311-Lab05W8-stored procedures 1 RT

Lab 5 – Week 8   Stored Procedures/Conditional Statements)

Due Friday of week 8, which is the week after the reading week, before midnight.

Submission:

Your submission through Blackboard will be a single text-based SQL file with appropriate header and commenting.

Please ensure your file runs when the entire file is executed in SQL Developer..

The file will be run in Oracle. If it runs well, it gets the full marks. Not running gets a much lesser mark

Create a new Worksheet in SQL Developer. Save the file as L05_ID#_LASTNAME.sql

Your submission needs to be commented and include the question, the solutions. SEE EXAMPLE ON LAST PAGE

In this Lab, you create PL/SQL stored procedures to perform the following tasks. As you know, a stored procedure does not return any value. To send values back to the caller, you can use OUT parameters.

A parameter can be

- IN parameter
- OUT parameter
- IN OUT parameter

See the following template:

```
CREATE OR REPLACE procedure_name(arg1 IN/OUT/IN OUT data_type, ...)
AS
BEGIN
  ....
EXCEPTION
WHEN OTHERS
  THEN
      DBMS_OUTPUT.PUT_LINE (Error!');
END procedure_name;
```

For all the stored procedures make sure you handle all exceptions such as

- TOO_MANY_ROWS
- NO_DATA_FOUND
- OTHERS
- . . .

Besides checking all required exceptions, have the OTHER exception checked just in case any error occurs that has not been anticipated at the time you write the code.

Tasks

NOTE: Remember SET SERVEROUTPUT ON or you will not see anything working. Place it the beginning.

Remember ROLLBACK; to make sure the changes in question 3, 4 and 5 do not MESS UP MY PRODUCT TABLE

1. Write a stored procedure that gets an integer number and prints
   The number is even.
   If a number is divisible by 2.
   Otherwise, it prints
   The number is odd.

CREATE **OR REPLACE** is essential as you will mess up the next student's run of the script

2. Create a stored procedure named *find_me*. This procedure gets an employee number from the user and prints the following employee information about that user:
   First name Last name Email Phone Hire Date and Job ID

   The procedure gets a value as the employee ID of type NUMBER.
   See the following example for employee ID 107: This is not the answer you will get for employee 107

```
First name: Summer
Last name: Payne
Email: summer.payne@example.com
Phone: 515.123.8181
Hire date: 07-JUN-16
Job title: Public Accountant
```

   The procedure displays a proper error message if any error occurs.

3. Every year, the company increases the selling price of all products in one product type.
   For example, the company wants to increase the selling price of products in type Tents by $5.
   Write a procedure named *update_price_fortype* to update the selling price of all products in the given type and the given amount to be added to the current selling price if the price is greater than 0.
   The procedure shows the number of updated rows if the update is successful.
   The procedure gets two parameters:
   - Prod_type IN VARCHAR2
   - amount NUMBER(9,2) this means the amount to increase selling price by

   To define the type of variables that store values of a table column, you can also write:

```
variable_name table_name.column_name%type;
```

   The above statement defines a variable of the same type as the type of the table column.

```
Example: category_id     products.category_id%type;
```

   Or you need to see the table definition to find the type of the prod_type update column. Make sure the type of your variable is compatible with the value that is stored in your variable.

To show the number of affected rows the update query, declare a variable named `rows_updated` of type NUMBER and use the SQL variable `sql%rowcount` to set your variable. Then, print its value in your stored procedure. (Something like this is in the notes supplied to you)

```
Rows_updated := sql%rowcount;
```

`SQL%`*`ROWCOUNT`* stores the number of rows affected by an INSERT, UPDATE, or DELETE.

NOTE: Do not forget ROLLBACK;

4. Every year, the company increases the price of products by 2 to 5% (Example of 2% -- prod_sell * 1.02) based on if the selling price (prod_sell) is less than the average price of all products.
   Write a stored procedure named *update_low_prices_99 where 99 is replaced by your OracleID number. If you are dbs311_213d87 then the number to use will be 87*
   This procedure does not have any parameters. You need to find the average sell price of all products and store it into a variable of the same data type. If the average price is less than or equal to $1000, then update the products selling price by 2% if that products sell price is less than the calculated average.
   If the average price is greater than $1000, then update products selling price by 1% if the price of the products selling price is less than the calculated average.
   The query displays an error message if any error occurs. Otherwise, it displays the number of updated rows.

   An example of an output produced by your code might be the following or perhaps nicer

   ***** OUTPUT update_low_prices_99 STARTED *****

   Number of updates: 27

   ----ENDED --------

   NOTE: Do not forget ROLLBACK;

5. The company needs a report that shows three categories of products based their prices. The company needs to know if the product price is low, fair, or expensive. Let us assume that
   - If the list price is less than the (average sell price – minimum sell price) divided by 2
     The product's price is LOW.
   - If the list price is greater than the maximum less the average divided by 2
     The product' price is HIGH.
   - If the list price is between
     - (average price – minimum price) / 2 AND (maximum price – average price) / 2 INCLUSIVE

     The product's price is FAIR.

   Write a procedure named *price_report_99(you know what the 99 is replaced with)* to show the number of products in each price category:

   The following is a sample output of the procedure if no error occurs:

   ```
   Low:  10
   Fair: 50
   High: 18
   ```

   The values in the above examples are just random values and may not match the real numbers in your result.

The procedure has no parameter. First, you need to find the average, minimum, and maximum prices (list_price, selling price) in your database and store them into variables avg_price, min_price, and max_price.

You need three more variables to store the number of products in each price category:

low_count
fair_count
high_count

Make sure you choose a proper datatype for each variable. You may need to define more variables based on your solution.

NOTE: Do not forget ROLLBACK;

For multiple line comments you can do this
/* multiple comments
Multiple comments
*/

Example Submission

```
-- ***********************
-- Name: Your Name
-- Student ID: #########
-- Date: The current date
-- Purpose: Lab 5 DBS311
-- ***********************

-- Question 1 – write a brief note about what the question is asking
-- Q1 SOLUTION –

CREATE OR REPLACE procedure_name(arg1 data_type, ...) AS

BEGIN

  ....

EXCEPTION

     WHEN OTHERS

          THEN

               DBMS_OUTPUT.PUT_LINE (Error!');

     END procedure_name;

-- Question 2 – the question goes here

......

-- Q2 Solution – your code goes here

......
```