**Database Triggers - The Superstore Business Challenge**

**Scenario 1: Stock level management**The trigger takes care of inventory levels for products\_dim in nearly real-time whenever an order is placed in Orders\_fact. Whenever a product is ordered, the stock is decremented based on the quantity ordered. At the same time, it alerts when stock has gone below a critical supply limit, allowing time to replenish products properly when necessary. This talk recently eliminated the manual component where errors may have occurred due to procedures. The whole purpose for this mechanism is to ensure that the stock in the warehouse is not relying on manual stock controls through external documents. The business can operate as it should!  
  
ALTER TABLE products\_dim ADD COLUMN stocklevel INT DEFAULT 100;

CREATE OR REPLACE FUNCTION update\_stocklevel()

RETURNS TRIGGER AS $$

DECLARE new\_stock INT;

BEGIN

UPDATE products\_dim SET stocklevel = stocklevel - NEW."Quantity"

WHERE "Product ID" = NEW."Product ID";

SELECT stocklevel INTO new\_stock FROM products\_dim

WHERE "Product ID" = NEW."Product ID";

IF new\_stock < 10 THEN

RAISE NOTICE 'Stock for Product % is low (current: %)', NEW."Product ID", new\_stock;

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS trg\_update\_stocklevel ON Orders\_fact;

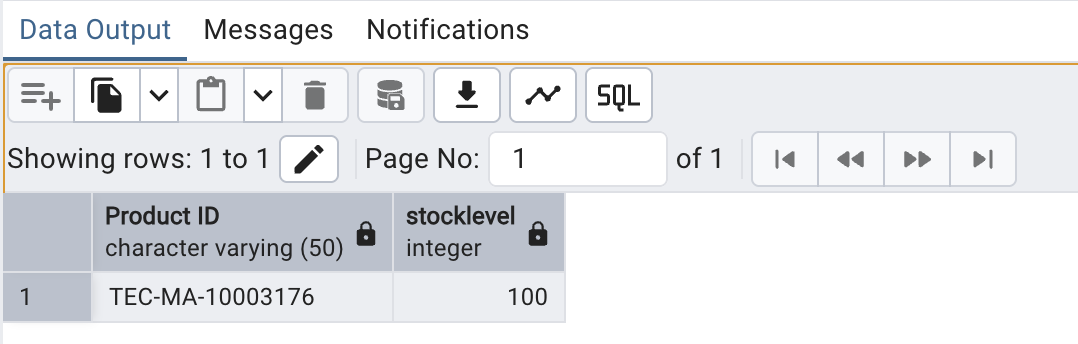
CREATE TRIGGER trg\_update\_stocklevel

AFTER INSERT ON Orders\_fact

FOR EACH ROW

EXECUTE FUNCTION update\_stocklevel();

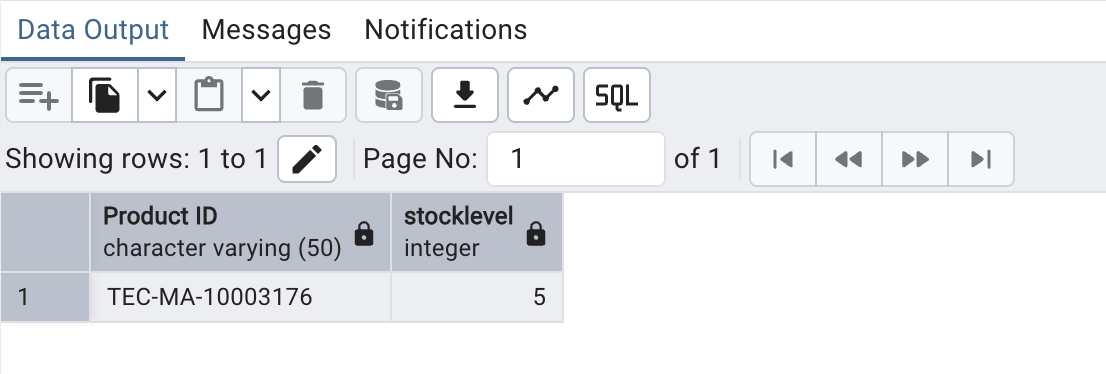
SELECT "Product ID", stocklevel FROM products\_dim WHERE "Product ID" = 'TEC-MA-10003176';



INSERT INTO orders\_fact("Row ID","Order ID","Order Date","Ship Date","Ship Mode","Customer ID","Product ID","Sales ($)","Quantity","Discount($)","Profit($)")

VALUES (9999999,'TEST-INV-001','2025-09-01','2025-09-02','Standard','CUST-TEST','TEC-MA-10003176', 500.00, 95, 0.0, 0.0);

SELECT "Product ID", stocklevel FROM products\_dim WHERE "Product ID" = 'TEC-MA-10003176';



**Scenario 2: Automated profit calculation**

The trigger ensures that the Profit($) column in Orders\_fact will be automatically calculated for each order. Therefore, the calculation is not reliant on a manual process, which is a common source of errors. The follow-up costs from the Products\_Fact are looked up and the sales, discount, and quantity are factored to demonstrate the true profitability of the sales reporting. The benefits of automation lead to valid and consistent reporting, speedier reports, and decisions against BI metrics. The trigger eliminates the possibility of human errors associated when completing financial calculations and documenting subsequent variables from BI reporting.

ALTER TABLE orders\_fact ADD COLUMN "Profit($)" NUMERIC;

ALTER TABLE Products\_Fact ADD COLUMN IF NOT EXISTS "Cost" NUMERIC DEFAULT 20;

CREATE OR REPLACE FUNCTION auto\_calc\_profit()

RETURNS TRIGGER AS $$

DECLARE prod\_cost NUMERIC;

BEGIN

SELECT "Cost" INTO prod\_cost

FROM Products\_Fact

WHERE "Product ID" = NEW."Product ID";

IF prod\_cost IS NULL THEN prod\_cost := 20;

END IF;

NEW."Profit($)" := NEW."Sales ($)" - (NEW."Sales ($)" \* NEW."Discount($)") - (NEW."Quantity" \* prod\_cost);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_calc\_profit

BEFORE INSERT ON Orders\_fact

FOR EACH ROW

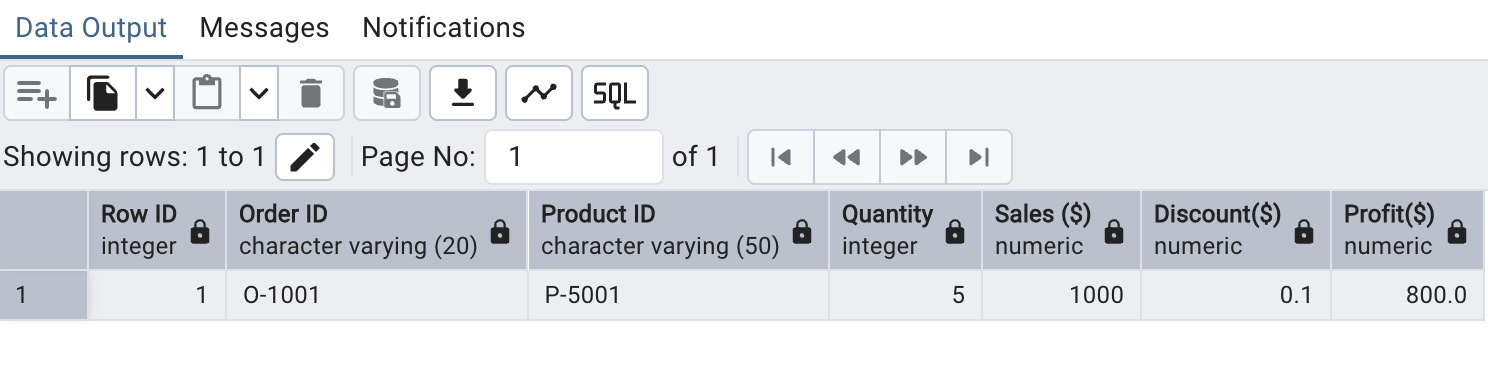
EXECUTE FUNCTION auto\_calc\_profit();

INSERT INTO Orders\_fact ("Row ID", "Order ID", "Product ID", "Quantity", "Sales ($)", "Discount($)")

VALUES (1, 'O-1001', 'P-5001', 5, 1000, 0.1);

SELECT "Row ID", "Order ID", "Product ID", "Quantity", "Sales ($)", "Discount($)", "Profit($)"

FROM Orders\_fact WHERE "Order ID" = 'O-1001';



**Scenario 3: Cross-table statement management**

This trigger takes care of updating the order status to 'Returned' in the Orders table when all items of that order have been returned, as recorded in the Returns table. It aggregates the level of returns, using the condition that the returned quantity of each item in the order should equal the order quantity to update the order status. This sort of automation helps companies out by reducing manual updates, improving accuracy of customer service data and enhancing the reliability of any operational reporting, while providing much better visibility in tracking returns and maintaining the integrity of order lifecycle information.

CREATE TABLE IF NOT EXISTS Orders (

"OrderID" VARCHAR(20) PRIMARY KEY,

"OrderStatus" VARCHAR(20),

"TotalQuantity" INT

);

CREATE TABLE IF NOT EXISTS Returns (

"ReturnID" SERIAL PRIMARY KEY,

"OrderID" VARCHAR(50) REFERENCES Orders("OrderID"),

"QuantityReturned" INT

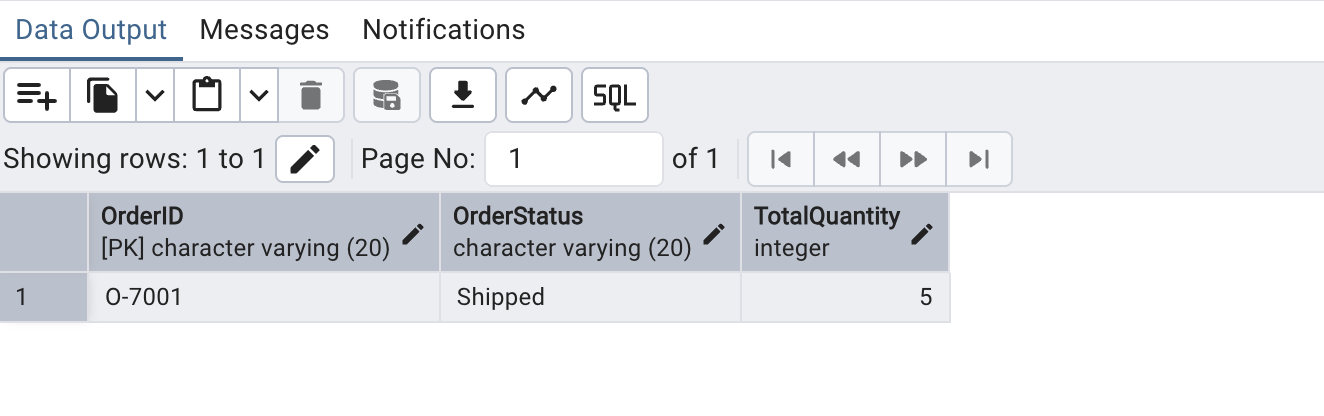
);

INSERT INTO Orders ("OrderID", "OrderStatus", "TotalQuantity")

VALUES ('O-7001', 'Shipped', 5);

-- Check initial status

SELECT \* FROM Orders WHERE "OrderID" = 'O-7001';



CREATE OR REPLACE FUNCTION update\_order\_status()

RETURNS TRIGGER AS $$

DECLARE

total\_returned INT;

total\_ordered INT;

BEGIN

SELECT SUM("QuantityReturned") INTO total\_returned FROM Return s

WHERE "OrderID" = NEW."OrderID";

SELECT "TotalQuantity" INTO total\_ordered FROM Orders

WHERE "OrderID" = NEW."OrderID";

IF total\_returned >= total\_ordered THEN

UPDATE Orders

SET "OrderStatus" = 'Returned'

WHERE "OrderID" = NEW."OrderID";

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_update\_order\_status

AFTER INSERT ON Returns

FOR EACH ROW

EXECUTE FUNCTION update\_order\_status();

INSERT INTO Returns ("OrderID", "QuantityReturned") VALUES ('O-7001', 3);

INSERT INTO Returns ("OrderID", "QuantityReturned") VALUES ('O-7001', 2);

-- Check final status

SELECT \* FROM Orders WHERE "OrderID" = 'O-7001';

