/*
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