/\*

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;

/

/\* 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);

/\*Package Implementation\*/

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);

END G3\_inventory\_pkg;

/

-- Package Body: Implements the functionality declared in the package specification.

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 for Business Rules ###

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;

END G3\_inventory\_pkg;

/

/\*Milestone 1 Questions answer\*/

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

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;

**/\*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.\*/**

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);

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

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;

**/\*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.\*/**

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;

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

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

FROM Suppliers

GROUP BY Supplier\_ID, Product\_ID;

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

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

FROM SalesTransactions

WHERE Quantity\_Sold < 0

GROUP BY Product\_ID

ORDER BY Return\_Count DESC;

**/\*Reduce Time to Restock Shelves After Sell-Outs : Identify products with low stock and flag for immediate restock 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.\*/**

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;

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

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;

### **Team Members' Contributions**

| Team Member | Contribution |
| --- | --- |
| Samuel Gomez | Triggers, Constraints, Questions Answered |
| Nicholas Lauw | Introductory paragraphs |
| Noah Zhou | Trigger and Constraint Packaging |
| John Hsu | Triggers and Packages |