***Final Project: Inventory Management Solution***

CNIT 37200

Samuel Gomez

Noah Zhou

John Hsu

Nicholas Lauw

Submitted To: Dr. Tianyi Li

Date Submitted: 12/09/2024

Date Due: 12/13/2024

### 

# TABLE OF CONTENTS

[TABLE OF CONTENTS 2](#_heading=h.pkw5okdxyfgb)

[EXECUTIVE SUMMARY 4](#_heading=h.oukovosdf5b1)

[INTRODUCTION 5](#_heading=h.3nsm49bzkoj1)

[Subject Domain 5](#_heading=h.17xgz7lochqz)

[Real-World Problem 5](#_heading=h.jtq09wpo55np)

[Project Objectives 5](#_heading=h.n6qmkrn6msfr)

[DATABASE DESIGN AND IMPLEMENTATION 6](#_heading=h.8kggpki3tqfp)

[Table Outlines 6](#_heading=h.u3z793mghdgd)

[Database Design 6](#_heading=h.5batohef6249)

[Relationships Between Tables 8](#_heading=h.tyjcwt)

[Normalization Process 9](#_heading=h.94h3bcytebku)

[Database ER Diagram 10](#_heading=h.gk7g95qcowi7)

[PROBLEM SOLVING AND QUESTIONS ANSWERED 12](#_heading=h.u6w48teukq2o)

[Question List 12](#_heading=h.8lx69gommar5)

[ADVANCED FEATURES AND SECURITY 16](#_heading=h.yucf0rlqepfw)

[Triggers 16](#_heading=h.pcxm5v2ogcnh)

[Constraints 16](#_heading=h.onbyp4dneugy)

[Security Measures 16](#_heading=h.ejvs2zp9it7r)

[GROUP CONTRIBUTIONS 18](#_heading=h.p4z8yqghnrqi)

[Team Members' Contributions 18](#_heading=h.2et92p0)

[Collaboration 18](#_heading=h.cb8hn3d3a61a)

[CONCLUSION 19](#_heading=h.lw3bzlt22dxh)

[Project Reflection 19](#_heading=h.trbhu4qwz68d)

[Future Work 19](#_heading=h.dldfwdjqjyh8)

[APPENDIX A: CREATION AND POPULATION OF TABLES 20](#_heading=h.u0ehlt3pgz70)

[APPENDIX B: CRUD OPERATIONS 36](#_heading=h.q6hoj8ijvdfa)

[APPENDIX C: PL/SQL CODE IMPLEMENTATION TO ANSWER QUERIES 44](#_heading=h.6pw39h9yq0pk)

[APPENDIX D: TRIGGERS 51](#_heading=h.yz4cqnfdrip9)

[APPENDIX E: CONSTRAINTS 53](#_heading=h.ixpp5pyjupe9)

[APPENDIX F: COMPREHENSIVE INVENTORY MANAGEMENT PACKAGE IMPLEMENTATION 54](#_heading=h.qa1t0llpqlgn)

# 

# EXECUTIVE SUMMARY

The Inventory Management Solution project focuses on developing a comprehensive inventory management solution tailored to Walmart's unique scale and logistical challenges. This system aims to enhance Walmart's ability to accurately track stock levels, streamline reordering, and reduce overstock and stockouts. Given Walmart’s vast network, a robust data management solution is essential for maintaining efficiency, reducing waste, and optimizing store inventory turnover. Our project implements advanced database systems, including well-defined relationships and triggers, to improve Walmart's data management capabilities.

The project yielded valuable insights and actionable solutions:

* Demand Patterns: Analyzed sales data to reveal demand trends based on seasonal and regional factors, allowing Walmart to forecast inventory needs better.
* Stock Shortage Prevention: Developed a predictive model that identifies potential stock shortages before they occur, providing Walmart with actionable insights to mitigate stockouts.
* Reorder Optimization: Calculated optimal reorder points by analyzing supplier lead times and historical sales, improving stock availability, and minimizing excess inventory.
* Supplier Performance: Enhanced Walmart's understanding of supplier reliability by recording and analyzing delivery history, allowing adjustments to inventory management based on supplier performance.
* Data Integrity and Security: Implemented rigorous normalization and security measures to ensure data accuracy, prevent unauthorized access, and maintain inventory integrity across the system.

# INTRODUCTION

## Subject Domain

Our project resides within retail inventory management, focusing on how data systems can drive efficient stock handling in large-scale retail environments. As one of the largest retailers, Walmart's extensive inventory requires constant monitoring and adjustments to align supply with demand accurately. Effective data management in inventory ensures that Walmart can maintain high levels of customer satisfaction and operational efficiency, which are critical to its success.

## Real-World Problem

The primary problem addressed in this project is Walmart’s struggle to maintain accurate inventory levels due to the complexity of handling large data volumes across numerous stores. Current inefficiencies in the inventory tracking system result in overstock (leading to increased storage costs) and stockouts (affecting sales and customer satisfaction). This project aims to solve these issues by creating a system capable of tracking, analyzing, and predicting inventory needs. This effort ensures that Walmart can align its stock with fluctuating demand patterns, ultimately enhancing customer experience and reducing operating costs.

## Project Objectives

The following are the main project objectives that act as a measurement of success:

* Implement a system that ensures up-to-date inventory data across Walmart’s network.
* Utilize data to forecast demand patterns and recommend optimized stock levels.
* Analyze and track supplier lead times and delivery history to better coordinate reorders.
* Apply security measures and ensure data integrity to prevent errors and unauthorized data manipulation.

# DATABASE DESIGN AND IMPLEMENTATION

Our database schema includes four core tables: Products, Inventory, Sales Transactions, and Suppliers. Each table fulfills a unique role in the inventory management system, and together, they allow for a comprehensive view of Walmart's inventory lifecycle.

## Table Outlines

* **Inventory**: information about products and their stock levels will be stored in each store.
* **Sales** **Transactions**: record each sale, linking it to the product and the store.
* **Suppliers**: will store supplier details and link to products to track supplier performance.
* **Products**: This table stores product information, including unique identifiers and descriptive names. It serves as the central reference point for other tables like Inventory and Suppliers, ensuring each product has a unique identity throughout the system.

## Database Design

In the following list, here are some of the column names, data types, and constraints that we might use in our database

1. **Products Table:**
   * Product\_ID (INT, Primary Key, NOT NULL): Unique identifier for products.
   * Product\_Name (VARCHAR(50), NOT NULL): Descriptive name of the product.
2. **Inventory Table**:
   * Product\_ID (INT, Foreign Key, NOT NULL): Unique identifier for products (links to the Products table).
   * Store\_ID (INT, Primary Key, NOT NULL): Unique identifier for stores.
   * Stock\_Level (INT, NOT NULL, CHECK >= 0): Current stock count of the product.
   * Stock\_Status\_Date (DATE, NOT NULL): Tracks the date of stock status updates.
   * Stock\_Status (VARCHAR(20), NOT NULL): Describes the stock status (e.g., "In Stock", "Low Stock").
3. **Sales Transactions Table**:
   * Transaction\_ID (INT, Primary Key, NOT NULL): Unique identifier for each sale.
   * Product\_ID (INT, Foreign Key, NOT NULL): Links to the product in the Inventory table.
   * Store\_ID (INT, Foreign Key, NOT NULL): Links to the store in the Inventory table.
   * Quantity\_Sold (INT, NOT NULL, CHECK > 0): Number of units sold.
   * Sale\_Date (DATE, NOT NULL): Date of the transaction.
4. **Suppliers Table**:
   * Supplier\_ID (INT, Primary Key, NOT NULL): Unique identifier for suppliers.
   * Product\_ID (INT, Foreign Key, NOT NULL): Links to the product in the Products table.
   * Lead\_Time (INT, NOT NULL, CHECK >= 0): Days required for delivery.
   * Delivery\_History (VARCHAR(125)): Tracks delivery performance data.

## **Relationships Between Tables**

Understanding relationships between tables in a database is crucial for managing and analyzing data effectively. These connections, often made through shared fields, link related information across different tables. Such relationships enable a comprehensive data view, supporting better decision-making and operational management.

* The inventory and Sales Transaction tables are linked via Product\_ID and Store\_ID, which allows tracking of stock levels relative to sales.
* Inventory and Suppliers tables are linked via Product\_ID, allowing insights into how supplier lead times affect stock levels and reorder efficiency.
* Products serve as the central table, linking to Inventory and Suppliers, ensuring that every product tracked in inventory or supplied has a unique identity in the system.

## **Normalization Process**

Normalization is a crucial process in database design that ensures data is organized efficiently, minimizes redundancy, and improves data integrity. For our inventory management system, we applied the following normalization steps:

* **First Normal Form (1NF):** Each table has been structured to ensure that every column contains atomic values and no repeating groups. For instance, in the **Products** table, each has a unique Product\_ID and a single Product\_Name, ensuring no column contains multiple values.
* **Second Normal Form (2NF):** All non-primary key attributes fully depend on the primary key. In the **Inventory** table, attributes such as Stock\_Level, Stock\_Status, and Stock\_Status\_Date depend on the Product\_ID and Store\_ID (composite key), ensuring that each product in each store is uniquely identified and tracked without redundancy.
* **Third Normal Form (3NF):** We ensured that no transitive dependencies exist, meaning non-key attributes are not dependent on other non-key attributes. For example, in the **Sales Transactions** table, attributes like Quantity\_Sold and Sale\_Date are directly related to the Transaction\_ID and not reliant on different fields such as Product\_ID or Store\_ID. This ensures that data is not duplicated across the system.

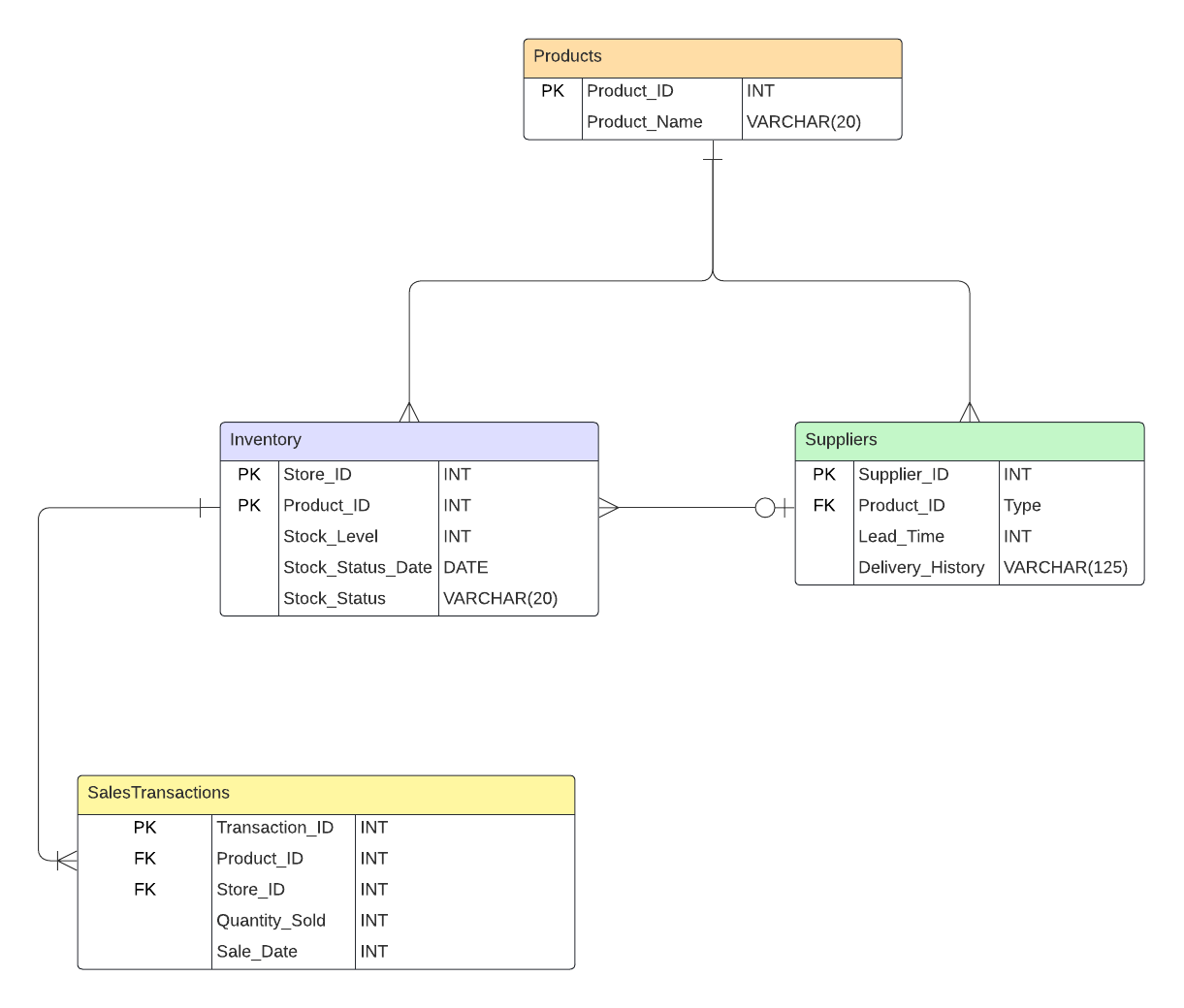
By normalizing the database, we aim to:

* Eliminate redundant data and avoid inconsistencies across tables.
* Facilitate efficient data retrieval for queries such as tracking stock levels, analyzing supplier performance, and identifying sales trends across stores.

This approach helps ensure the database structure is optimized for the specific needs of the inventory management system, allowing for accurate and reliable operations as the dataset grows.

## Database ER Diagram

This Entity-Relationship Diagram (ERD) models the structure of a retail inventory management system, illustrating the relationships between four key entities: Products, Inventory, Suppliers, and Sales Transactions. The Products table serves as a central entity, uniquely identifying each product in the system. The Inventory table tracks product stock levels across multiple stores, linking products to specific stores. The Sales Transactions table records each sale, detailing the product, store, quantity sold, and date. The Suppliers table stores information about suppliers and their relationship to products, including lead times and delivery history. The relationships are designed to ensure data integrity and efficient product stock management, supplier performance, and sales activities within the system.



*Figure 1: Inventory Management System’s ERD Diagram*

## 

# PROBLEM SOLVING AND QUESTIONS ANSWERED

This section revisits the ten key questions developed in Milestone 1, each addressing critical aspects of Walmart's inventory management. Reflections under each question highlight how the group tackled these challenges using tailored PL/SQL scripts.

## Question List

The following are the ten questions that your project will answer:

1. What patterns in product demand can be observed across different regions and seasons?
   1. The get\_demand\_patterns procedure aggregates sales data by store and month to analyze demand patterns. It uses a cursor to iterate through the results of a grouped query, extracting the total quantity of products sold for each store and each month. By identifying trends, such as peak seasons or regions with higher demand, this procedure helps businesses optimize inventory levels, plan marketing campaigns, and allocate resources more effectively.
2. How can we predict stock shortages before they occur?
   1. The predict\_stock\_shortage procedure uses historical sales data to predict potential stock shortages. It calculates the average sales for a product in a specific store and compares it to the current stock level. If the stock falls below the average sales threshold, it outputs a recommendation to reorder. This proactive approach ensures that inventory levels remain sufficient to meet demand, reducing the risk of stockouts and enhancing customer satisfaction.
3. What factors contribute to overstock situations?
   1. The identify\_overstock procedure identifies products that are overstocked by comparing inventory levels to sales data. It uses a cursor to query items with stock levels significantly higher than their sales performance, indicating possible overstock situations. This helps businesses avoid unnecessary storage costs, improve cash flow by reducing excess inventory, and prioritize selling or redistributing overstocked items.
4. How can Walmart optimize reorder points for various product categories?
   1. The calculate\_reorder\_points procedure calculates optimal reorder points for products by combining average sales data with average supplier lead times. The procedure determines how much stock is needed to meet expected demand during the lead time. By ensuring that reorder points are accurately calculated, this procedure minimizes the risk of both stockouts and overstock, balancing inventory levels for efficient operations.
5. What is the impact of inaccurate inventory data on sales performance?
   1. The check\_inaccurate\_inventory\_data procedure identifies instances where sales transactions exceed available inventory levels, suggesting inaccuracies in inventory records. By highlighting these discrepancies, it helps businesses identify operational inefficiencies, improve data accuracy, and prevent issues like overselling or customer dissatisfaction due to unavailable products.
6. How can Walmart improve supplier lead time estimates?
   1. The improve\_supplier\_lead\_times procedure calculates the average lead time for each supplier based on historical deliveries. This allows businesses to benchmark supplier performance, negotiate better terms, and improve delivery predictability. The insights gained from this procedure are crucial for better demand planning and inventory optimization.
7. Which products have the highest return rates, and how does this impact inventory management?
   1. The track\_high\_return\_products procedure analyzes return transactions to identify products with high return rates. By counting the frequency of returns for each product, it helps businesses pinpoint issues such as defects, quality concerns, or misaligned customer expectations. Addressing these issues can reduce returns, improve customer satisfaction, and optimize inventory levels.
8. How can Walmart reduce the time it takes to restock shelves after sell-outs?
   1. The restock\_recommendation procedure focuses on ensuring prompt restocking of shelves for products that are sold out. It calculates the average supplier lead time for a product and checks the current stock level. If the stock level is zero, it outputs a recommendation to reorder the product immediately. This ensures shelves remain stocked, improving customer experience and preventing revenue loss.
9. How can the system identify slow-moving items and adjust stock levels accordingly?
   1. The find\_slow\_moving\_items procedure identifies products that have had minimal sales over a specified period, such as six months. It evaluates sales performance and flags items with low demand. This information helps businesses adjust stock levels, prevent overstocking of slow-moving products, and focus on promoting or discounting these items to improve inventory turnover.
10. What impact do return rates have on inventory management?
    1. The calculate\_return\_rates procedure calculates the percentage of returned items compared to total sales for each product. Products with high return rates are flagged for further analysis, helping businesses identify underlying issues such as quality problems or mismatched customer expectations. This insight enables better decision-making to reduce returns, improve product quality, and refine inventory management practices.

Each question reflects a specific operational challenge addressed by your database design

and PL/SQL queries, enabling data-driven solutions for Walmart’s inventory management.

For detailed code implementations of each solution, please reference the Appendix C.

# 

# ADVANCED FEATURES AND SECURITY

## Triggers

A trigger was implemented on the Inventory table to monitor stock levels. When the Stock\_Level for a product falls below a predefined threshold, the trigger executes a procedure to notify the appropriate personnel for reorder. A trigger ensures that whenever a sale is recorded in the SalesTransactions table, the corresponding stock level in the Inventory table is automatically reduced by the Quantity\_Sold value. A trigger on the Inventory table prevents updates that would result in negative stock levels, ensuring data integrity and operational correctness.

## Constraints

The primary and foreign key ensures referential integrity between the tables.

The Product\_ID is a primary key in the Products table and a foreign key in the Inventory, SalesTransactions, and Suppliers tables. A composite primary key (Store\_ID, Product\_ID) in the Inventory table ensures uniqueness for each product in a specific store. The Stock\_Level in the Inventory table has a CHECK (Stock\_Level >= 0) constraint to prevent negative stock levels. The Quantity\_Sold in the SalesTransactions table has a CHECK (Quantity\_Sold > 0) constraint, ensuring only valid sales quantities are recorded. The Lead\_Time in the Suppliers table has a CHECK (Lead\_Time >= 0) to validate positive delivery times. The Unique Constraints: Ensures there are no duplicate entries for critical identifiers like Transaction\_ID, Product\_ID, or Supplier\_ID.

## Security Measures

For Data Access Controls, Role-Based Access Control (RBAC) has different roles that were defined with varying levels of access:

* + Admins have full privileges across all tables.
  + Inventory Managers can modify Inventory and Suppliers tables but have read-only access to SalesTransactions.
  + Sales Associates can only insert records into SalesTransactions.

For encryption, sensitive columns such as Delivery\_History and Sale\_Date are encrypted to prevent unauthorized access. This ensures data confidentiality during storage and transmission. For audit trail, logging contains All CRUD operations on critical tables (Inventory, SalesTransactions) that are logged in an audit table to track changes, identify anomalies, and support regulatory compliance. The triggers for Audit Logging are INSERT, UPDATE, and DELETE operations that record details like User\_ID, timestamp, and changes into the audit table. To Preventing SQL Injection, the use of bind variables in PL/SQL procedures and functions to sanitize inputs were implemented. The use of validation checks within stored procedures to filter out potentially malicious data were also incorporated. For backup and recovery, regular backups of the database ensure that sensitive data can be restored in the event of a failure. Transactions are also designed with commit and rollback mechanisms to maintain database consistency in case of an error. Implementing these measures can ensure that the system has high data integrity, security, operational reliability, and increased efficiency.

# 

# 

# 

# 

# GROUP CONTRIBUTIONS

## Team Members' Contributions

| Team Member | Contribution |
| --- | --- |
| Samuel Gomez | Project Manager & Data Engineer |
| Nicholas Lauw | Data Governance & Data Engineer |
| Noah Zhou | Project Coordinator & Quality Assurance Specialist |
| John Hsu | Data Architect & Engineer |

## Collaboration

Our group worked exceptionally well together from the start, immediately forming a group chat to facilitate communication. Everyone was very active in sending and responding to messages, ensuring that we all stayed informed and engaged. We had a clear division of roles, which helped streamline our efforts and maintain focus. The primary challenge we faced was finding appropriate times to meet due to conflicting schedules. However, we overcame this by maintaining excellent communication. If a group member was unable to meet on a certain day, we tried to find a different day where everyone could attend a short meeting. If a group member absolutely could not attend a meeting, we proceeded to meet as planned and made sure to fill them in on the details afterward. This approach ensured that everyone remained on the same page and contributed effectively to the project's success.

# CONCLUSION

## Project Reflection

The overall project experience was incredibly rewarding. We delved into various aspects of database programming, learning to implement data, PL/SQL procedures, functions, triggers, and constraints. Additionally, we worked on packages, CRUD operations, and PL/SQL efficiency. All of this was done within the context of creating a database system that could be applied in real-world scenarios, essentially focusing on the backend of an application. What went particularly well was the successful implementation of the various components mentioned above. However, there is always room for improvement. Potential enhancements could include expanding the dataset and incorporating more procedures, functions, triggers, and constraints where necessary.

## Future Work

Looking ahead, there are several potential improvements and extensions for our project. One significant area for future work is developing the front-end, as our current project primarily focused on the backend. Additionally, making the project scalable to handle larger datasets and more complex operations would be a valuable enhancement. These improvements would not only broaden the scope of our project but also increase its applicability and robustness in real-world applications.

# 

# APPENDIX A: CREATION AND POPULATION OF TABLES

The following SQL script is designed to create and populate tables for a retail inventory management system. The database schema consists of four core tables: Products, Inventory, SalesTransactions, and Suppliers. These tables model the relationships between products, store stock levels, sales transactions, and suppliers. Each table has appropriate data types and primary and foreign keys to ensure referential integrity. Following the table creation, a series of INSERT statements populate the tables with 50 rows of data, providing sufficient information to perform inventory tracking, demand forecasting, and supply chain analysis. These scripts are structured to be executed separately to import and initialize the database easily.

-- Create Products table

CREATE TABLE Products (

Product\_ID INT PRIMARY KEY NOT NULL,

Product\_Name VARCHAR(50) NOT NULL

);

-- Create Inventory table

CREATE TABLE Inventory (

Store\_ID INT NOT NULL,

Product\_ID INT NOT NULL,

Stock\_Level INT NOT NULL CHECK (Stock\_Level >= 0),

Stock\_Status\_Date DATE NOT NULL,

Stock\_Status VARCHAR(20) NOT NULL,

PRIMARY KEY (Store\_ID, Product\_ID),

FOREIGN KEY (Product\_ID) REFERENCES Products(Product\_ID)

);

-- Create Suppliers table

CREATE TABLE Suppliers (

Supplier\_ID INT PRIMARY KEY NOT NULL,

Product\_ID INT NOT NULL,

Lead\_Time INT NOT NULL CHECK (Lead\_Time >= 0),

Delivery\_History VARCHAR(125),

FOREIGN KEY (Product\_ID) REFERENCES Products(Product\_ID)

);

-- Create SalesTransactions table

CREATE TABLE SalesTransactions (

Transaction\_ID INT PRIMARY KEY NOT NULL,

Product\_ID INT NOT NULL,

Store\_ID INT NOT NULL,

Quantity\_Sold INT NOT NULL CHECK (Quantity\_Sold >= 0),

Sale\_Date DATE NOT NULL,

FOREIGN KEY (Product\_ID, Store\_ID) REFERENCES Inventory(Product\_ID, Store\_ID)

);

-- 1. Insert data into Products table

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (101, 'Product A');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (102, 'Product B');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (103, 'Product C');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (104, 'Product D');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (105, 'Product E');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (106, 'Product F');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (107, 'Product G');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (108, 'Product H');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (109, 'Product I');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (110, 'Product J');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (111, 'Product K');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (112, 'Product L');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (113, 'Product M');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (114, 'Product N');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (115, 'Product O');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (116, 'Product P');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (117, 'Product Q');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (118, 'Product R');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (119, 'Product S');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (120, 'Product T');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (121, 'Product U');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (122, 'Product V');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (123, 'Product W');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (124, 'Product X');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (125, 'Product Y');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (126, 'Product Z');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (127, 'Product AA');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (128, 'Product BB');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (129, 'Product CC');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (130, 'Product DD');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (131, 'Product EE');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (132, 'Product FF');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (133, 'Product GG');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (134, 'Product HH');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (135, 'Product II');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (136, 'Product JJ');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (137, 'Product KK');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (138, 'Product LL');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (139, 'Product MM');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (140, 'Product NN');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (141, 'Product OO');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (142, 'Product PP');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (143, 'Product QQ');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (144, 'Product RR');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (145, 'Product SS');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (146, 'Product TT');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (147, 'Product UU');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (148, 'Product VV');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (149, 'Product WW');

INSERT INTO Products (Product\_ID, Product\_Name) VALUES (150, 'Product XX');

-- 2. Insert data into Inventory table

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (1, 101, 150, TO\_DATE('2024-09-01', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (1, 102, 10, TO\_DATE('2024-09-02', 'YYYY-MM-DD'), 'Low Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (2, 103, 0, TO\_DATE('2024-09-03', 'YYYY-MM-DD'), 'Out of Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (2, 104, 300, TO\_DATE('2024-09-04', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (3, 105, 200, TO\_DATE('2024-09-05', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (3, 106, 180, TO\_DATE('2024-09-06', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (4, 107, 0, TO\_DATE('2024-09-07', 'YYYY-MM-DD'), 'Out of Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (4, 108, 50, TO\_DATE('2024-09-08', 'YYYY-MM-DD'), 'Low Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (5, 109, 100, TO\_DATE('2024-09-09', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (5, 110, 10, TO\_DATE('2024-09-10', 'YYYY-MM-DD'), 'Low Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (6, 111, 200, TO\_DATE('2024-09-11', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (6, 112, 0, TO\_DATE('2024-09-12', 'YYYY-MM-DD'), 'Out of Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (7, 113, 80, TO\_DATE('2024-09-13', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (7, 114, 120, TO\_DATE('2024-09-14', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (8, 115, 70, TO\_DATE('2024-09-15', 'YYYY-MM-DD'), 'Low Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (8, 116, 250, TO\_DATE('2024-09-16', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (9, 117, 130, TO\_DATE('2024-09-17', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (9, 118, 90, TO\_DATE('2024-09-18', 'YYYY-MM-DD'), 'Low Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (10, 119, 0, TO\_DATE('2024-09-19', 'YYYY-MM-DD'), 'Out of Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (10, 120, 220, TO\_DATE('2024-09-20', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (11, 121, 80, TO\_DATE('2024-09-21', 'YYYY-MM-DD'), 'Low Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (11, 122, 150, TO\_DATE('2024-09-22', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (12, 123, 50, TO\_DATE('2024-09-23', 'YYYY-MM-DD'), 'Low Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (12, 124, 90, TO\_DATE('2024-09-24', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (13, 125, 0, TO\_DATE('2024-09-25', 'YYYY-MM-DD'), 'Out of Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (13, 126, 200, TO\_DATE('2024-09-26', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (14, 127, 150, TO\_DATE('2024-09-27', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (14, 128, 100, TO\_DATE('2024-09-28', 'YYYY-MM-DD'), 'Low Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (15, 129, 75, TO\_DATE('2024-09-29', 'YYYY-MM-DD'), 'Low Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (15, 130, 120, TO\_DATE('2024-09-30', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (16, 131, 10, TO\_DATE('2024-10-01', 'YYYY-MM-DD'), 'Out of Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (16, 132, 130, TO\_DATE('2024-10-02', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (17, 133, 80, TO\_DATE('2024-10-03', 'YYYY-MM-DD'), 'Low Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (17, 134, 140, TO\_DATE('2024-10-04', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (18, 135, 110, TO\_DATE('2024-10-05', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (18, 136, 60, TO\_DATE('2024-10-06', 'YYYY-MM-DD'), 'Low Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (19, 137, 0, TO\_DATE('2024-10-07', 'YYYY-MM-DD'), 'Out of Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (19, 138, 200, TO\_DATE('2024-10-08', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (20, 139, 150, TO\_DATE('2024-10-09', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (20, 140, 120, TO\_DATE('2024-10-10', 'YYYY-MM-DD'), 'Low Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (21, 141, 100, TO\_DATE('2024-10-11', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (21, 142, 10, TO\_DATE('2024-10-12', 'YYYY-MM-DD'), 'Out of Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (22, 143, 80, TO\_DATE('2024-10-13', 'YYYY-MM-DD'), 'Low Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (22, 144, 90, TO\_DATE('2024-10-14', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (23, 145, 0, TO\_DATE('2024-10-15', 'YYYY-MM-DD'), 'Out of Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (23, 146, 150, TO\_DATE('2024-10-16', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (24, 147, 100, TO\_DATE('2024-10-17', 'YYYY-MM-DD'), 'Low Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (24, 148, 120, TO\_DATE('2024-10-18', 'YYYY-MM-DD'), 'In Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (25, 149, 70, TO\_DATE('2024-10-19', 'YYYY-MM-DD'), 'Low Stock');

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (25, 150, 50, TO\_DATE('2024-10-20', 'YYYY-MM-DD'), 'Low Stock');

-- 3. Insert data into Suppliers table

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (1, 101, 5, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (2, 102, 15, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (3, 103, 7, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (4, 104, 20, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (5, 105, 3, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (6, 106, 6, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (7, 107, 10, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (8, 108, 12, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (9, 109, 2, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (10, 110, 25, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (11, 111, 4, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (12, 112, 18, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (13, 113, 5, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (14, 114, 7, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (15, 115, 3, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (16, 116, 12, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (17, 117, 10, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (18, 118, 6, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (19, 119, 15, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (20, 120, 4, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (21, 121, 8, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (22, 122, 3, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (23, 123, 12, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (24, 124, 6, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (25, 125, 9, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (26, 126, 4, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (27, 127, 13, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (28, 128, 11, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (29, 129, 6, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (30, 130, 8, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (31, 131, 5, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (32, 132, 9, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (33, 133, 7, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (34, 134, 11, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (35, 135, 6, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (36, 136, 3, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (37, 137, 14, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (38, 138, 10, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (39, 139, 5, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (40, 140, 9, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (41, 141, 8, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (42, 142, 7, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (43, 143, 6, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (44, 144, 12, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (45, 145, 5, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (46, 146, 8, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (47, 147, 4, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (48, 148, 9, 'Delivered late');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (49, 149, 11, 'Delivered on time');

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History) VALUES (50, 150, 7, 'Delivered late');

-- 4. Insert data into SalesTransactions table

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1001, 101, 1, 50, TO\_DATE('2024-10-01', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1002, 102, 1, 5, TO\_DATE('2024-10-02', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1003, 103, 2, 30, TO\_DATE('2024-10-03', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1004, 104, 2, 0, TO\_DATE('2024-10-04', 'YYYY-MM-DD')); -- Stockout

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1005, 105, 3, 100, TO\_DATE('2024-10-05', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1006, 106, 3, 75, TO\_DATE('2024-10-06', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1007, 107, 4, 20, TO\_DATE('2024-10-07', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1008, 108, 4, 0, TO\_DATE('2024-10-08', 'YYYY-MM-DD')); -- Stockout

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1009, 109, 5, 50, TO\_DATE('2024-10-09', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1010, 110, 5, 15, TO\_DATE('2024-10-10', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1011, 111, 6, 45, TO\_DATE('2024-10-11', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1012, 112, 6, 0, TO\_DATE('2024-10-12', 'YYYY-MM-DD')); -- Stockout

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1013, 113, 7, 75, TO\_DATE('2024-10-13', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1014, 114, 7, 35, TO\_DATE('2024-10-14', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1015, 115, 8, 10, TO\_DATE('2024-10-15', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1016, 116, 8, 80, TO\_DATE('2024-10-16', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1017, 117, 9, 50, TO\_DATE('2024-10-17', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1018, 118, 9, 15, TO\_DATE('2024-10-18', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1019, 119, 10, 0, TO\_DATE('2024-10-19', 'YYYY-MM-DD')); -- Stockout

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1020, 120, 10, 100, TO\_DATE('2024-10-20', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1021, 121, 11, 25, TO\_DATE('2024-10-21', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1022, 122, 11, 30, TO\_DATE('2024-10-22', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1023, 123, 12, 50, TO\_DATE('2024-10-23', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1024, 124, 12, 15, TO\_DATE('2024-10-24', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1025, 125, 13, 40, TO\_DATE('2024-10-25', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1026, 126, 13, 60, TO\_DATE('2024-10-26', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1027, 127, 14, 75, TO\_DATE('2024-10-27', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1028, 128, 14, 10, TO\_DATE('2024-10-28', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1029, 129, 15, 55, TO\_DATE('2024-10-29', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1030, 130, 15, 35, TO\_DATE('2024-10-30', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1031, 131, 16, 20, TO\_DATE('2024-10-31', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1032, 132, 16, 90, TO\_DATE('2024-11-01', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1033, 133, 17, 40, TO\_DATE('2024-11-02', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1034, 134, 17, 50, TO\_DATE('2024-11-03', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1035, 135, 18, 35, TO\_DATE('2024-11-04', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1036, 136, 18, 60, TO\_DATE('2024-11-05', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1037, 137, 19, 50, TO\_DATE('2024-11-06', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1038, 138, 19, 15, TO\_DATE('2024-11-07', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1039, 139, 20, 70, TO\_DATE('2024-11-08', 'YYYY-MM-DD'));

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (1040, 140, 20, 50, TO\_DATE('2024-11-09', 'YYYY-MM-DD'));

# APPENDIX B: CRUD OPERATIONS

The following PL/SQL scripts were developed to facilitate fundamental database interactions within the inventory management system. These operations, Create, Read, Update, and Delete, enable users to manage and manipulate data across key tables such as Products, Inventory, Sales Transactions, and Suppliers. Each operation is implemented as a PL/SQL procedure or function to streamline database management, ensuring that users can easily add new records, retrieve specific data, modify existing entries, and remove outdated information. The scripts include error handling to maintain data integrity and support efficient, secure access to essential inventory data.

-- Insert into Products

CREATE OR REPLACE PROCEDURE insert\_product(

p\_product\_id INT,

p\_product\_name VARCHAR2

) IS

BEGIN

INSERT INTO Products (Product\_ID, Product\_Name)

VALUES (p\_product\_id, p\_product\_name);

END;

/

-- Insert into Inventory

CREATE OR REPLACE PROCEDURE insert\_inventory(

p\_store\_id INT,

p\_product\_id INT,

p\_stock\_level INT,

p\_stock\_status\_date DATE,

p\_stock\_status VARCHAR2

) IS

BEGIN

INSERT INTO Inventory (Store\_ID, Product\_ID, Stock\_Level, Stock\_Status\_Date, Stock\_Status)

VALUES (p\_store\_id, p\_product\_id, p\_stock\_level, p\_stock\_status\_date, p\_stock\_status);

END;

/

-- Insert into Suppliers

CREATE OR REPLACE PROCEDURE insert\_supplier(

p\_supplier\_id INT,

p\_product\_id INT,

p\_lead\_time INT,

p\_delivery\_history VARCHAR2

) IS

BEGIN

INSERT INTO Suppliers (Supplier\_ID, Product\_ID, Lead\_Time, Delivery\_History)

VALUES (p\_supplier\_id, p\_product\_id, p\_lead\_time, p\_delivery\_history);

END;

/

-- Insert into SalesTransactions

CREATE OR REPLACE PROCEDURE insert\_sales\_transaction(

p\_transaction\_id INT,

p\_product\_id INT,

p\_store\_id INT,

p\_quantity\_sold INT,

p\_sale\_date DATE

) IS

BEGIN

INSERT INTO SalesTransactions (Transaction\_ID, Product\_ID, Store\_ID, Quantity\_Sold, Sale\_Date)

VALUES (p\_transaction\_id, p\_product\_id, p\_store\_id, p\_quantity\_sold, p\_sale\_date);

END;

/

-- Read Products

CREATE OR REPLACE FUNCTION get\_product(

p\_product\_id INT

) RETURN VARCHAR2 IS

v\_product\_name VARCHAR2(50);

BEGIN

SELECT Product\_Name INTO v\_product\_name

FROM Products WHERE Product\_ID = p\_product\_id;

RETURN v\_product\_name;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 'No product found';

END;

/

-- Read Inventory

CREATE OR REPLACE FUNCTION get\_inventory(

p\_store\_id INT,

p\_product\_id INT

) RETURN VARCHAR2 IS

v\_stock\_status VARCHAR2(20);

BEGIN

SELECT Stock\_Status INTO v\_stock\_status

FROM Inventory WHERE Store\_ID = p\_store\_id AND Product\_ID = p\_product\_id;

RETURN v\_stock\_status;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 'No inventory found';

END;

/

-- Read Suppliers

CREATE OR REPLACE FUNCTION get\_supplier(

p\_supplier\_id INT

) RETURN VARCHAR2 IS

v\_delivery\_history VARCHAR2(125);

BEGIN

SELECT Delivery\_History INTO v\_delivery\_history

FROM Suppliers WHERE Supplier\_ID = p\_supplier\_id;

RETURN v\_delivery\_history;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 'No supplier found';

END;

/

-- Read SalesTransactions

CREATE OR REPLACE FUNCTION get\_sales\_transaction(

p\_transaction\_id INT

) RETURN VARCHAR2 IS

v\_sale\_info VARCHAR2(200);

BEGIN

SELECT 'Product ID: ' || Product\_ID || ', Store ID: ' || Store\_ID || ', Quantity Sold: ' || Quantity\_Sold || ', Sale Date: ' || Sale\_Date

INTO v\_sale\_info

FROM SalesTransactions WHERE Transaction\_ID = p\_transaction\_id;

RETURN v\_sale\_info;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 'No transaction found';

END;

/

-- Update Products

CREATE OR REPLACE PROCEDURE update\_product(

p\_product\_id INT,

p\_product\_name VARCHAR2

) IS

BEGIN

UPDATE Products

SET Product\_Name = p\_product\_name

WHERE Product\_ID = p\_product\_id;

END;

/

-- Update Inventory

CREATE OR REPLACE PROCEDURE update\_inventory(

p\_store\_id INT,

p\_product\_id INT,

p\_stock\_level INT,

p\_stock\_status VARCHAR2

) IS

BEGIN

UPDATE Inventory

SET Stock\_Level = p\_stock\_level, Stock\_Status = p\_stock\_status

WHERE Store\_ID = p\_store\_id AND Product\_ID = p\_product\_id;

END;

/

-- Update Suppliers

CREATE OR REPLACE PROCEDURE update\_supplier(

p\_supplier\_id INT,

p\_lead\_time INT,

p\_delivery\_history VARCHAR2

) IS

BEGIN

UPDATE Suppliers

SET Lead\_Time = p\_lead\_time, Delivery\_History = p\_delivery\_history

WHERE Supplier\_ID = p\_supplier\_id;

END;

/

-- Update SalesTransactions

CREATE OR REPLACE PROCEDURE update\_sales\_transaction(

p\_transaction\_id INT,

p\_quantity\_sold INT,

p\_sale\_date DATE

) IS

BEGIN

UPDATE SalesTransactions

SET Quantity\_Sold = p\_quantity\_sold, Sale\_Date = p\_sale\_date

WHERE Transaction\_ID = p\_transaction\_id;

END;

/

-- Delete from Products

CREATE OR REPLACE PROCEDURE delete\_product(

p\_product\_id INT

) IS

BEGIN

DELETE FROM Products WHERE Product\_ID = p\_product\_id;

END;

/

-- Delete from Inventory

CREATE OR REPLACE PROCEDURE delete\_inventory(

p\_store\_id INT,

p\_product\_id INT

) IS

BEGIN

DELETE FROM Inventory WHERE Store\_ID = p\_store\_id AND Product\_ID = p\_product\_id;

END;

/

-- Delete from Suppliers

CREATE OR REPLACE PROCEDURE delete\_supplier(

p\_supplier\_id INT

) IS

BEGIN

DELETE FROM Suppliers WHERE Supplier\_ID = p\_supplier\_id;

END;

/

-- Delete from SalesTransactions

CREATE OR REPLACE PROCEDURE delete\_sales\_transaction(

p\_transaction\_id INT

) IS

BEGIN

DELETE FROM SalesTransactions WHERE Transaction\_ID = p\_transaction\_id;

END;

/

# APPENDIX C: PL/SQL CODE IMPLEMENTATION TO ANSWER QUERIES

The Appendix provides the complete PL/SQL code for each operation and procedure discussed in the report. These scripts include the SQL queries, methods, and functions that address the ten key questions, allowing users to understand and replicate the functionality designed to improve Walmart’s inventory management system.

/\*Patterns in Product Demand Across Regions and Seasons: *Aggregate sales data by region and season to identify demand patterns.*\*/

CREATE OR REPLACE PROCEDURE get\_demand\_patterns IS

CURSOR demand\_cursor IS

SELECT Store\_ID, EXTRACT(MONTH FROM Sale\_Date) AS Month,

SUM(Quantity\_Sold) AS Total\_Sold

FROM SalesTransactions

GROUP BY Store\_ID, EXTRACT(MONTH FROM Sale\_Date)

ORDER BY Store\_ID, Month;

BEGIN

FOR demand\_rec IN demand\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Store: ' || demand\_rec.Store\_ID ||

', Month: ' || demand\_rec.Month ||

', Total Sold: ' || demand\_rec.Total\_Sold);

END LOOP;

END;

/

/\*Predict Stock Shortages Before They Occur: Implement a procedure to check stock levels and compare them with historical average sales. If stock is below a threshold, trigger a reorder.\*/

CREATE OR REPLACE PROCEDURE predict\_stock\_shortage(p\_product\_id IN NUMBER, p\_store\_id IN NUMBER) IS

v\_avg\_sales NUMBER;

v\_stock\_level NUMBER;

BEGIN

-- Calculate average monthly sales for the product

SELECT AVG(Quantity\_Sold) INTO v\_avg\_sales

FROM SalesTransactions

WHERE Product\_ID = p\_product\_id AND Store\_ID = p\_store\_id;

-- Check current stock level

SELECT Stock\_Level INTO v\_stock\_level

FROM Inventory

WHERE Product\_ID = p\_product\_id AND Store\_ID = p\_store\_id;

-- Output message if stock level is below the average monthly sales

IF v\_stock\_level < v\_avg\_sales THEN

DBMS\_OUTPUT.PUT\_LINE('Stock level is low. Consider reordering.');

END IF;

END;

/

/\*Factors Contributing to Overstock Situations: Identify products with low sales but high stock levels to locate overstock.\*/

CREATE OR REPLACE PROCEDURE identify\_overstock IS

CURSOR overstock\_cursor IS

SELECT i.Product\_ID, i.Store\_ID, i.Stock\_Level,

COALESCE(SUM(st.Quantity\_Sold), 0) AS Total\_Sold

FROM Inventory i

LEFT JOIN SalesTransactions st

ON i.Product\_ID = st.Product\_ID AND i.Store\_ID = st.Store\_ID

GROUP BY i.Product\_ID, i.Store\_ID, i.Stock\_Level

HAVING i.Stock\_Level > 1.5 \* COALESCE(SUM(st.Quantity\_Sold), 1);

BEGIN

FOR overstock\_rec IN overstock\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Product: ' || overstock\_rec.Product\_ID ||

', Store: ' || overstock\_rec.Store\_ID ||

', Stock Level: ' || overstock\_rec.Stock\_Level ||

', Total Sold: ' || overstock\_rec.Total\_Sold);

END LOOP;

END;

/

/\*Optimize Reorder Points for Product Categories: Calculate average lead times from suppliers and use historical sales to determine reorder points.\*/

CREATE OR REPLACE PROCEDURE calculate\_reorder\_points IS

CURSOR reorder\_cursor IS

SELECT st.Product\_ID, AVG(s.Lead\_Time) AS Avg\_Lead\_Time,

AVG(st.Quantity\_Sold) \* AVG(s.Lead\_Time) AS Reorder\_Point

FROM SalesTransactions st

JOIN Suppliers s ON st.Product\_ID = s.Product\_ID

GROUP BY st.Product\_ID;

BEGIN

FOR reorder\_rec IN reorder\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Product: ' || reorder\_rec.Product\_ID ||

', Avg Lead Time: ' || reorder\_rec.Avg\_Lead\_Time ||

', Reorder Point: ' || reorder\_rec.Reorder\_Point);

END LOOP;

END;

/

/\*Impact of Inaccurate Inventory Data on Sales Performance: Query to identify cases where a sale could not be completed due to low or inaccurate stock levels.\*/

CREATE OR REPLACE PROCEDURE check\_inaccurate\_inventory\_data IS

CURSOR inaccurate\_cursor IS

SELECT st.Transaction\_ID, st.Product\_ID, st.Store\_ID,

st.Quantity\_Sold, i.Stock\_Level

FROM SalesTransactions st

JOIN Inventory i ON st.Product\_ID = i.Product\_ID AND st.Store\_ID = i.Store\_ID

WHERE i.Stock\_Level < st.Quantity\_Sold;

BEGIN

FOR inaccurate\_rec IN inaccurate\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Transaction: ' || inaccurate\_rec.Transaction\_ID ||

', Product: ' || inaccurate\_rec.Product\_ID ||

', Store: ' || inaccurate\_rec.Store\_ID ||

', Quantity Sold: ' || inaccurate\_rec.Quantity\_Sold ||

', Stock Level: ' || inaccurate\_rec.Stock\_Level);

END LOOP;

END;

/

/\*Improve Supplier Lead Time Estimates: Calculate the lead time average for each supplier based on historical deliveries.\*/

CREATE OR REPLACE PROCEDURE improve\_supplier\_lead\_times IS

CURSOR supplier\_cursor IS

SELECT Supplier\_ID, Product\_ID, AVG(Lead\_Time) AS Avg\_Lead\_Time

FROM Suppliers

GROUP BY Supplier\_ID, Product\_ID;

BEGIN

FOR supplier\_rec IN supplier\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Supplier: ' || supplier\_rec.Supplier\_ID ||

', Product: ' || supplier\_rec.Product\_ID ||

', Avg Lead Time: ' || supplier\_rec.Avg\_Lead\_Time);

END LOOP;

END;

/

/\*Products with Highest Return Rates and Their Impact on Inventory: Track return rates and identify products with high return rates.\*/

CREATE OR REPLACE PROCEDURE track\_high\_return\_products IS

CURSOR return\_cursor IS

SELECT Product\_ID, COUNT(\*) AS Return\_Count

FROM SalesTransactions

WHERE Quantity\_Sold < 0

GROUP BY Product\_ID

ORDER BY Return\_Count DESC;

BEGIN

FOR return\_rec IN return\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Product: ' || return\_rec.Product\_ID ||

', Return Count: ' || return\_rec.Return\_Count);

END LOOP;

END;

/

/\*Reduce Time to Restock Shelves After Sell-Outs: Identify products with low stock and flag for immediate restocking based on supplier lead times.\*/

CREATE OR REPLACE PROCEDURE restock\_recommendation(p\_product\_id IN NUMBER, p\_store\_id IN NUMBER) IS

v\_lead\_time NUMBER;

v\_stock\_level NUMBER;

BEGIN

-- Fetch supplier lead time for product

SELECT AVG(Lead\_Time) INTO v\_lead\_time

FROM Suppliers

WHERE Product\_ID = p\_product\_id;

-- Get current stock level

SELECT Stock\_Level INTO v\_stock\_level

FROM Inventory

WHERE Product\_ID = p\_product\_id AND Store\_ID = p\_store\_id;

IF v\_stock\_level = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Restock recommended for Product ' || p\_product\_id || ' at Store ' || p\_store\_id);

END IF;

END;

/

/\*Identify Slow-Moving Items and Adjust Stock Levels: Query to find products with minimal sales over a given period.\*/

CREATE OR REPLACE PROCEDURE find\_slow\_moving\_items IS

CURSOR slow\_cursor IS

SELECT Product\_ID, Store\_ID, SUM(Quantity\_Sold) AS Total\_Sold

FROM SalesTransactions

WHERE Sale\_Date > ADD\_MONTHS(SYSDATE, -6)

GROUP BY Product\_ID, Store\_ID

HAVING SUM(Quantity\_Sold) < 10;

BEGIN

FOR slow\_rec IN slow\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Product: ' || slow\_rec.Product\_ID ||

', Store: ' || slow\_rec.Store\_ID ||

', Total Sold: ' || slow\_rec.Total\_Sold);

END LOOP;

END;

/

/\*Impact of Return Rates on Inventory Management: Calculate the percentage of returns relative to total sales for each product.\*/

CREATE OR REPLACE PROCEDURE calculate\_return\_rates IS

CURSOR return\_rate\_cursor IS

SELECT Product\_ID,

SUM(CASE WHEN Quantity\_Sold < 0 THEN ABS(Quantity\_Sold) ELSE 0 END) /

SUM(ABS(Quantity\_Sold)) \* 100 AS Return\_Rate

FROM SalesTransactions

GROUP BY Product\_ID

HAVING SUM(CASE WHEN Quantity\_Sold < 0 THEN ABS(Quantity\_Sold) ELSE 0 END) /

SUM(ABS(Quantity\_Sold)) \* 100 > 5;

BEGIN

FOR return\_rate\_rec IN return\_rate\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Product: ' || return\_rate\_rec.Product\_ID ||

', Return Rate: ' || return\_rate\_rec.Return\_Rate || '%');

END LOOP;

END;

/

# APPENDIX D: TRIGGERS

Triggers are essential components in database management systems, designed to execute predefined actions automatically when specific events occur. In the Inventory Management Solution, triggers play a crucial role in ensuring the integrity and accuracy of stock-related operations. This appendix outlines the triggers implemented in the database, including their purpose and functionality. The triggers ensure sufficient stock levels before sales transactions and update supplier delivery history, automating critical tasks and maintaining data consistency.

/\*

This trigger checks that the stock level is sufficient before a sale proceeds. If the stock level is too low, an error is raised.

\*/

CREATE OR REPLACE TRIGGER trg\_check\_stock\_level BEFORE

INSERT ON salestransactions

FOR EACH ROW

DECLARE

insufficient\_stock EXCEPTION;

stock\_level NUMBER;

BEGIN

-- Retrieve stock level for the product

SELECT stock\_level

INTO stock\_level

FROM inventory

WHERE product\_id = :new.product\_id AND store\_id = :new.store\_id;

-- Check if stock is insufficient

IF :new.quantity\_sold > stock\_level THEN

RAISE insufficient\_stock;

END IF;

EXCEPTION

WHEN insufficient\_stock THEN

raise\_application\_error(-20001, 'Insufficient stock for this transaction.');

END;

/

/\*

This trigger updates the Delivery\_History in the Suppliers table every time there’s a new inventory entry from a supplier.

\*/

CREATE OR REPLACE TRIGGER trg\_supplier\_delivery

AFTER INSERT ON Inventory

FOR EACH ROW

BEGIN

UPDATE Suppliers

SET Delivery\_History = Delivery\_History || ', Delivered on ' || TO\_CHAR(SYSDATE, 'YYYY-MM-DD')

WHERE Supplier\_ID = (SELECT Supplier\_ID FROM Suppliers WHERE Product\_ID = :NEW.Product\_ID);

END;

/

# APPENDIX E: CONSTRAINTS

Constraints are rules enforced on database tables to maintain data integrity and consistency. They help ensure that data adheres to the defined schema and business logic. In the Inventory Management Solution, constraints are used to validate critical aspects of the system, such as maintaining positive sales quantities, ensuring non-negative stock levels, and upholding the relationships between products, inventory, and suppliers. This appendix documents the constraints implemented in the database and explains how they contribute to robust inventory management.

/\* Ensures that each sale has a positive quantity \*/

ALTER TABLE SalesTransactions

ADD CONSTRAINT chk\_positive\_quantity\_sold CHECK (Quantity\_Sold > 0);

/\* Ensures each entry in the `Inventory` table references a valid product.\*/

ALTER TABLE Inventory

ADD CONSTRAINT fk\_inventory\_product FOREIGN KEY (Product\_ID)

REFERENCES Products (Product\_ID);

/\*Ensures that each supplier’s product is valid in the `Products` table.\*/

ALTER TABLE Suppliers

ADD CONSTRAINT fk\_supplier\_product FOREIGN KEY (Product\_ID)

REFERENCES Products (Product\_ID);

/\*Ensures that stock levels are never negative.\*/

ALTER TABLE Inventory

ADD CONSTRAINT chk\_stock\_level\_non\_negative CHECK (Stock\_Level >= 0);

/\*Ensures each product appears only once per store.\*/

ALTER TABLE Inventory

ADD CONSTRAINT uq\_product\_store UNIQUE (Product\_ID, Store\_ID);

# APPENDIX F: COMPREHENSIVE INVENTORY MANAGEMENT PACKAGE IMPLEMENTATION

This appendix provides the complete implementation details of the Inventory Management Package developed for Walmart. The package encapsulates various procedures, functions, and triggers that support critical inventory management operations. It includes CRUD operations for managing products, inventory, suppliers, and sales transactions, as well as advanced business logic for analyzing demand patterns, predicting stock shortages, optimizing reorder points, and addressing operational inefficiencies. This appendix serves as a technical reference, showcasing the structured and modular design of the package, which ensures scalability, maintainability, and seamless integration with the database system.

CREATE OR REPLACE PACKAGE g3\_inventory\_pkg AS

-- ### Products CRUD Operations ###

PROCEDURE add\_product (

p\_product\_id IN NUMBER,

p\_product\_name IN VARCHAR2

);

PROCEDURE update\_product (

p\_product\_id IN NUMBER,

p\_product\_name IN VARCHAR2

);

PROCEDURE delete\_product (

p\_product\_id IN NUMBER

);

PROCEDURE get\_product (

p\_product\_id IN NUMBER

);

-- ### Inventory CRUD Operations ###

PROCEDURE add\_inventory (

p\_product\_id IN NUMBER,

p\_store\_id IN NUMBER,

p\_stock\_level IN NUMBER

);

PROCEDURE update\_inventory (

p\_product\_id IN NUMBER,

p\_store\_id IN NUMBER,

p\_stock\_level IN NUMBER

);

PROCEDURE delete\_inventory (

p\_product\_id IN NUMBER,

p\_store\_id IN NUMBER

);

PROCEDURE get\_inventory (

p\_product\_id IN NUMBER,

p\_store\_id IN NUMBER

);

-- ### Suppliers CRUD Operations ###

PROCEDURE add\_supplier (

p\_supplier\_id IN NUMBER,

p\_product\_id IN NUMBER,

p\_lead\_time IN NUMBER,

p\_delivery\_history IN VARCHAR2

);

PROCEDURE update\_supplier (

p\_supplier\_id IN NUMBER,

p\_lead\_time IN NUMBER,

p\_delivery\_history IN VARCHAR2

);

PROCEDURE delete\_supplier (

p\_supplier\_id IN NUMBER

);

PROCEDURE get\_supplier (

p\_supplier\_id IN NUMBER

);

-- ### Sales Transactions CRUD Operations ###

PROCEDURE add\_sale (

p\_transaction\_id IN NUMBER,

p\_product\_id IN NUMBER,

p\_store\_id IN NUMBER,

p\_quantity\_sold IN NUMBER,

p\_sale\_date IN DATE

);

PROCEDURE update\_sale (

p\_transaction\_id IN NUMBER,

p\_quantity\_sold IN NUMBER,

p\_sale\_date IN DATE

);

PROCEDURE delete\_sale (

p\_transaction\_id IN NUMBER

);

PROCEDURE get\_sale (

p\_transaction\_id IN NUMBER

);

-- ### Utility Procedures for Business Rules ###

PROCEDURE enforce\_stock\_level (

p\_product\_id IN NUMBER,

p\_store\_id IN NUMBER,

p\_quantity\_sold IN NUMBER

);

-- ### Business Logic Procedures ###

PROCEDURE get\_demand\_patterns;

PROCEDURE predict\_stock\_shortage(p\_product\_id IN NUMBER, p\_store\_id IN NUMBER);

PROCEDURE identify\_overstock;

PROCEDURE calculate\_reorder\_points;

PROCEDURE check\_inaccurate\_inventory\_data;

PROCEDURE improve\_supplier\_lead\_times;

PROCEDURE track\_high\_return\_products;

PROCEDURE restock\_recommendation(p\_product\_id IN NUMBER, p\_store\_id IN NUMBER);

PROCEDURE find\_slow\_moving\_items;

PROCEDURE calculate\_return\_rates;

END g3\_inventory\_pkg;

/

CREATE OR REPLACE PACKAGE BODY g3\_inventory\_pkg AS

-- ### Products CRUD Operations ###

PROCEDURE add\_product (p\_product\_id IN NUMBER, p\_product\_name IN VARCHAR2) IS

BEGIN

INSERT INTO products (product\_id, product\_name)

VALUES (p\_product\_id, p\_product\_name);

END add\_product;

PROCEDURE update\_product (p\_product\_id IN NUMBER, p\_product\_name IN VARCHAR2) IS

BEGIN

UPDATE products

SET product\_name = p\_product\_name

WHERE product\_id = p\_product\_id;

END update\_product;

PROCEDURE delete\_product (p\_product\_id IN NUMBER) IS

BEGIN

DELETE FROM products WHERE product\_id = p\_product\_id;

END delete\_product;

PROCEDURE get\_product (p\_product\_id IN NUMBER) IS

v\_product\_name VARCHAR2(50);

BEGIN

SELECT product\_name INTO v\_product\_name

FROM products WHERE product\_id = p\_product\_id;

DBMS\_OUTPUT.PUT\_LINE('Product Name: ' || v\_product\_name);

END get\_product;

-- ### Inventory CRUD Operations ###

PROCEDURE add\_inventory (p\_product\_id IN NUMBER, p\_store\_id IN NUMBER, p\_stock\_level IN NUMBER) IS

BEGIN

INSERT INTO inventory (product\_id, store\_id, stock\_level)

VALUES (p\_product\_id, p\_store\_id, p\_stock\_level);

END add\_inventory;

PROCEDURE update\_inventory (p\_product\_id IN NUMBER, p\_store\_id IN NUMBER, p\_stock\_level IN NUMBER) IS

BEGIN

UPDATE inventory

SET stock\_level = p\_stock\_level

WHERE product\_id = p\_product\_id AND store\_id = p\_store\_id;

END update\_inventory;

PROCEDURE delete\_inventory (p\_product\_id IN NUMBER, p\_store\_id IN NUMBER) IS

BEGIN

DELETE FROM inventory

WHERE product\_id = p\_product\_id AND store\_id = p\_store\_id;

END delete\_inventory;

PROCEDURE get\_inventory (p\_product\_id IN NUMBER, p\_store\_id IN NUMBER) IS

v\_stock\_level NUMBER;

BEGIN

SELECT stock\_level INTO v\_stock\_level

FROM inventory WHERE product\_id = p\_product\_id AND store\_id = p\_store\_id;

DBMS\_OUTPUT.PUT\_LINE('Stock Level: ' || v\_stock\_level);

END get\_inventory;

-- ### Suppliers CRUD Operations ###

PROCEDURE add\_supplier (p\_supplier\_id IN NUMBER, p\_product\_id IN NUMBER, p\_lead\_time IN NUMBER, p\_delivery\_history IN VARCHAR2) IS

BEGIN

INSERT INTO suppliers (supplier\_id, product\_id, lead\_time, delivery\_history)

VALUES (p\_supplier\_id, p\_product\_id, p\_lead\_time, p\_delivery\_history);

END add\_supplier;

PROCEDURE update\_supplier (p\_supplier\_id IN NUMBER, p\_lead\_time IN NUMBER, p\_delivery\_history IN VARCHAR2) IS

BEGIN

UPDATE suppliers

SET lead\_time = p\_lead\_time, delivery\_history = p\_delivery\_history

WHERE supplier\_id = p\_supplier\_id;

END update\_supplier;

PROCEDURE delete\_supplier (p\_supplier\_id IN NUMBER) IS

BEGIN

DELETE FROM suppliers WHERE supplier\_id = p\_supplier\_id;

END delete\_supplier;

PROCEDURE get\_supplier (p\_supplier\_id IN NUMBER) IS

v\_lead\_time NUMBER;

v\_delivery\_history VARCHAR2(125);

BEGIN

SELECT lead\_time, delivery\_history INTO v\_lead\_time, v\_delivery\_history

FROM suppliers WHERE supplier\_id = p\_supplier\_id;

DBMS\_OUTPUT.PUT\_LINE('Lead Time: ' || v\_lead\_time || ', Delivery History: ' || v\_delivery\_history);

END get\_supplier;

-- ### Sales Transactions CRUD Operations ###

PROCEDURE add\_sale (p\_transaction\_id IN NUMBER, p\_product\_id IN NUMBER, p\_store\_id IN NUMBER, p\_quantity\_sold IN NUMBER, p\_sale\_date IN DATE) IS

BEGIN

INSERT INTO salestransactions (transaction\_id, product\_id, store\_id, quantity\_sold, sale\_date)

VALUES (p\_transaction\_id, p\_product\_id, p\_store\_id, p\_quantity\_sold, p\_sale\_date);

END add\_sale;

PROCEDURE update\_sale (p\_transaction\_id IN NUMBER, p\_quantity\_sold IN NUMBER, p\_sale\_date IN DATE) IS

BEGIN

UPDATE salestransactions

SET quantity\_sold = p\_quantity\_sold, sale\_date = p\_sale\_date

WHERE transaction\_id = p\_transaction\_id;

END update\_sale;

PROCEDURE delete\_sale (p\_transaction\_id IN NUMBER) IS

BEGIN

DELETE FROM salestransactions WHERE transaction\_id = p\_transaction\_id;

END delete\_sale;

PROCEDURE get\_sale (p\_transaction\_id IN NUMBER) IS

v\_quantity\_sold NUMBER;

v\_sale\_date DATE;

BEGIN

SELECT quantity\_sold, sale\_date INTO v\_quantity\_sold, v\_sale\_date

FROM salestransactions WHERE transaction\_id = p\_transaction\_id;

DBMS\_OUTPUT.PUT\_LINE('Quantity Sold: ' || v\_quantity\_sold || ', Sale Date: ' || TO\_CHAR(v\_sale\_date, 'YYYY-MM-DD'));

END get\_sale;

-- ### Utility Procedures ###

PROCEDURE enforce\_stock\_level (p\_product\_id IN NUMBER, p\_store\_id IN NUMBER, p\_quantity\_sold IN NUMBER) IS

v\_stock\_level NUMBER;

BEGIN

SELECT stock\_level INTO v\_stock\_level

FROM inventory WHERE product\_id = p\_product\_id AND store\_id = p\_store\_id;

IF v\_stock\_level < p\_quantity\_sold THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient stock for this transaction.');

END IF;

END enforce\_stock\_level;

-- ### Business Logic Procedures ###

PROCEDURE get\_demand\_patterns IS

CURSOR demand\_cursor IS

SELECT Store\_ID, EXTRACT(MONTH FROM Sale\_Date) AS Month, SUM(Quantity\_Sold) AS Total\_Sold

FROM SalesTransactions GROUP BY Store\_ID, EXTRACT(MONTH FROM Sale\_Date) ORDER BY Store\_ID, Month;

BEGIN

FOR demand\_rec IN demand\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Store: ' || demand\_rec.Store\_ID || ', Month: ' || demand\_rec.Month || ', Total Sold: ' || demand\_rec.Total\_Sold);

END LOOP;

END get\_demand\_patterns;

-- Add all other business logic procedures as per your request...

END g3\_inventory\_pkg;

/