**PROCEDURES**

1. **Create a procedure that simulates applying a discount rate for products in a specific category during a special occasion. The procedure will take a CategoryID, a DiscountPercentage for the special occasion, and a StartDate for the special occasion. It will then apply the discount to products in the specified category, but only if the special occasion is ongoing**.

/\*Create a procedure that simulates applying a discount rate for products in a specific category during a special occasion.

The procedure will take a CategoryID, a DiscountPercentage for the special occasion, and a StartDate for the special occasion.

It will then apply the discount to products in the specified category, but only if the special occasion is ongoing.\*/

CREATE OR REPLACE PROCEDURE ApplyDiscountForSpecialOccasion(

p\_CategoryID IN CATEGORY.CategoryID%TYPE,

p\_DiscountPercentage IN NUMBER,

p\_StartDate IN DATE

) IS

-- Declare cursor

CURSOR ProductCursor IS

SELECT P.ProductID, P.ProductName, P.UnitPrice

FROM PRODUCT P

JOIN SUBCATEGORY SC ON P.subCategoryID = SC.subCategoryID

WHERE SC.CategoryID = p\_CategoryID;

-- Declare variables

dw\_ProductID PRODUCT.ProductID%TYPE;

dw\_ProductName PRODUCT.ProductName%TYPE;

dw\_OriginalUnitPrice PRODUCT.UnitPrice%TYPE;

dw\_DiscountedUnitPrice PRODUCT.UnitPrice%TYPE;

-- Variable to check if the category exists

dw\_CategoryExists NUMBER := 0;

BEGIN

-- Check if the provided CategoryID exists

SELECT COUNT(\*)

INTO dw\_CategoryExists

FROM CATEGORY

WHERE CategoryID = p\_CategoryID;

-- If the CategoryID doesn't exist, raise an exception

IF dw\_CategoryExists = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Invalid or non-existent CategoryID: ' || p\_CategoryID);

END IF;

-- Open the cursor

OPEN ProductCursor;

-- Fetch the first row

FETCH ProductCursor INTO dw\_ProductID, dw\_ProductName, dw\_OriginalUnitPrice;

-- Loop through the cursor

WHILE ProductCursor%FOUND LOOP

-- Check if the special occasion is ongoing

IF p\_StartDate <= SYSDATE THEN

-- Calculate the discounted unit price

dw\_DiscountedUnitPrice := dw\_OriginalUnitPrice - (dw\_OriginalUnitPrice \* p\_DiscountPercentage / 100);

-- Update the product's unit price

UPDATE PRODUCT

SET UnitPrice = dw\_DiscountedUnitPrice

WHERE ProductID = dw\_ProductID;

-- Display update information

DBMS\_OUTPUT.PUT\_LINE('Discount applied for Product ID: ' || dw\_ProductID);

DBMS\_OUTPUT.PUT\_LINE('Product Name: ' || dw\_ProductName);

DBMS\_OUTPUT.PUT\_LINE('Original Unit Price: ' || dw\_OriginalUnitPrice);

DBMS\_OUTPUT.PUT\_LINE('Discounted Unit Price: ' || dw\_DiscountedUnitPrice);

DBMS\_OUTPUT.PUT\_LINE('----------------------');

ELSE

-- Display information that the special occasion is not ongoing

DBMS\_OUTPUT.PUT\_LINE('Special Occasion has not started yet.');

END IF;

-- Fetch the next row

FETCH ProductCursor INTO dw\_ProductID, dw\_ProductName, dw\_OriginalUnitPrice;

END LOOP;

-- Close the cursor

CLOSE ProductCursor;

DBMS\_OUTPUT.PUT\_LINE('Discount application completed.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No products found in the specified category.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' || SQLERRM);

RAISE;

END ApplyDiscountForSpecialOccasion;

/

1. **Create a procedure that checks whether a customer has exceeded their credit limit before placing an order. If the customer's total outstanding balance (sum of all unpaid orders) exceeds their credit limit, the procedure will raise an exception.**

CREATE OR REPLACE PROCEDURE CreditLimitChecker(p\_CustomerID IN CUSTOMER.CustomerID%TYPE,

p\_OrderTotalValue IN ORDERS.OrderTotalValue%TYPE) IS

dw\_TotalOutstandingBalance DECIMAL(10, 2);

dw\_CreditLimit DECIMAL(10, 2);

BEGIN

-- Get the total outstanding balance for the customer

SELECT NVL(SUM(OD.ODTotalValue), 0) INTO dw\_TotalOutstandingBalance

FROM ORDERS O

JOIN ORDERDETAILS OD ON O.OrderID = OD.OrderID

WHERE O.CustomerID = p\_CustomerID

AND O.OrderType IN ('Online', 'Offline');

-- Get the credit limit for the customer

SELECT CreditLimit INTO dw\_CreditLimit

FROM CUSTOMER

WHERE CustomerID = p\_CustomerID;

-- Check if the order exceeds the credit limit

IF dw\_TotalOutstandingBalance + p\_OrderTotalValue > dw\_CreditLimit THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Order exceeds credit limit. Cannot proceed with the order.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Credit limit check passed. Order can be placed.');

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Customer not found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' || SQLERRM);

RAISE;

END CreditLimitChecker;

/

1. **Generate OrderID when inserting a record into the Order table using triggers**

CREATE OR REPLACE TRIGGER OrderIDTriggerGenerator

BEFORE INSERT ON ORDERS

FOR EACH ROW

DECLARE

dw\_MaxOrderID NUMBER;

BEGIN

-- Get the maximum existing OrderID

SELECT MAX(TO\_NUMBER(SUBSTR(OrderID, 3)))

INTO dw\_MaxOrderID

FROM ORDERS;

-- If there are no existing records, set the starting value to 0

IF dw\_MaxOrderID IS NULL THEN

dw\_MaxOrderID := 0;

END IF;

-- Generate the next OrderID by incrementing the maximum value

:NEW.OrderID := 'OR' || LPAD(dw\_MaxOrderID + 1, 8, '0');

EXCEPTION

-- Handle exceptions, for example, if the SELECT INTO returns no rows

WHEN NO\_DATA\_FOUND THEN

-- If there are no existing records, set the starting value to 0

dw\_MaxOrderID := 0;

-- Generate the next OrderID by incrementing the starting value

:NEW.OrderID := 'OR' || LPAD(dw\_MaxOrderID + 1, 8, '0');

WHEN OTHERS THEN

-- Handle other exceptions (customize as needed)

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);

-- You may choose to raise or log the exception or take other actions

END OrderIDTriggerGenerator;

/