COSC 344 Assignment 3 - MATTHEW BROOKER 541670

Procedure

- My Procedure allows supermarket managers to change the price of any product in the store. The manager would execute the program with the productID of the product they want to change and the proposed price.

- To use the procedure:
    1. Load the database by typing **@load** at the terminal window.
    2. Compile and create procedure by typing **@proc** at the terminal window.
    3. Make sure server output display is on by typing **SET SERVEROUTPUT ON;** at the terminal window.
    4. Execute the procedure successfully by typing at the terminal window: **EXEC changeProductPrice**(*number between 1000 and 1030 inclusive, price you want to set it to);*

       or for example:

EXEC changeProductPrice(*1000, 6.99);*

if you now type:

```
SELECT productid, price
FROM product
WHERE productid = 1000;
```

At the terminal window, you will see that the price of the flyspray is now $6.99 rather than $7.50, as it was before.

- To fire the exception clause execute the program with a non-existent productID, less than 1000 or more than 1030. You will get this output at the terminal window:

The productID you entered does not exist.

Trigger

- My trigger maintains a derived attribute called number_of_employees, found in department. this counts how many employees work for each department as employees are employed, fired or moved between departments.

- To use the trigger:

1. Load the database by typing **@load** at the terminal window.
2. Compile and create procedure by typing **@trig** at the terminal window.
3. Test the trigger:

For employing (INSERTING) a new employee, type this at the terminal window:

insert INTO employee
values('888333333','John','Y','Douglas',TO_DATE('12-06-1990','DD-MM-YYYY'), 'Pine Hill','M',10000, 20, '888665555', 2);

Now check to see if the derived attribute has added another employee to it's respective department number by typing:

SELECT * FROM department;

You will see that that the number of employees working for the Butchery is now 1.

To change (UPDATE) John's department number to 'Grocery', type this at the terminal.

UPDATE employee
SET dno = 3
WHERE fname = 'John';

The next time you type

SELECT * FROM department;

at the terminal window, you will find that in 'Grocery', the 'number_of_employees' will be incremented to 1 while the 'number_of_employees' will be decremented to 0 in 'Butchery.'

For firing (DELETING) this ‘John’ character, type this at the terminal window:

DELETE FROM employee
WHERE fname = ‘John’;

The next time you type

SELECT * FROM department;

at the terminal window, you will find that the number_of_employees found in ‘Grocery’ will be decremented to 0.

Proc.sql

```
-- This procedure allows managers to change the price of the available
-- products in the supermarket. If the productid input does not equal
-- an existing product id, an exception is raised.

CREATE OR REPLACE PROCEDURE changeProductPrice(priceChangeId IN CHAR,
                                    newPrice IN NUMBER)
AS
  CURSOR pr IS
        SELECT * FROM product;
  prod pr%ROWTYPE;
 noMatch EXCEPTION;

-- For each product, if the manager inputs an existing productid,
-- set the product's price to the new price and leave the program.
-- If the price is not changed, then the exception is raised.
BEGIN
  FOR prod IN pr LOOP
        IF prod.productid = priceChangeId THEN
        UPDATE product
        SET price = newPrice
        WHERE productid = priceChangeId;
        RETURN;
        END IF;
  END LOOP;
  RAISE noMatch;
```

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN NULL;
  WHEN noMatch THEN
        DBMS_OUTPUT.PUT_LINE('The productID you entered does not exist.');
END;
/
```

Trig.sql

```
-- This trigger calculates the number of employees
-- found in each department after an employee (or
-- specifically a department number) is inserted,
-- updated or deleted.

CREATE OR REPLACE TRIGGER numEmployees
AFTER INSERT OR UPDATE OR DELETE OF dno on employee
 FOR EACH ROW
 BEGIN
  IF INSERTING THEN
  -- Increment the no. of employees for the respective
  -- department number.
   UPDATE department
   SET number_of_employees = number_of_employees + 1
   WHERE dnumber = :NEW.dno;
  ELSIF UPDATING THEN
  -- Increment the no. of employees found in the new
  -- department and decrement from the old department.
   UPDATE department
   SET number_of_employees = number_of_employees + 1
   WHERE dnumber = :NEW.dno;
   UPDATE department
   SET number_of_employees = number_of_employees - 1
   WHERE dnumber = :OLD.dno;
  ELSE
  -- If deleting, decrement no. of employees found
  -- in the old department.
   UPDATE department
   SET number_of_employees = number_of_employees - 1
   WHERE dnumber = :OLD.dno;
  END IF;
END;
/
```

- The only change I made to the database was that I added the derived attribute **number_of_employees** and set their initial values to 0. It is an **INT** value with a **NOT NULL** constraint.