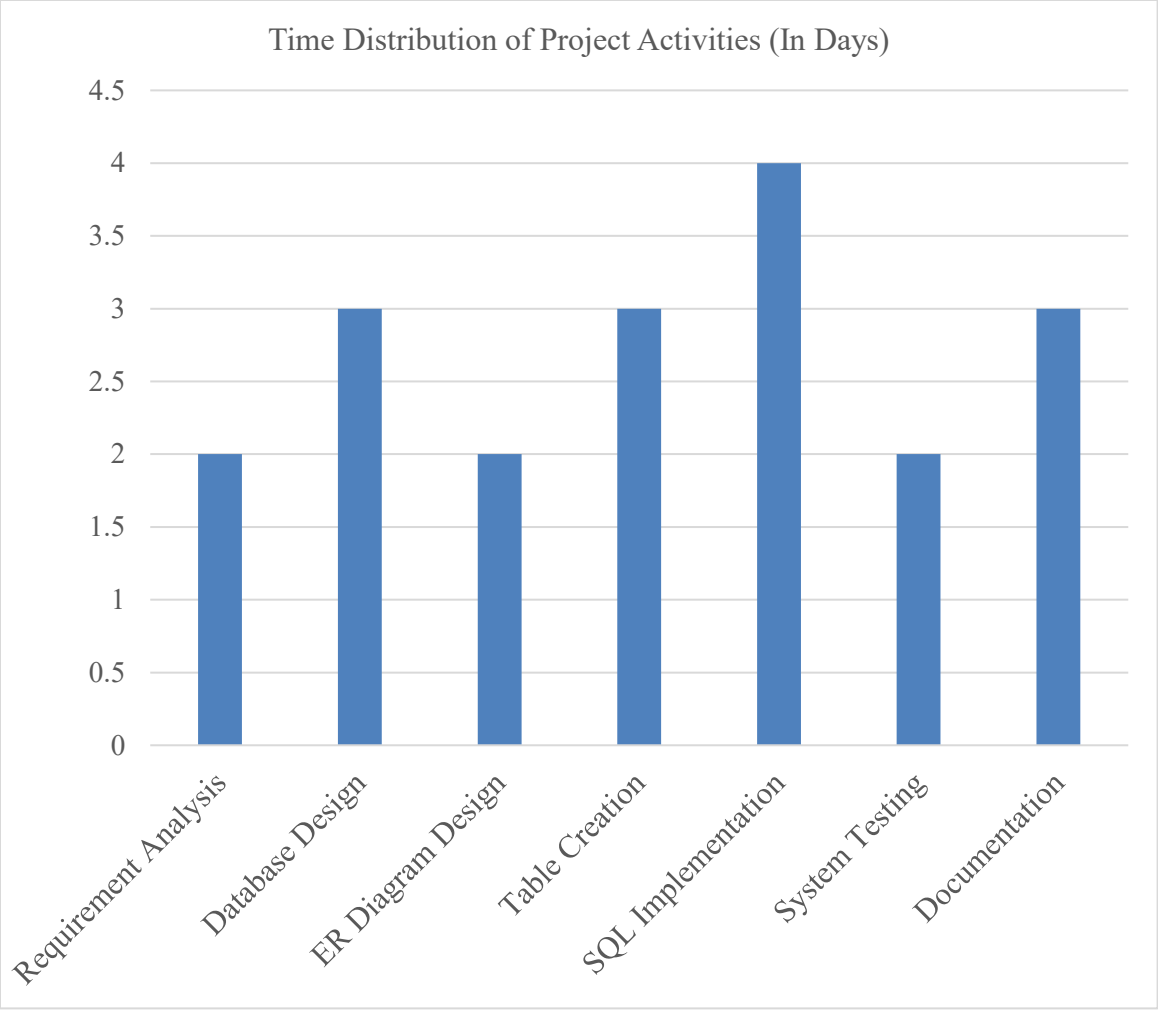
Transaction ensures that stock is updated and distribution is recorded atomically.

START TRANSACTION;

```
UPDATE Stock

SET quantity = quantity - 5

WHERE item_id = 101;


INSERT INTO Distribution

VALUES (1, 1001, 101, 5, CURDATE(), 201);

COMMIT;
```

The project was divided into structured phases to ensure proper planning and execution. Each activity such as requirement analysis, database design, table creation, SQL implementation, testing, and documentation was allocated time based on its complexity and importance.

The graph below shows the number of days spent on each phase, highlighting that the SQL implementation required the most time among all activities.



CHAPTER 10 SYSTEM TESTING AND RESULT

The system was tested using sample data to verify table relationships, trigger execution, and transaction handling. The test results confirm that stock is updated correctly and excess ration distribution is prevented.

```
mysql> show tables;
+-------------------------+
| Tables_in_rationcarddb1 |
+-------------------------+
| distribution            |
| family_members          |
| officer                 |
| ration_card             |
| stock                   |
+-------------------------+
5 rows in set (0.578 sec)
```

This output confirms that all required tables (Tables_in_rationcarddb1) were created successfully in the database.

```
mysql> select*from ration_card;
+---------+-----------+------------+--------+
| card_id | card_type | issue_date | status |
+---------+-----------+------------+--------+
|    1001 | BPL       | 2023-01-10 | ACTIVE |
|    1002 | APL       | 2022-05-12 | ACTIVE |
|    1003 | AAY       | 2021-08-15 | ACTIVE |
|    1004 | BPL       | 2023-03-21 | ACTIVE |
|    1005 | APL       | 2022-07-11 | ACTIVE |
|    1006 | BPL       | 2023-02-18 | ACTIVE |
|    1007 | APL       | 2022-09-05 | ACTIVE |
|    1008 | AAY       | 2021-11-22 | ACTIVE |
|    1009 | BPL       | 2023-04-01 | ACTIVE |
|    1010 | APL       | 2022-10-19 | ACTIVE |
|    1011 | BPL       | 2023-06-25 | ACTIVE |
|    1012 | AAY       | 2021-12-30 | ACTIVE |
+---------+-----------+------------+--------+
12 rows in set (0.023 sec)
```

This output verifies that ration records were inserted correctly into the ration_card table.

```
mysql> select*from family_members;
+-----------+---------+---------------+------+----------+
| member_id | card_id | member_name   | age  | relation |
+-----------+---------+---------------+------+----------+
|         1 |    1001 | Zaara Shaikh  |   21 | Self     |
|         2 |    1001 | Ahmed Shaikh  |   50 | Father   |
|         3 |    1002 | Priya Sharma  |   30 | Self     |
|         4 |    1003 | Ravi Patel    |   45 | Self     |
|         5 |    1004 | Ayesha Khan   |   28 | Self     |
|         6 |    1005 | Rahul Verma   |   35 | Self     |
|         7 |    1006 | Fatima Shaikh |   25 | Self     |
|         8 |    1007 | Arjun Singh   |   40 | Self     |
|         9 |    1008 | Meena Kumari  |   55 | Self     |
|        10 |    1009 | Imran Khan    |   38 | Self     |
+-----------+---------+---------------+------+----------+
10 rows in set (0.011 sec)
```

This output verifies that family records were inserted correctly into the family_members table.

```
mysql> select*from officer;
+------------+--------------+----------------+
| officer_id | officer_name | designation    |
+------------+--------------+----------------+
|        201 | Mr. Khan     | Food Inspector |
|        202 | Mr. Sharma   | Supervisor     |
|        203 | Ms. Patel    | Clerk          |
|        204 | Mr. Singh    | Inspector      |
|        205 | Ms. Verma    | Officer        |
|        206 | Mr. Gupta    | Supervisor     |
|        207 | Ms. Shaikh   | Clerk          |
|        208 | Mr. Khan     | Inspector      |
|        209 | Mr. Yadav    | Officer        |
|        210 | Ms. Sharma   | Supervisor     |
+------------+--------------+----------------+
10 rows in set (0.012 sec)
```

This output verifies that officer records were inserted correctly into the officer table.

```
mysql> select*from stock;
+---------+-------------+----------+
| item_id | item_name   | quantity |
+---------+-------------+----------+
|     101 | Rice        |      495 |
|     102 | Wheat       |      400 |
|     103 | Sugar       |      300 |
|     104 | Oil         |      200 |
|     105 | Salt        |      150 |
|     106 | Dal         |      250 |
|     107 | Tea         |      180 |
|     108 | Milk Powder |      120 |
|     109 | Maize       |      220 |
+---------+-------------+----------+
9 rows in set (0.011 sec)
```

This output verifies that stock records were inserted correctly into the stock table.

```
mysql> select*from distribution;
+---------+---------+---------+----------------+------------+------------+
| dist_id | card_id | item_id | quantity_given | dist_date  | officer_id |
+---------+---------+---------+----------------+------------+------------+
|       1 |    1001 |     101 |              5 | 2026-02-22 |        201 |
+---------+---------+---------+----------------+------------+------------+
1 row in set (0.008 sec)
```

This output verifies that distribution records were inserted correctly into the distribution table.

10.2 Data Validation

Data validation was performed using database constraints to ensure accuracy and integrity of stored information.

The following constraints were verified:

```
mysql> UPDATE Stock
    -> SET quantity = quantity - 9
    -> WHERE item_id = 101;
Query OK, 1 row affected (0.064 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

This query updates the quantity of the stock item with item_id 101 by reducing it by 9 units. The output confirms that one row was successfully updated with no warnings, indicating the stock quantity was modified correctly.

```
ysql> CREATE TRIGGER check_stock_before_distribution
   -> BEFORE INSERT ON Distribution
   -> FOR EACH ROW
   -> BEGIN
   ->     DECLARE available_stock INT;
   ->
   ->     SELECT quantity INTO available_stock
   ->     FROM Stock
   ->     WHERE item_id = NEW.item_id;
   ->
   ->     IF available_stock < NEW.quantity_given THEN
   ->         SIGNAL SQLSTATE '45000'
   ->         SET MESSAGE_TEXT = 'Not enough stock available';
   ->     END IF;
   -> END;
   -> //
Query OK, 0 rows affected (0.308 sec)
```

This trigger checks the available stock before inserting a record into the Distribution table.

If the available quantity is less than the required quantity, it shows an error message “Not enough stock available” and prevents the insertion.

CHAPTER 11 SECURITY, BACKUP AND RECOVERY

Database security is maintained through controlled access to the database. Backup and recovery can be performed using MySQL backup utilities to prevent data loss in case of system failure.

CHAPTER 12 FUTURE SCOPE AND CONCLUSION

The system can be extended by adding a web-based or mobile-based user interface, biometric authentication, and real-time reporting features. In conclusion, the Ration Card & Beneficiary Management Database successfully demonstrates OSDBMS concepts such as normalization, constraints, triggers, and transaction management.

CHAPTER 13 REFERENCES

1. MySQL Documentation
2. Database System Concepts by Silberschatz
3. Online learning resources related to SQL and MySQL.

CHAPTER 14 GLOSSARY

• **DBMS (Database Management System):**
Software used to store, manage, and retrieve data efficiently in the form of databases.

• **SQL (Structured Query Language):**
A standard language used to create, insert, update, delete, and retrieve data from a database.

• **Primary Key:**
A unique identifier for each record in a table that does not allow duplicate or NULL values.

• **Foreign Key:**
A field in a table that creates a relationship with the primary key of another table to maintain data consistency.

• **Transaction:**
A group of SQL operations executed together to ensure data consistency. It follows the principle of either complete success or complete failure.

• **MySQL:**
An open-source relational database management system used to store and manage structured data.

15. APPENDIX / SQL CODE

```
CREATE DATABASE RationCardDB1;

USE RationCardDB1;


CREATE TABLE Ration_Card (

    card_id INT PRIMARY KEY,

    card_type VARCHAR(20) CHECK (card_type IN ('APL','BPL','AAY')),

    issue_date DATE,

    status VARCHAR(10) DEFAULT 'ACTIVE'
```

```
);

CREATE TABLE Family_Members (
    member_id INT PRIMARY KEY,
    card_id INT,
    member_name VARCHAR(50),
    age INT,
    relation VARCHAR(30),
    FOREIGN KEY (card_id) REFERENCES Ration_Card(card_id)
);

CREATE TABLE Officer (
    officer_id INT PRIMARY KEY,
    officer_name VARCHAR(50),
    designation VARCHAR(30)
);

CREATE TABLE Stock (
    item_id INT PRIMARY KEY,
    item_name VARCHAR(30),
    quantity INT CHECK (quantity >= 0)
);
```

```
CREATE TABLE Distribution (
    dist_id INT PRIMARY KEY,
    card_id INT,
    item_id INT,
    quantity_given INT,
    dist_date DATE,
    officer_id INT,
    FOREIGN KEY (card_id) REFERENCES Ration_Card(card_id),
    FOREIGN KEY (item_id) REFERENCES Stock(item_id),
    FOREIGN KEY (officer_id) REFERENCES Officer(officer_id)
);

DELIMITER //

CREATE TRIGGER check_stock_before_distribution
BEFORE INSERT ON Distribution
FOR EACH ROW
BEGIN
    DECLARE available_stock INT;

    SELECT quantity INTO available_stock
    FROM Stock
    WHERE item_id = NEW.item_id;
```

```
    IF available_stock < NEW.quantity_given THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Not enough stock available';
    END IF;
END;
//

DELIMITER ;

INSERT INTO Ration_Card VALUES
(1001,'BPL','2023-01-10','ACTIVE'),
(1002,'APL','2022-05-12','ACTIVE'),
(1003,'AAY','2021-08-15','ACTIVE'),
(1004,'BPL','2023-03-21','ACTIVE'),
(1005,'APL','2022-07-11','ACTIVE'),
(1006,'BPL','2023-02-18','ACTIVE'),
(1007,'APL','2022-09-05','ACTIVE'),
(1008,'AAY','2021-11-22','ACTIVE'),
(1009,'BPL','2023-04-01','ACTIVE'),
(1010,'APL','2022-10-19','ACTIVE'),
(1011,'BPL','2023-06-25','ACTIVE'),
(1012,'AAY','2021-12-30','ACTIVE');
```

```
INSERT INTO Family_Members VALUES
(1,1001,'Zaara Shaikh',21,'Self'),
(2,1001,'Ahmed Shaikh',50,'Father'),
(3,1002,'Priya Sharma',30,'Self'),
(4,1003,'Ravi Patel',45,'Self'),
(5,1004,'Ayesha Khan',28,'Self'),
(6,1005,'Rahul Verma',35,'Self'),
(7,1006,'Fatima Shaikh',25,'Self'),
(8,1007,'Arjun Singh',40,'Self'),
(9,1008,'Meena Kumari',55,'Self'),
(10,1009,'Imran Khan',38,'Self');

INSERT INTO Officer VALUES
(201,'Mr. Khan','Food Inspector'),
(202,'Mr. Sharma','Supervisor'),
(203,'Ms. Patel','Clerk'),
(204,'Mr. Singh','Inspector'),
(205,'Ms. Verma','Officer'),
(206,'Mr. Gupta','Supervisor'),
(207,'Ms. Shaikh','Clerk'),
(208,'Mr. Khan','Inspector'),
(209,'Mr. Yadav','Officer'),
```

```
(210,'Ms. Sharma','Supervisor');

INSERT INTO Stock VALUES
(101,'Rice',500),
(102,'Wheat',400),
(103,'Sugar',300),
(104,'Oil',200),
(105,'Salt',150),
(106,'Dal',250),
(107,'Tea',180),
(108,'Milk Powder',120),
(109,'Maize',220);

START TRANSACTION;

UPDATE Stock
SET quantity = quantity - 5
WHERE item_id = 101;

INSERT INTO Distribution
VALUES (1, 1001, 101, 5, CURDATE(), 201);

COMMIT;
```