**Assignment 3**

**(Stored Procedures/Conditional Statements)**

Submission

***Your submission 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.***

Create a new Worksheet in SQL Developer. Save the file as A3\_Group#.sql

Use the same tables as in Assignment 1.

Your submission needs to be commented and include the question, the solutions.

**Group work acknowledgment**

We, ------------(mention your names), declare that the attached assignment is our own work in accordance with the Centennial Academic Policy. No part of this assignment has been copied manually or electronically from any other source (including web sites) **or distributed to other students.**

Specify below what each member has done towards the completion of this work:

Name Task(s)

1-

2-

3-

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 patameter
* 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

1. Write a store procedure that get an integer number and prints

*The number is even.*

If a number is divisible by 2.

Otherwise, it prints

*The number is odd.*

Do the testing as follows:

set SERVEROUTPUT ON

BEGIN

spEvenOdd(6);

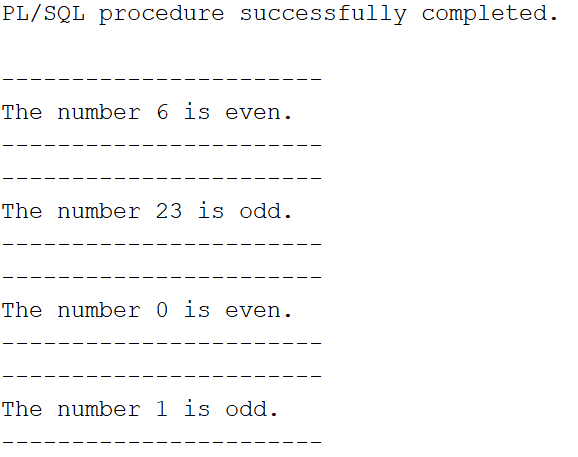
spEvenOdd(23);

spEvenOdd(0);

spEvenOdd(1);

END;

Output:



2. Create a stored procedure named *find\_employee*. This procedure gets an employee number and prints the following employee information:

First name

Last name

Hire date

Job title

Execute the following block to call the procedure

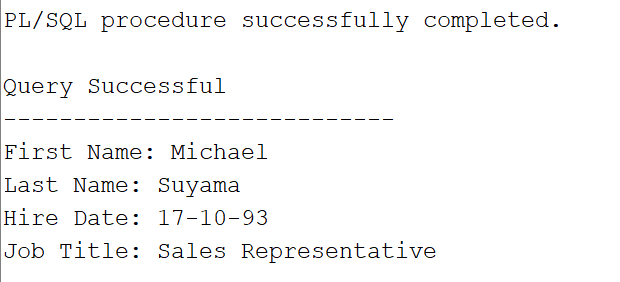
BEGIN

spFind\_Employee(6);

END;

The procedure gets a value as the employee ID of type NUMBER.

See the following example for employee ID 6:



The procedure display a proper error message if any error accours.

3. Every year, the company increases the price of all products in one category. For example, the company wants to increase the price (Unitprice) of products in category 1 by $5. Write a procedure named *update\_price\_by\_cat* to update the price of all products in a given category and the given amount to be added to the current 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:

* categoryid IN NUMBER
* amount NUMBER(9,2)

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

vriable\_name table\_name.column\_name%type;

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

category\_id products.categoryid%type;

Or you need to see the table definition to find the type of the categoryid 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.

Rows\_updated := sql%rowcount;

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

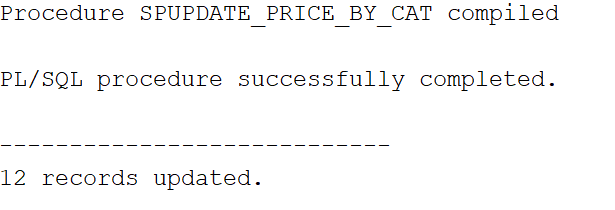
Call the procedure using following anonymous block:

BEGIN

spUpdate\_Price\_By\_Cat(2,1.00);

END;

Following is the expected output:



In order to undo the update in the products table. Execute rollback as follows.

rollback;

4. Every year, the company increase the price of products whose price is less than the average price of all products by 1%. (Unitprice \* 1.01). Write a stored procedure named *update\_price\_under\_avg*. This procedure do not have any parameters. You need to find the average price of all products and store it into a variable of the same type. If the average price is less than or equal to $1000, update products’ price by 2% if the price of the product is less than the calculated average. If the average price is greater than $1000, update products’ price by 1% if the price of the product is less than the calculated average. The query displays an error message if any error occurs. Otherwise, it displays the number of updated rows.

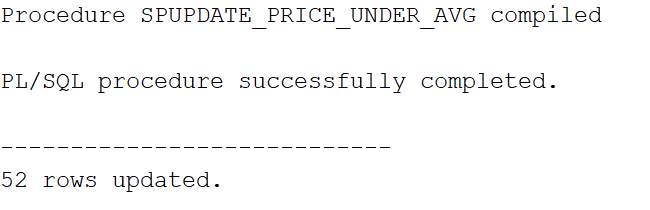
Execute the following anonymous block to call teh procedure:

BEGIN

spUpdate\_Price\_Under\_Avg();

END;

Following is the expected output:



In order to undo the update in the products table. Execute rollback as follows.

rollback;

5. The company needs a report that shows three category of products based their prices. The company needs to know if the product price is cheap, fair, or expensive. Let’s assume that

* If the Unit price is less than
* (avg\_price - min\_price) / 2

The product’s price is cheap.

* If the Unit price is greater than
* (max\_price - avg\_price) / 2

The product’ price is expensive.

* If the Unit price is between
* (avg\_price - min\_price) / 2
* and
* (max\_price - avg\_price) / 2
* the end values included

The product’s price is fair.

Write a procedure named *product\_price\_report* to show the number of products in each price category:

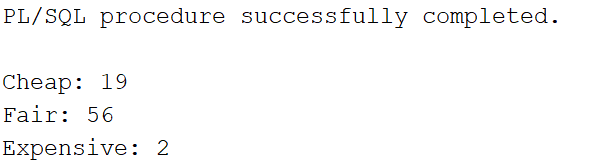
Call the procedure by executing following block:

begin

spProduct\_Price\_Report();

end;

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



The procedure has no parameter. First, you need to find the average, minimum, and maximum prices (Unitprice) in your database and store them into varibles avg\_price, min\_price, and max\_price.

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

cheap\_count  
fair\_count  
exp\_count

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

Example Submission

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
-- Name: Group Member Name  
-- ID: #########

-- Name: Group Member Name  
-- ID: #########

-- Name: Group Member Name  
-- ID: #########  
-- Date: The current date  
-- Purpose: Assignment 3  
-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- 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 –

-- Q2 Solution –

**Grading rubrics**

Each question is worth 10 pts. Total is 50 pts.

If the output is included without the query, the answer is worth 0.