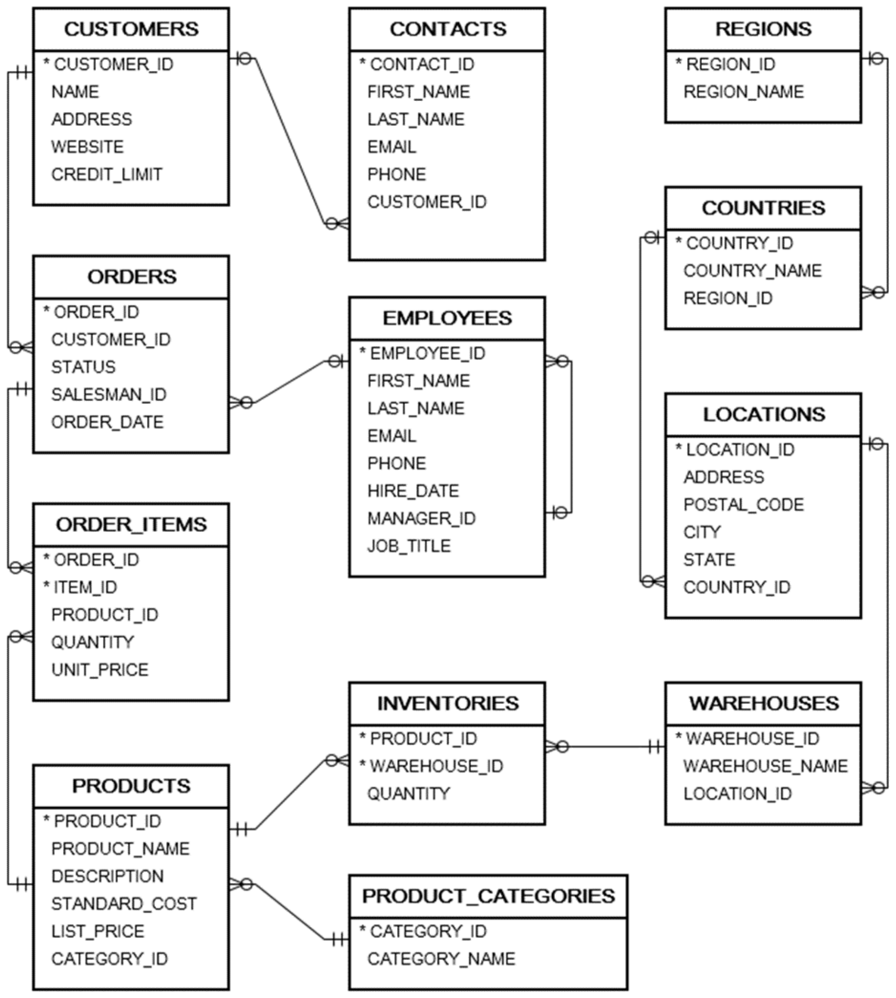
IS480 Advanced Database Management Project

(Chelsea Lam, Jon Aldrete, Travis Chanchang, Eduardo Anaya, Brandon Pham, Catherine Dao)

Business Situation:

We are a company that focuses on selling computer hardware. Our products consist of graphics cards, CPUs, personal computers, and RAM sticks. The products are sold and shipped to Europe, The Americas, Asia, Middle East, and Africa. Using our system, we can efficiently create new orders, locate warehouses close to customers' locations, update prices when prices change, find inflation in every country, transfer products to different warehouses, update the database, and create a procedure for adding new customers.

Tables:



The tables that we used consisted of order_items, orders, inventories, products, warehouses, locations, and countries. We used the order_items to make parameters and variables that would contribute to making a new order and verifying the stock of products. The orders table was used to gather the data needed for the order ID of a product. The inventories table was used to collect data regarding the amount of product and inventory there was in the old and new warehouses. The products table was used to calculate the inflation of products based on different countries and regions, to create new orders, and to transfer products from an old to a new warehouse. The warehouses table was mainly used in our moveProduct procedure that would move a certain amount of product to different warehouses. Lastly, the countries and locations table were used in regards to our InflationIncrease procedure that would calculate the inflation per region or country.

Functions:

Our first function is named fPrice and its primary function is to find what region is selling the most. In the function there are two variables named vPrice and vDiscount, they output the value that is assigned to them. Furthermore, we then created a variable named vDiscountedPrice, which finds the discounted price. We set vDiscountedPrice equal to the difference of vPrice and the product of vPrice and vDiscount. Finally we return vDiscountedPrice and END the function.

We also have a function named checkInventory that looks up the amount there is in computer products in the original warehouse inventory. There are two parameters in this function, one for product and another one for the old warehouse. These parameters go by the name p_product and p_oldWarehouse. We have a variable for the quantity of product in our inventory named v_quantityInInventory. Then we begin an SQL query where we extract data from our Inventories table, where we match the Product ID to our parameter p_product and also match the Inventory.ProductID to our parameter p_oldWarehouse. This then returns the quantity in inventory of products from the old warehouse.

Procedures:

The first procedure that we used was one that creates a new order; the name of the procedure is newOrder. The newOrder procedure contains five parameters and they consist of p_ORDERID, p_itemID, p_productID, p_Quantity, and p_Unitprice. This procedure contains a cursor named cOnhandQuantity, which grabs the inventories.quantity from the inventories table, where it matches the INVENTORIES.PRODUCT_ID to the parameter p_productID. We then instantiate four variables: v_OrderQuantity, v_onhand, v_TotalOrderedQuantity, and v_totalOnhandQuantity. We then BEGIN with equaling both v_TotalOrderedQuantity and v_totalOnhandQuantity to zero. Furthermore, we open the cOrderedQuantity cursor and create a while loop that fetches cOrderedQuantity INTO v_OrderQuantity. We then get the result of v_OrderQuantity plus v_TotalOrderedQuantity. The loop then ends and the cOrderedQuantity is closed. Moreover, we open another cursor named cOnhandQuantity. We create a while loop that fetches cOnhandQuantity INTO v_onhand and we get the result of v_onhand plus v_totalOnhandQuantity. The loop is then closed. We then created an IF THEN ELSE statement where IF v_TotalOrderedQuantity is less than or equal to v_totalOnhandQuantity THEN we insert the new order information into our ORDER_ITEMS table and the into our values which consist of p_ORDERID, p_itemID, p_productID, p_Quantity, and p_Unitprice. The ELSE statement will then be triggered with the print statement, "'We have less than what the customer orders. Backorder Accepted?'". The IF statement and procedure are then both ended.

Our next procedure deals with new orders, the name of the procedure is p_NewOrders. We have three parameters for this procedure named p_order_ID, p_order_Date, and p_customer_ID. Under BEGIN this procedure INSERTS INTO order which consists of the order_ID, order_Date, and customerID.

Our third procedure consists of updating the prices of our products based on inflation rate per country. The name of the procedure is InflationIncrease and it contains two parameters named p_CountryID and P_InflationRate. We also have two variables named v_unit_price and v_inflationRate. We then create a cursor named cPrices. We selected the unit_price table from the order_items, products, inventories, warehouses, locations, and countries columns where the product ID of the order items is equal to product ID of the product, product ID of the products is equal to the product ID of the inventories, warehouse ID of the inventory is equal to the warehouse ID of the warehouses, the location ID of the warehouses is equal to the location ID of the locations, the country ID of the locations is equal to the country ID of the countries and where the country ID of the countries is equal to our country ID parameter: p_CountryID. We continue in the BEGIN clause by equalling the v_inflationRate variable to p_InflationRate. We then open the cPrices cursor, start a loop, then fetch cPrices INTO the variable v_unit_price. Then we SET unit_prices equal to unit_price multiplied by (1 + v_inflationRate). The loop is then exited when there are no more prices found. The loop is then ended, we close the cPrices cursor, and we end the procedure. Finally we execute InflationIncrease ('US', .25).

Moreover, we have another procedure named moveProduct. The main purpose of this procedure is to move a product to another warehouse/destination. In this procedure we have four parameters: p_product, p_quantity, p_oldWarehouse, and p_newWarehouse. The parameter p_product is the product, p_quantity is the quantity being moved, p_oldWarehouse is the original warehouse, and p_newWarehouse is the new destination warehouse. We instantiated two variables v_quantityInInventory and v_overRequestedAmount. We then added four more variables that contain parameters, these variables are v_product, v_quantity, v_oldWarehouse , and v_newWarehouses. Moreover we start with the BEGIN clause where we equal our four variables to the four parameters we have instantiated. Furthermore, we ran a query that looks up the warehouse ID and quantity of product in the dataset. We then made an if statement thats checks if there is excess stock over 100 units in the particular warehouse for a particular product where IF v_quantityInInventory > 100 and p_quantity <= v_quantityInInventory THEN the v_quantityInInventory is set equal to checkInventory(p_product,p_oldWarehouse). After this, we can print out a print statement that prints out the amount of product that went from the old warehouse to the new warehouse. We then ran an if statement and query that adds the quantity of product in the new warehouse. The if statement is IF v_quantityInInventory > 100 and p_quantity <= v_quantityInInventory THEN we update the inventories and set the quantity to the difference of the v_quantityInInventory and p_quantity WHERE INVENTORIES.PRODUCT_ID is equal to our parameter for product AND INVENTORIES.WAREHOUSE_ID is equal to our parameter for the oldWarehouse. We then continue with the print statement that prints the amount of products left in the old warehouse. Furthermore, we made an ELSIF statement if the amount requested is over the amount in the inventory, then the system will print out “You want to transfer too many of'|| p_product”. If that does not go through then the ELSE statement will be triggered which prints out “Can not transfer the products”. The if statement then ends and we end the entire procedure.

Our final procedure is called New_Customer and it involves adding a new customer to our database. We made five parameters in the procedure and they are named p_customerID, p_name, p_address, p_website, and p_creditlimit. In the BEGIN clause we INSERT INTO CUSTOMERS alongside their CUSTOMER_ID, NAME, ADDRESS, WEBSITE, and CREDIT_LIMIT then we assign the values with the five parameters we instantiated earlier. We then end the procedure.

Packages:

We have a package named Bizops which contains the function checkInventory and the procedure moveProduct. The checkInventory function is responsible for looking up the quantity of product that there is in the original warehouse inventory. The moveProduct procedure uses the checkInventory function to move certain product amounts to different warehouses.

Code

IF-Then-ELSE Statements – Create New Order and Verify Stock:

```
CREATE OR REPLACE PROCEDURE newOrder(
p_ORDERID IN ORDER_ITEMS.ORDER_ID%TYPE,
p_itemID IN ORDER_ITEMS.ITEM_ID%TYPE,
p_productID IN ORDER_ITEMS.PRODUCT_ID%TYPE,
p_Quantity IN ORDER_ITEMS.QUANTITY%TYPE,
p_Unitprice IN ORDER_ITEMS.UNIT_PRICE%TYPE
) AS

---- Vars for containing the parameters data

v_ORDERID ORDER_ITEMS.ORDER_ID%TYPE;
v_itemID ORDER_ITEMS.ITEM_ID%TYPE;
v_productID ORDER_ITEMS.PRODUCT_ID%TYPE;
v_Quantity ORDER_ITEMS.QUANTITY%TYPE;
v_Unitprice ORDER_ITEMS.UNIT_PRICE%TYPE;

CURSOR cOrderedQuantity IS
SELECT Order_items.quantity
from order_items, orders
WHERE ORDER_ITEMS.ORDER_ID = ORDERS.ORDER_ID
AND product_id = p_productID
AND ORDERS.STATUS = 'Pending';

CURSOR cOnhandQuantity IS
SELECT inventories.quantity
FROM INVENTORIES
WHERE INVENTORIES.PRODUCT_ID = p_productID;

v_OrderQuantity ORDER_ITEMS.QUANTITY%TYPE;
v_onhand INVENTORIES.QUANTITY%TYPE;
v_TotalOrderedQuantity ORDER_ITEMS.QUANTITY%TYPE;
v_totalOnhandQuantity ORDER_ITEMS.QUANTITY%TYPE;

BEGIN

-- Pass parameters to variables
v_ORDERID := p_ORDERID;
v_itemID := p_itemID;
v_productID := p_productID;
```

```
v_Quantity := p_Quantity;
v_Unitprice := p_Unitprice;

v_TotalOrderedQuantity := 0;
v_totalOnhandQuantity := 0;

OPEN cOrderedQuantity;
WHILE cOrderedQuantity%FOUND LOOP
FETCH cOrderedQuantity INTO v_OrderQuantity;
v_TotalOrderedQuantity :=+ v_OrderQuantity;

END LOOP;
CLOSE cOrderedQuantity;

OPEN cOnhandQuantity;
WHILE cOnhandQuantity%FOUND
LOOP
FETCH cOnhandQuantity INTO v_onhand;
v_totalOnhandQuantity :=+ v_onhand;
END LOOP;

IF v_TotalOrderedQuantity <= v_totalOnhandQuantity THEN
INSERT INTO ORDER_ITEMS(ORDER_ID, ITEM_ID, PRODUCT_ID, QUANTITY,
UNIT_PRICE)
VALUES (v_ORDERID, v_itemID, v_productID, v_Quantity, v_Unitprice);

ELSE
dbms_output.put_line('We have less than what the customer orders. Backorder Accepted?');
END IF;
END;
```

Find which warehouse is closest to where most customers are located

Procedure that adds new orders to order tables

```
CREATE OR REPLACE PROCEDURE p_NewOrders (
   p_order_ID orders.order_ID%TYPE,
   p_order_Date orders.order_Date%TYPE,
   p_customer_ID orders.customerID%TYPE,
) AS

BEGIN

INSERT INTO orders (order_ID, order_Date, customerID);
```

Function to decrease price by 10%

```
SET SERVEROUTPUT ON
SET ECHO ON

DECLARE

vPrice NUMBER;
vDiscount NUMBER;
vDiscountedPrice NUMBER;

BEGIN;

vPrice := 100;
vDiscount := .10;

vDiscountedPrice := vPrice-(vPrice*vDiscount);

dbms_output.put_line(vDiscountedPrice);

END;

CREATE OR REPLACE FUNCTION fPrice (
   vPrice NUMBER,
   vDiscount NUMBER
) RETURN NUMBER AS

vDiscountedPrice NUMBER;

BEGIN

vDiscountedPrice := vPrice-(vPrice*vDiscount);

RETURN vDiscountedPrice;

END;
```

LOOPs and **Cursors - Inflation Rate Procedure based on each country:**

```
--Inflation Rate Procedure based on each country
SET SERVEROUTPUT ON
SET ECHO ON

CREATE OR REPLACE PROCEDURE InflationIncrease(
    p_CountryID countries.country_id%type,
    P_InflationRate NUMBER) AS

v_unit_price order_items.unit_price%type;
v_inflationRate NUMBER;

CURSOR cPrices IS
SELECT unit_price
```

```
FROM order_items, products, inventories, warehouses, locations,
countries
WHERE order_items.product_id = products.product_id
AND  products.product_id = inventories.product_id
AND inventories.warehouse_id = warehouses.warehouse_id
AND warehouses.location_id = locations.location_id
AND locations.country_id = countries.country_id
AND countries.country_id = p_CountryID;
BEGIN
v_inflationRate := p_InflationRate;
OPEN cPrices;
LOOP
FETCH cPrices INTO v_unit_price;
    UPDATE order_items
    SET unit_price = unit_price * (1 + v_inflationRate);
EXIT WHEN cPrices%NOTFOUND;
END LOOP;
CLOSE cPrices;

END;

EXEC InflationIncrease ('US', .25);
```

Single attribute data retrieval (Select Into ….) and **Modifications/Updates to the DB - Move products to different warehouses for order fulfillment:**

```
SET SERVEROUTPUT ON
SET ECHO ON

--FUNCTION-----------------------------------------------------------------------------------
--look up the quantity in the original warehouse inventory for a particular product
CREATE OR REPLACE FUNCTION checkInventory(
   p_product IN INVENTORIES.PRODUCT_ID%Type, --the product
   p_oldWarehouse IN INVENTORIES.WAREHOUSE_ID%Type --the original warehouse
)RETURN NUMBER AS

v_quantityInInventory INVENTORIES.QUANTITY%TYPE;

BEGIN

SELECT INVENTORIES.QUANTITY INTO v_quantityInInventory
FROM INVENTORIES
WHERE INVENTORIES.PRODUCT_ID = p_product
AND INVENTORIES.WAREHOUSE_ID = p_oldWarehouse;
COMMIT;
RETURN v_quantityInInventory;
```

```
END;
—-----------------------------------------------------------------------------------------------------------

SET SERVEROUTPUT ON
SET ECHO ON

--PROCEDURE-----------------------------------------------------------------------------------------
--Move a certain product amount to different warehouses

CREATE OR REPLACE PROCEDURE moveProduct(
   p_product IN INVENTORIES.PRODUCT_ID%Type, --the product
   p_quantity IN INVENTORIES.QUANTITY%Type, --the quantity being moved
   p_oldWarehouse IN INVENTORIES.WAREHOUSE_ID%Type, --the original warehouse
   p_newWarehouse IN INVENTORIES.WAREHOUSE_ID%Type -- the new destinastion
warehouse
) AS

v_quantityInInventory INVENTORIES.QUANTITY%TYPE;
v_overRequestedAmount INVENTORIES.QUANTITY%TYPE;

---- Add variables to contain the parameters
v_product INVENTORIES.PRODUCT_ID%Type;
v_quantity INVENTORIES.QUANTITY%Type;
v_oldWarehouse INVENTORIES.WAREHOUSE_ID%Type;
v_newWarehouse INVENTORIES.WAREHOUSE_ID%Type;

BEGIN

v_product:= p_product;
v_quantity := p_quantity;
v_oldWarehouse := p_oldWarehouse;
v_newWarehouse := p_newWarehouse;

--look up the warehouse id and quantity of the product in the dataset
-- SELECT INVENTORIES.QUANTITY INTO v_quantityInInventory
-- FROM INVENTORIES
-- WHERE INVENTORIES.PRODUCT_ID = p_product
-- AND INVENTORIES.WAREHOUSE_ID = p_oldWarehouse;

--if there is excess stock over 100 units in the particular warehouse for a particular product
--and the requested amount is less than or equal to the quantity amount in the inventory
-- IF v_quantityInInventory > 100  and p_quantity <= v_quantityInInventory THEN
v_quantityInInventory := checkInventory(p_product,p_oldWarehouse);
DBMS_OUTPUT.PUT_LINE('There is ' || v_quantityInInventory || ' of ' || p_product ||
' from ' || p_oldWarehouse || ' to ' || p_newWarehouse);
```

```
 IF v_quantityInInventory > 100  and p_quantity <= v_quantityInInventory THEN
--add the amount in the  new warehouse
UPDATE INVENTORIES
SET QUANTITY = (INVENTORIES.QUANTITY + p_quantity)
WHERE INVENTORIES.PRODUCT_ID = p_product
AND INVENTORIES.WAREHOUSE_ID = p_newWarehouse;

--subtract the amount being moved out of the original warehouse
UPDATE INVENTORIES
SET QUANTITY = (v_quantityInInventory - p_quantity)
WHERE INVENTORIES.PRODUCT_ID = p_product
AND INVENTORIES.WAREHOUSE_ID = p_oldWarehouse;

DBMS_OUTPUT.PUT_LINE('There is ' || (v_quantityInInventory - p_quantity) || ' of ' ||
p_product ||' in ' || p_oldWarehouse || ' left.');
--IF IT IS FALSE,

--1) the amount requested is over the amount in the inventory
ELSIF p_quantity > v_quantityInInventory THEN
v_overRequestedAmount := (p_quantity - v_quantityInInventory);
dbms_output.put_line('You want to transfer too many of'|| p_product);

--2)
ELSE
dbms_output.put_line('Can not transfer the products.');

END IF;

END;
```

Procedure to add new customers

```
SERVEROUTPUT ON
SET ECHO ON

CREATE OR REPLACE PROCEDURE New_Customer(
   p_customerID CUSTOMERS.CUSTOMER_ID%TYPE,
   p_name CUSTOMERS.NAME%TYPE,
   p_address CUSTOMERS.ADDRESS%TYPE,
   p_website CUSTOMERS.WEBSITE%TYPE,
   p_creditlimit CUSTOMERS.CREDIT_LIMIT%TYPE
)AS
```

```
BEGIN
INSERT INTO CUSTOMERS (CUSTOMER_ID, NAME, ADDRESS, WEBSITE,
CREDIT_LIMIT)
VALUES (p_customerID, p_name, p_address, p_website, p_creditlimit);

END;

SET SERVEROUTPUT ON
SET ECHO ON

CREATE OR REPLACE PROCEDURE UPDATE_CONTACTS(
   p_contactID CONTACTS.CONTACT_ID%TYPE,
   p_firstname CONTACTS.FIRST_NAME%TYPE,
   p_lastname CONTACTS.LAST_NAME%TYPE,
   p_email CONTACTS.EMAIL%TYPE,
   p_phone CONTACTS.PHONE%TYPE,
   p_customerID CUSTOMERS.CUSTOMER_ID%TYPE
)AS

BEGIN

UPDATE CONTACTS SET FIRST_NAME = 'Brandon', LAST_NAME = 'Tracy'
WHERE EMAIL = '@gmail.com';

END;
```

Package Code:

```
Package Specification/Declaration:

CREATE OR REPLACE PACKAGE BizOps IS

--checkInventory Function---------------
FUNCTION checkInventory(
   p_product IN INVENTORIES.PRODUCT_ID%Type, --the product
   p_oldWarehouse IN INVENTORIES.WAREHOUSE_ID%Type --the original warehouse
)RETURN NUMBER;
--newOrders Procedure-------------------
PROCEDURE newOrder(
p_ORDERID IN ORDER_ITEMS.ORDER_ID%TYPE,
p_itemID IN ORDER_ITEMS.ITEM_ID%TYPE,
p_productID IN ORDER_ITEMS.PRODUCT_ID%TYPE,
p_Quantity IN ORDER_ITEMS.QUANTITY%TYPE,
p_Unitprice IN ORDER_ITEMS.UNIT_PRICE%TYPE
);

--InflationIncrease Procedure----------------
PROCEDURE InflationIncrease(
   p_CountryID countries.country_id%type,
   P_InflationRate NUMBER);

--Procedure moveProduct-----------------------
PROCEDURE moveProduct(
   p_product IN INVENTORIES.PRODUCT_ID%Type, --the product
   p_quantity IN INVENTORIES.QUANTITY%Type, --the quantity being moved
   p_oldWarehouse IN INVENTORIES.WAREHOUSE_ID%Type, --the original warehouse
   p_newWarehouse IN INVENTORIES.WAREHOUSE_ID%Type -- the new destinastion
warehouse
);

--New_Customer Procedure---------------------
PROCEDURE New_Customer(
   p_customerID CUSTOMERS.CUSTOMER_ID%TYPE,
   p_name CUSTOMERS.NAME%TYPE,

   p_address CUSTOMERS.ADDRESS%TYPE,
   p_website CUSTOMERS.WEBSITE%TYPE,
   p_creditlimit CUSTOMERS.CREDIT_LIMIT%TYPE
);

--UPDATE_CONTACTS--------------------------
PROCEDURE UPDATE_CONTACTS(
   p_contactID CONTACTS.CONTACT_ID%TYPE,
   p_firstname CONTACTS.FIRST_NAME%TYPE,
   p_lastname CONTACTS.LAST_NAME%TYPE,
   p_email CONTACTS.EMAIL%TYPE,
   p_phone CONTACTS.PHONE%TYPE,
   p_customerID CUSTOMERS.CUSTOMER_ID%TYPE
);

END BizOps;
```

Package Body:

```
--Package Body
CREATE OR REPLACE PACKAGE BODY BizOps IS

--FIRST FUNCTION -------------------------------------------
--FUNCTION--------------------------------------------------------------------------------------
```

```
--look up the quantity in the original warehouse inventory for a particular product
FUNCTION checkInventory(
   p_product IN INVENTORIES.PRODUCT_ID%Type, --the product
   p_oldWarehouse IN INVENTORIES.WAREHOUSE_ID%Type --the original warehouse
)RETURN NUMBER AS

v_quantityInInventory INVENTORIES.QUANTITY%TYPE;

BEGIN

SELECT INVENTORIES.QUANTITY INTO v_quantityInInventory
FROM INVENTORIES
WHERE INVENTORIES.PRODUCT_ID = p_product
AND INVENTORIES.WAREHOUSE_ID = p_oldWarehouse;
COMMIT;
RETURN v_quantityInInventory;

END;

--NEWORDERS PROCEDURE------------------------------------------

--------------------------------------------------------------
PROCEDURE newOrder(
p_ORDERID IN ORDER_ITEMS.ORDER_ID%TYPE,
p_itemID IN ORDER_ITEMS.ITEM_ID%TYPE,
p_productID IN ORDER_ITEMS.PRODUCT_ID%TYPE,
p_Quantity IN ORDER_ITEMS.QUANTITY%TYPE,
p_Unitprice IN ORDER_ITEMS.UNIT_PRICE%TYPE
) AS

--cursor for finding quantity that is ordered based on the order ID and product ID for orders
that are pending
CURSOR cOrderedQuantity IS
SELECT Order_items.quantity
from order_items, orders
WHERE ORDER_ITEMS.ORDER_ID = ORDERS.ORDER_ID
AND product_id = p_productID
AND ORDERS.STATUS = 'Pending';

--cursor for finding quantity in the inventory currently for a specific product
CURSOR cOnhandQuantity IS
SELECT inventories.quantity
FROM INVENTORIES
WHERE INVENTORIES.PRODUCT_ID = p_productID;

--variables
```

```
v_OrderQuantity ORDER_ITEMS.QUANTITY%TYPE; --each time cursor loop, fetch the
amount
v_onhand INVENTORIES.QUANTITY%TYPE; --each time cursor loop, fetch the amount
v_TotalOrderedQuantity ORDER_ITEMS.QUANTITY%TYPE; --variable to save the total
ordered quantity
v_totalOnhandQuantity ORDER_ITEMS.QUANTITY%TYPE; --variable to save the total
inventory quantity

--variables for the parameters
v_ORDERID  ORDER_ITEMS.ORDER_ID%TYPE;
v_itemID  ORDER_ITEMS.ITEM_ID%TYPE;
v_productID  ORDER_ITEMS.PRODUCT_ID%TYPE;
v_Quantity  ORDER_ITEMS.QUANTITY%TYPE;
v_Unitprice  ORDER_ITEMS.UNIT_PRICE%TYPE;

BEGIN

v_TotalOrderedQuantity := 0;
v_totalOnhandQuantity := 0;

--setting the variables to the parameter

v_ORDERID := p_ORDERID;
v_itemID := p_itemID;
v_productID := p_productID;
v_Quantity := p_Quantity;
v_Unitprice := p_Unitprice;

OPEN cOrderedQuantity; --start the first cursor
WHILE cOrderedQuantity%FOUND LOOP --go through the cursor,while data is found, loop
and when it’s not found, exit
FETCH cOrderedQuantity INTO v_OrderQuantity; --fetch
v_TotalOrderedQuantity :=+ v_OrderQuantity; --add the quantities to get the total

END LOOP;
CLOSE cOrderedQuantity; --end the first cursor

OPEN cOnhandQuantity; --start the second cursor
WHILE cOnhandQuantity%FOUND
LOOP
FETCH cOnhandQuantity INTO v_onhand;
v_totalOnhandQuantity :=+ v_onhand; --add the quantities to get to inventory quantity of the
product
END LOOP;
CLOSE cOnhandQuantity; --end the second cursor
```

```
--If the total quantity that is ordered is more than the total quantity in the inventory for the
product
--then add the parameters into Order Items Table
IF v_TotalOrderedQuantity <= v_totalOnhandQuantity THEN

INSERT INTO ORDER_ITEMS(ORDER_ID, ITEM_ID, PRODUCT_ID, QUANTITY,
UNIT_PRICE)
-- VALUES ((Select max(ORDER_ID)+1 FROM ORDER_ITEMS),(Select max(Item_ID)+1
FROM ORDER_ITEMS), v_productID, v_Quantity, v_Unitprice);
VALUES (106,2,63,58,1718.88);

--Else print that the order can not be fulfilled
ELSE
dbms_output.put_line('We have less than what the customer orders. Backorder Accepted?');
END IF;
END;

--INFLATION INCREASE PROCEDURE-----------------------------------
-----------------------------------------------------------------
PROCEDURE InflationIncrease(
   p_CountryID countries.country_id%type,

   P_InflationRate NUMBER) AS

v_unit_price order_items.unit_price%type;
v_inflationRate NUMBER;

CURSOR cPrices IS
SELECT unit_price
FROM order_items, products, inventories, warehouses, locations, countries
WHERE order_items.product_id = products.product_id
AND  products.product_id = inventories.product_id
AND inventories.warehouse_id = warehouses.warehouse_id
AND warehouses.location_id = locations.location_id
AND locations.country_id = countries.country_id
AND countries.country_id = p_CountryID;
BEGIN
v_inflationRate := p_InflationRate;
OPEN cPrices;
LOOP
FETCH cPrices INTO v_unit_price;
   UPDATE order_items
   SET unit_price = unit_price * (1 + v_inflationRate);
EXIT WHEN cPrices%NOTFOUND;
```

```
END LOOP;
CLOSE cPrices;

END;


--CHECKINVENTORY PROCEDURE --------------------------------------
--THIS PROCEDURE USES THE FUNCTION "CHECKINVENTORY"
--Move a certain product amount to different warehouses

PROCEDURE moveProduct(
   p_product IN INVENTORIES.PRODUCT_ID%Type, --the product
   p_quantity IN INVENTORIES.QUANTITY%Type, --the quantity being moved
   p_oldWarehouse IN INVENTORIES.WAREHOUSE_ID%Type, --the original warehouse
   p_newWarehouse IN INVENTORIES.WAREHOUSE_ID%Type -- the new destinastion
warehouse
) AS

v_quantityInInventory INVENTORIES.QUANTITY%TYPE;
v_overRequestedAmount INVENTORIES.QUANTITY%TYPE;

---- Add variables to contain the parameters
v_product INVENTORIES.PRODUCT_ID%Type;
v_quantity INVENTORIES.QUANTITY%Type;

v_oldWarehouse INVENTORIES.WAREHOUSE_ID%Type;
v_newWarehouse INVENTORIES.WAREHOUSE_ID%Type;

BEGIN

v_product:= p_product;
v_quantity := p_quantity;
v_oldWarehouse := p_oldWarehouse;
v_newWarehouse := p_newWarehouse;


--if there is excess stock over 100 units in the particular warehouse for a particular product
--and the requested amount is less than or equal to the quantity amount in the inventory
-- IF v_quantityInInventory > 100  and p_quantity <= v_quantityInInventory THEN
v_quantityInInventory := checkInventory(p_product,p_oldWarehouse);
DBMS_OUTPUT.PUT_LINE('There is ' || v_quantityInInventory || ' of ' || p_product ||
' from ' || p_oldWarehouse || ' to ' || p_newWarehouse);


 IF v_quantityInInventory > 100  and p_quantity <= v_quantityInInventory THEN
--add the amount in the  new warehouse
```

```
UPDATE INVENTORIES
SET QUANTITY = (INVENTORIES.QUANTITY + p_quantity)
WHERE INVENTORIES.PRODUCT_ID = p_product
AND INVENTORIES.WAREHOUSE_ID = p_newWarehouse;

--subtract the amount being moved out of the original warehouse
UPDATE INVENTORIES
SET QUANTITY = (v_quantityInInventory - p_quantity)
WHERE INVENTORIES.PRODUCT_ID = p_product
AND INVENTORIES.WAREHOUSE_ID = p_oldWarehouse;

DBMS_OUTPUT.PUT_LINE('There is ' || (v_quantityInInventory - p_quantity) || ' of ' ||
p_product ||' in ' || p_oldWarehouse || ' left.');
--IF IT IS FALSE,

--1) the amount requested is over the amount in the inventory
ELSIF p_quantity > v_quantityInInventory THEN
v_overRequestedAmount := (p_quantity - v_quantityInInventory);
dbms_output.put_line('You want to transfer too many of'|| p_product);

--2)
ELSE
dbms_output.put_line('Can not transfer the products.');

END IF;

END;

--New_Customer Procedure---------------------------------
---------------------------------------------------------
PROCEDURE New_Customer(
   p_customerID CUSTOMERS.CUSTOMER_ID%TYPE,
   p_name CUSTOMERS.NAME%TYPE,
   p_address CUSTOMERS.ADDRESS%TYPE,
   p_website CUSTOMERS.WEBSITE%TYPE,
   p_creditlimit CUSTOMERS.CREDIT_LIMIT%TYPE
)AS


BEGIN
INSERT INTO CUSTOMERS (CUSTOMER_ID, NAME, ADDRESS, WEBSITE,
CREDIT_LIMIT)
VALUES (p_customerID, p_name, p_address, p_website, p_creditlimit);

END;
```

```
--UPDATE_CONTACTS-------------------------------------
------------------------------------------------------
PROCEDURE UPDATE_CONTACTS(
   p_contactID CONTACTS.CONTACT_ID%TYPE,
   p_firstname CONTACTS.FIRST_NAME%TYPE,
   p_lastname CONTACTS.LAST_NAME%TYPE,
   p_email CONTACTS.EMAIL%TYPE,
   p_phone CONTACTS.PHONE%TYPE,
   p_customerID CUSTOMERS.CUSTOMER_ID%TYPE
)AS

BEGIN

UPDATE CONTACTS SET FIRST_NAME = 'Brandon', LAST_NAME = 'Tracy'
WHERE EMAIL = '@gmail.com';

END;

END BizOps;
```

Problems the procedures and functions may solve and who would benefit:

There are many problems that our code solves. Our first procedure, newOrder creates a new order and verifies that there is stock in the warehouses. It solves a problem that customers may encounter where sometimes the order they place may get canceled due to there being no more items in stock for that item. Customers who are going to place an order are the main beneficiaries from this procedure as they get notified whether their preferred item is in stock or not before they place their order.

Our function, named fPrice serves the purpose of decreasing the price of a product by ten percent. This will be beneficial to customers who plan on using promo codes to get a discount on our products. The discounted price will be shown to them and they will be able to make the choice on whether they want to purchase the product or not.

Moreover, we have another procedure named InflationIncrease which calculates and updates the inflation per country. For example, if there is a customer that wants to purchase a product through another country's website then the inflation will be calculated and updated based on the county the customer resides in. The customer will be displayed the price of the product with the inflation already calculated.

We also have a procedure named movePoduct that moves products from the old warehouse into the new warehouses. This procedure benefits both the customer and the warehouse workers. It helps the customer because they will get shipping updates on their order. It helps the warehouse workers because it notifies them when a product arrives and when they should ship it out.

Lastly, we have procedures named UPDATE_CONTACTS, and New_Customer which add and update new customers onto the database. This will be beneficial to the management team as they need to constantly update the customer's information so that they know who they need to ship their products to. They can quickly update the contact information of the customer and contact them if there is any issue with their product.