Database Management for Sustainable Development

SDG 2: Zero Hunger

A Final Project
presented to the faculty of the
School of Arts, Science and Technology
National Teachers College
Quiapo, Manila

In Partial Fulfillment of the Requirements
in Database Management
for the Degree of
Bachelor of Science in Information Technology

Submitted by:
Ariong, Ali Jr.
Bigcas, Paul Sander B.
Dela Cruz, Kurt Ahron D.
Gonzaga, Cedie L.
Virayo, Charo Marie K.

Submitted to:
Ms. Justin Louise R. Neypes

December 2025

INTRODUCTION

1.1 PROJECT OVERVIEW

This project was developed to support the goal of Zero Hunger by creating a simple but reliable database system for managing food donations and distributions. Many organizations that help address food insecurity rely on accurate records to ensure that donated food is properly stored and distributed to those who need it most. The system helps organize essential information such as food inventory, donor details, and distribution records, making it easier to track supplies and avoid shortages or waste.

The project uses a pure command-line interface (CLI) approach, where all database operations are performed directly through SQL commands in the RDBMS client. Instead of using a graphical interface or application, the focus is placed on the core database itself, its structure, logic and security. Separate SQL scripts are used to create tables, enforce rules, insert sample test data and control user access. This approach highlights how a well-designed database can effectively support real-world operations while remaining clear, testable, and easy to maintain.

1.2 PROBLEM STATEMENT

Many organizations that support food donation and distribution still rely on manual records or unorganized digital files to manage their data. This makes it difficult to accurately track food inventory, donor information, and distribution history. As a result, food items may expire, be misplaced, or fail to reach the people who need them most, leading to waste, shortages, and unequal distribution of resources. These issues hinder efforts to achieve Zero Hunger and reduce the effectiveness of food assistance programs.

In addition, the absence of a centralized and structured database system increases the risk of data duplication, inconsistency, and security breaches. Without clear rules and proper access controls, managing food donation records becomes inefficient and prone to errors. Therefore, there is a need for a reliable and well-organized database management system that can store, secure, and organize food-related data effectively. Such a system would support sustainable food distribution, improve operational efficiency, and ensure that donations reach those who need them most.

2.1 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

FUNCTIONAL REQUIREMENTS

ID	Requirement	Alignment
FR1	The system initializes with valid data for Donors, Inventory (33 items), and Distributions, secured by foreign keys.	DDL/DML
FR2	The system enforces integrity using CHECK constraints (positive stock) and a TRIGGER (Safety Check) to prevent expired item distribution	Core DBMS
FR3	The DistributeFood Stored procedure executes a critical transactions (update stock + log) using COMMIT and ROLLBACK to ensure ACID compliance	Final DBMS
FR4	The system generates required Zero Hunger reports (ex. View_ExpiringStock) to track food sustainability and donor impact.	DQL/Reporting

NON-FUNCTIONAL REQUIREMENTS

ID	Requirement	Metric
NFR1	The database must handle invalid input (such as negative stock or expired items) gracefully without crashing.	The system uses SIGNAL SQLSTATE to return informative errors like “ERROR: Insufficient stock” or “ERROR: EXPIRED ITEM” instead of generic crashes.
NFR2	The SQL script and stored logic must be modular, and	The SQL file is organized into distinct sections (DDL,

	follow standard coding practices	DML, Stored Logic) and includes inline comments (e.g., Log Transaction, Check 2: Expired?) to explain complex logic.
NFR3	The required SDG reports (VIEWS) must execute efficiently on standard hardware.	The View_ExpiringStock query uses index Primary Keys and simple conditional logic (DATEDIFF), ensuring report generation takes less than 1 second for the required 50+ records.

2.2 DATA REQUIREMENTS

The system requires structured input data stored in relational tables. The main data entities include donors, food inventory, donations, and distributions.

- **Donor Data:**
  - Includes donor ID, donor name, email, and join date. These fields are relatively small in size and stored as text and date data types.

- **Food Inventory Data:**
  - Includes item ID, item name, category, expiration date, and stock quality. This table is expected to grow as more items are donated and tracked over time.

- **Donation Records:**
  - Stores transaction data such as donation ID, donor ID, item ID, quantity donated, and donation date. This table can grow significantly and must support historical tracking.

- **Distribution Records:**
  - Includes distribution ID, item ID, beneficiary name, quantity distributed, and distribution date. This table supports monitoring outgoing food supplies.

2.3 SCHEMA NORMALIZATION ANALYSIS

The SDG2_ZeroHungerDB has been designed and normalized to Third Normal Form (3NF). This ensures the database is free from data redundancy, update anomalies, and insertion anomalies, which is essential for maintaining accurate inventory and distribution records. Below is the justification and functional dependency analysis for the primary entities in the system.

1. FoodInventory Table

Purpose:
Stores core stock details and serves as the central entity for tracking food expiration and availability.

Normalization Justification:
1NF: All attributes (item_name, category, expiration_date, stock_quantity) contain atomic values. There are no repeating groups.
2NF: The table uses a single-column primary key (item_id). Since there is no composite key, all non-key attributes depend fully on the primary key.
3NF: No transitive dependencies exist. Attributes such as expiration_date and stock_quantity depend directly on item_id and not on other non-key attributes like category.

Note on Category: Although category is dependent on item_name, item_id functions as a specific SKU or batch entry, serving as the sole determinant for the record.

Functional Dependencies (FDs):
item_id → item_name, category, expiration_date, stock_quantity

2. DonationLogs Table

Purpose:
Records the history of incoming supplies and links donors to the inventory.

Normalization Justification:
1NF: All values are atomic.
2NF: The primary key is log_id. Foreign keys (donor_id, item_id) are stored as references.
3NF: Non-key attributes (qty_donated, donation_date) are fully dependent on log_id.

Data Integrity: Donor information (e.g., email) or food details (e.g., expiration date) is not stored in this table but referenced via foreign keys. This prevents update anomalies.

Functional Dependencies (FDs):
log_id → donor_id, item_id, qty_donated, donation_date
donor_id → (References Donors table)
item_id → (References FoodInventory table)

3. Distributions Table
Purpose:
Tracks outgoing food to beneficiaries.

Normalization Justification:
3NF Compliance: beneficiary_name and quantity_given are directly dependent on dist_id (specific distribution event).

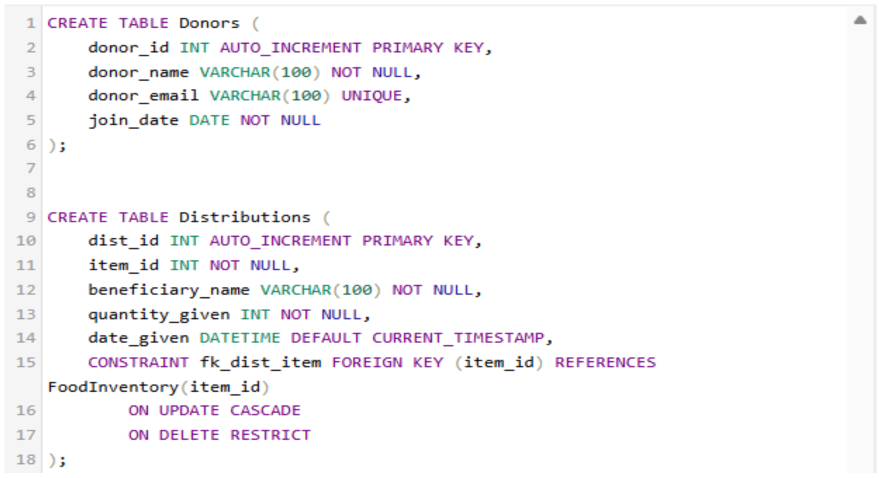
Transitive Dependency Check: If beneficiary_address were stored here, repeated distributions to the same beneficiary would cause redundancy. In this design, beneficiary_name is treated as a transaction attribute to maintain 3NF compliance.

Functional Dependencies (FDs):
dist_id → item_id, beneficiary_name, quantity_given, date_given

3.1 Core DBMS Concepts Used

1. **Tables and Relationships**
    - **Justification:**

- Tables are the foundation of any database system, they are how we organize and store information in a structured way. In this project, tables help us manage details about donors, food inventory, distributions, and donation logs. By keeping each type of data in its own table, the system remains organized and easy to maintain. Relationships between tables, established through foreign keys, connect different pieces of information, for example, a donor's details are linked to the items they have donated and each distributed item is tied back to the inventory. These connections ensure that the system can track donations and distributions accurately, preventing errors and making reporting easy.

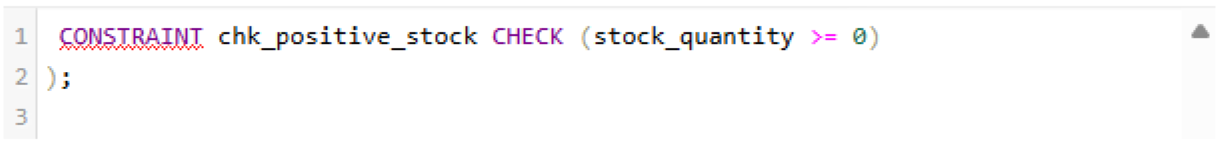


```
CREATE TABLE Donors (
    donor_id INT AUTO_INCREMENT PRIMARY KEY,
    donor_name VARCHAR(100) NOT NULL,
    donor_email VARCHAR(100) UNIQUE,
    join_date DATE NOT NULL
);


CREATE TABLE Distributions (
    dist_id INT AUTO_INCREMENT PRIMARY KEY,
    item_id INT NOT NULL,
    beneficiary_name VARCHAR(100) NOT NULL,
    quantity_given INT NOT NULL,
    date_given DATETIME DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_dist_item FOREIGN KEY (item_id) REFERENCES
FoodInventory(item_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
);
```

2. Constraints

- **Justification:**

  - Constraints act like rules in a game, they make sure that the data entered into the system is sensible and consistent. In our system, we use constraints to prevent mistakes, such as adding negative stock numbers or having duplicate donor emails. This keeps the database reliable and ensures that the food donation process runs smoothly.

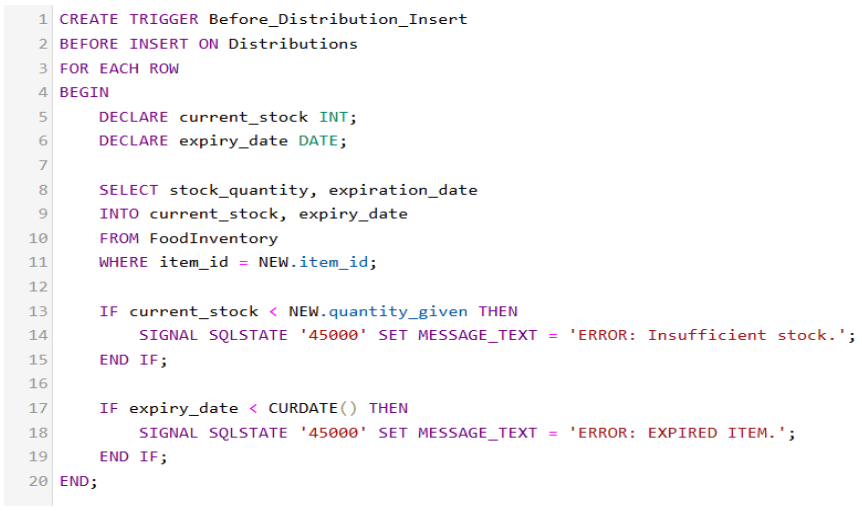


```sql
CONSTRAINT chk_positive_stock CHECK (stock_quantity >= 0)
);

```

3. Trigger

- **Justification:**

  - While constraints are great for simple rules, some situations require a more dynamic approach. That's where triggers come in, they automatically respond to certain events. For example, before distributing food, a trigger checks if there is enough stock and whether the item has expired. If the food is not available or has passed its expiration date, the system stops the transaction and shows an error. Triggers help enforce business rules in real-time, adding a layer of safety to operations.

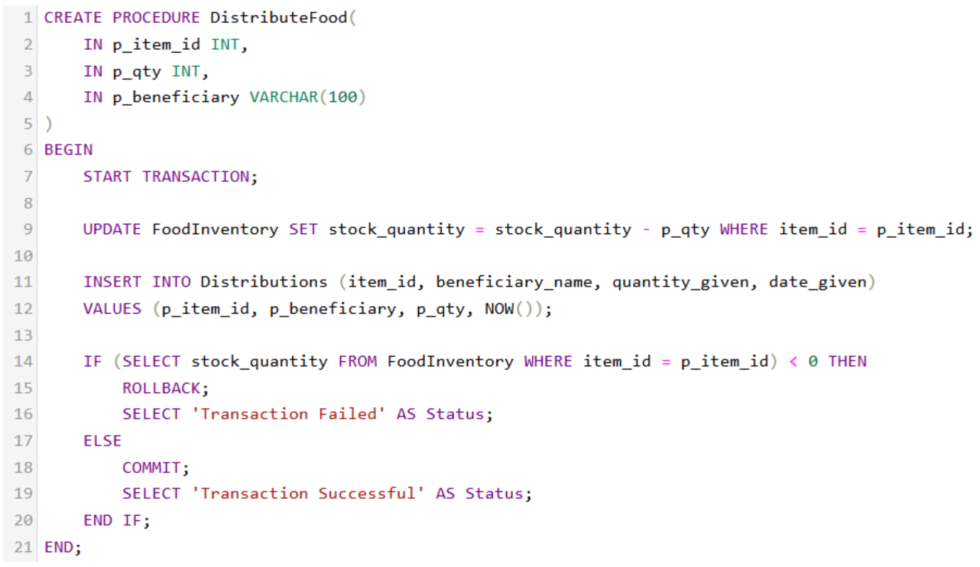


```sql
CREATE TRIGGER Before_Distribution_Insert
BEFORE INSERT ON Distributions
FOR EACH ROW
BEGIN
    DECLARE current_stock INT;
    DECLARE expiry_date DATE;

    SELECT stock_quantity, expiration_date
    INTO current_stock, expiry_date
    FROM FoodInventory
    WHERE item_id = NEW.item_id;

    IF current_stock < NEW.quantity_given THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'ERROR: Insufficient stock.';
    END IF;

    IF expiry_date < CURDATE() THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'ERROR: EXPIRED ITEM.';
    END IF;
END;
```

4. Stored Procedures

- **Justification:**

  - Stored procedures are like pre-packaged instructions that the system can follow whenever needed. So for our system, distributing food requires multiple steps such as reducing inventory, recording the distribution, and checking that stock levels don't go negative. By wrapping these actions in a stored procedure, we can perform them safely in one go. This ensures that the database remains accurate and consistent, even if something goes wrong during the process.

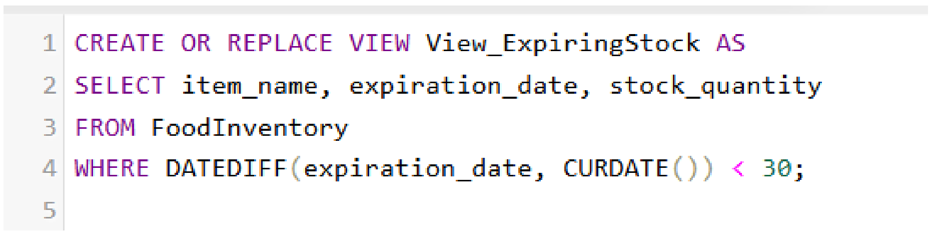


```sql
CREATE PROCEDURE DistributeFood(
    IN p_item_id INT,
    IN p_qty INT,
    IN p_beneficiary VARCHAR(100)
)
BEGIN
    START TRANSACTION;

    UPDATE FoodInventory SET stock_quantity = stock_quantity - p_qty WHERE item_id = p_item_id;

    INSERT INTO Distributions (item_id, beneficiary_name, quantity_given, date_given)
    VALUES (p_item_id, p_beneficiary, p_qty, NOW());

    IF (SELECT stock_quantity FROM FoodInventory WHERE item_id = p_item_id) < 0 THEN
        ROLLBACK;
        SELECT 'Transaction Failed' AS Status;
    ELSE
        COMMIT;
        SELECT 'Transaction Successful' AS Status;
    END IF;
END;
```

5. Views

- **Justification:**

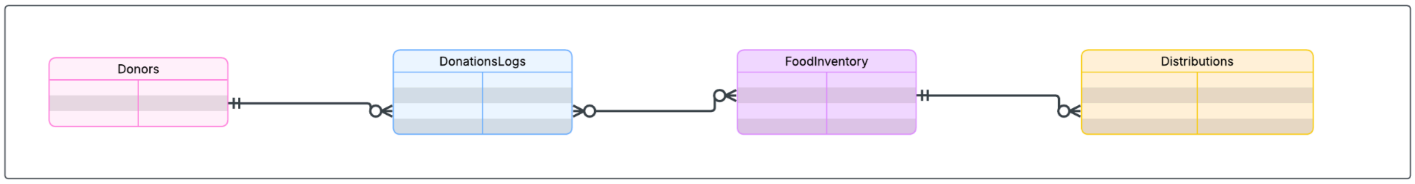
  - The views in our system show which items are about to expire. This allows staff to prioritize distributions and minimize food waste. By simplifying access to important data, views make the system easier to use and help staff make informed decisions without sifting through all the tables.



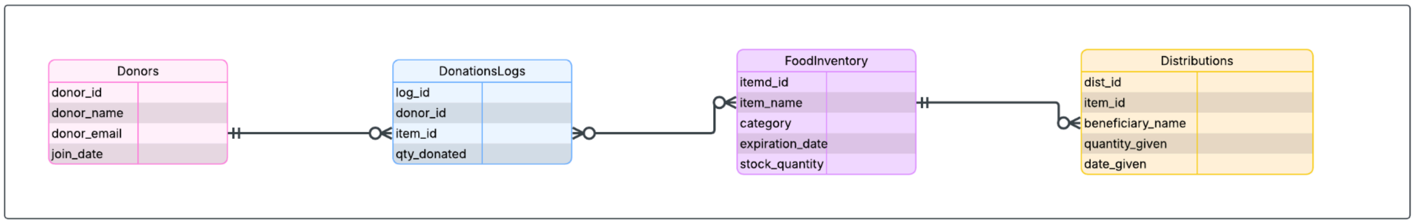
```sql
CREATE OR REPLACE VIEW View_ExpiringStock AS
SELECT item_name, expiration_date, stock_quantity
FROM FoodInventory
WHERE DATEDIFF(expiration_date, CURDATE()) < 30;

```

3.2 ER Diagram

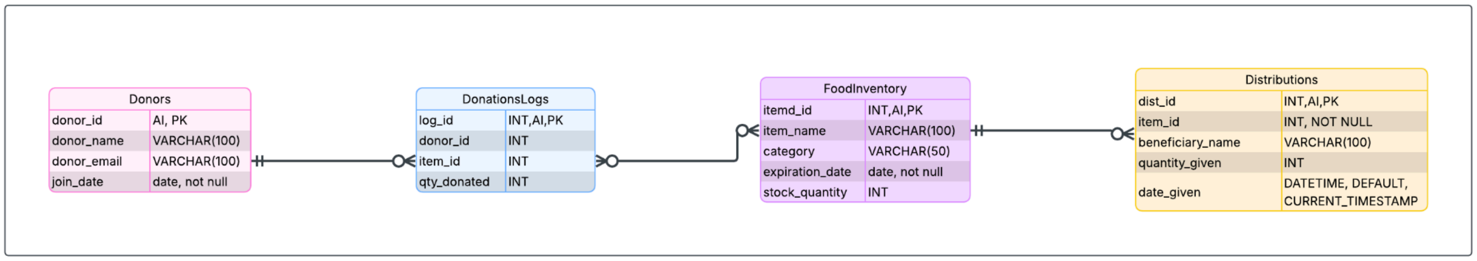
Conceptual



Logical



Physical



1. Conceptual Diagram

The **Conceptual Diagram** provides a high-level view of the system without technical details. It identifies the four main entities: Donors, DonationLogs, FoodInventory, and Distributions.

2. Logical Diagram

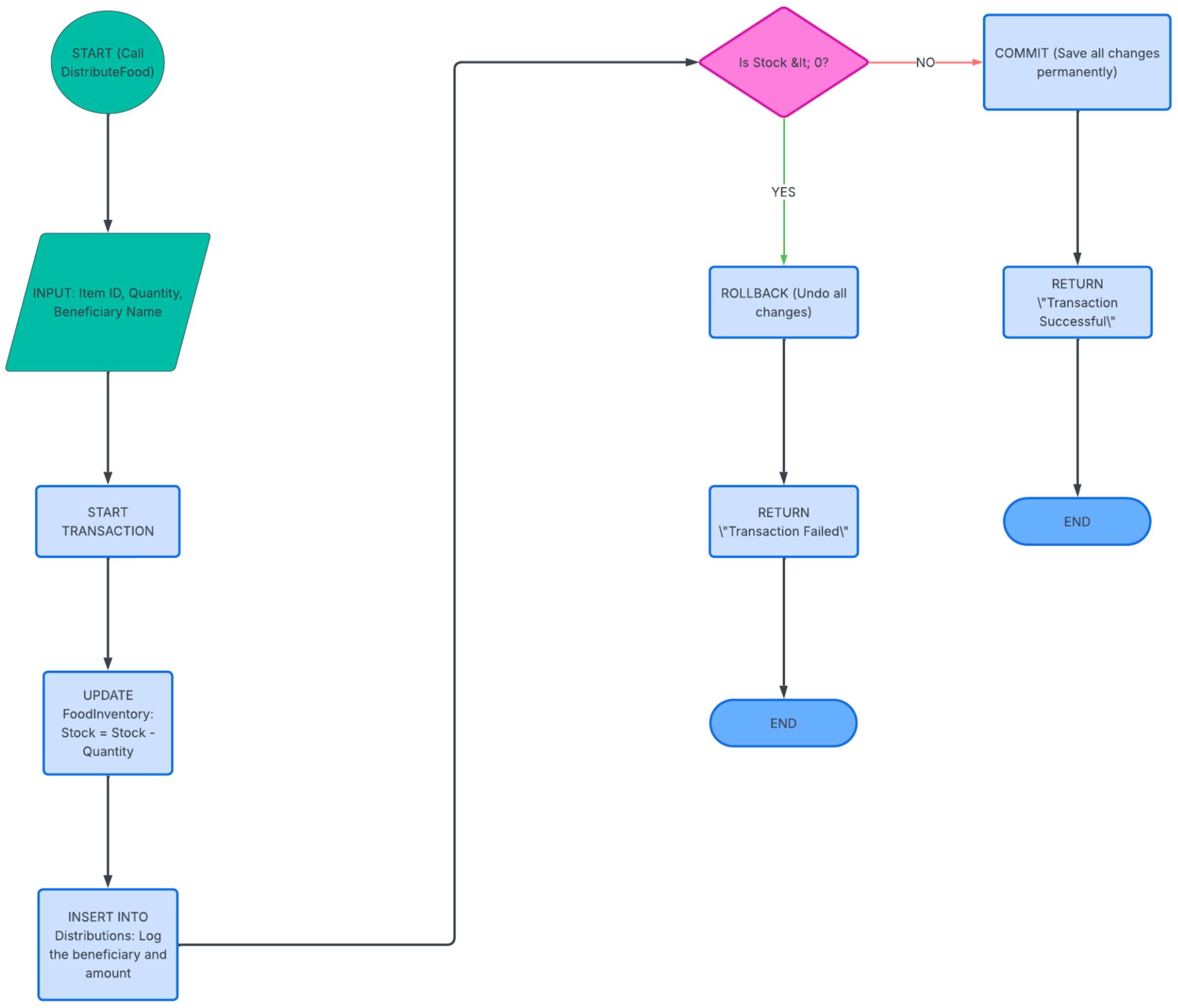
The **Logical Diagram** organizes our data into four main groups: Donors, DonationLogs, FoodInventory, and Distributions.

The Donors group contains attributes like the name and email so we know who gave the food. The FoodInventory group shows attributes like the stock quantity and expiration date to track the food status . Finally, the DonationLogs and Distributions groups include attributes like the quantity and date to record every transaction.

3. Physical Diagram

The **Physical Diagram** represents the actual database schema used for implementation. It specifies the technical details for every column, including Data Types (e.g., VARCHAR(100), INT, DATETIME) and Constraints (e.g., NOT NULL, AUTO_INCREMENT).

3.3 Food Distribution Transaction Flowchart


START (Call DistributeFood)
INPUT: Item ID, Quantity, Beneficiary Name
START TRANSACTION
UPDATE FoodInventory: Stock = Stock - Quantity
INSERT INTO Distributions: Log the beneficiary and amount
Is Stock &lt; 0?
NO
COMMIT (Save all changes permanently)
YES
ROLLBACK (Undo all changes)
RETURN \"Transaction Failed\"
END
RETURN \"Transaction Successful\"
END

Our flowchart explains the step-by-step process of safely distributing food using a database transaction.

The process begins by starting a transaction, which groups all the actions together so they happen at the same time. First, the system tries to subtract the quantity from the inventory and records the distribution details . After these steps, the system checks if the remaining stock is less than zero. If the stock number is negative, it means there is an error, so the system uses the Rollback command to undo all changes and cancel the transaction. If the stock number is correct, the system uses the Commit command to save the changes permanently and confirms that the transaction was successful.

CONCLUSION

This project demonstrates how a well-designed database system can make a tangible impact on food donation and distribution efforts. By organizing donor information, tracking food inventory, and recording distribution activities, the system provides a clear and reliable way to manage resources efficiently. The use of constraints, triggers, stored procedures, and views ensures data accuracy, prevents errors, and supports real-time decision-making, helping organizations reduce waste and reach more people in need.

Through a structured approach using a command-line interface, the project emphasizes the importance of database integrity, security, and maintainability. It shows that even without complex graphical interfaces, a robust relational database can significantly improve operational efficiency. Ultimately, this system contributes to the larger goal of Zero Hunger by enabling better tracking, planning, and reporting of food donations, ensuring that every contribution can make the maximum possible difference.