# DBS311 Lab 6

Clear screenshots of successful run of PL/SQL statements (for each question regardless of supplied solutions) and output is required in a single file. Also, take screenshots of the procedure code.

Zero will be assigned otherwise. You may use SQL developer or SQL plus. If you want to be independent of mySeneca apps or Seneca Oracle instance, install Oracle XE in your laptop and use SQL plus.

1. Write a store procedure called *Even\_Odd* 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.*

**Show testing with one even and one odd integer.**

**SET SERVEROUTPUT ON;**

**SET PAGESIZE 120;**

**CREATE OR REPLACE PROCEDURE Even\_Odd (value IN INT) AS**

**BEGIN**

**IF MOD(value,2) =0 THEN**

**DBMS\_OUTPUT.PUT\_LINE ('The number is even');**

**ELSE**

**DBMS\_OUTPUT.PUT\_LINE ('The number is odd');**

**END IF;**

**END;**

**/**

**DECLARE**

**value INT :=100;**

**BEGIN**

**Even\_Odd(value);**

**END;**

**/**

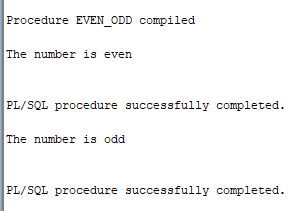
**DECLARE**

**value INT :=99;**

**BEGIN**

**Even\_Odd(value);**

**END;**

****

1. Create a stored procedure named *Find\_Employee*. This procedure gets an employee number and prints the following employee information:

First name

Last name

Email

Phone

Hire date

Job title

The procedure gets a value as the employeeID of type NUMBER.

See the following example for employeeID 107:

First name: Summer

Last name: Payn

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.

**Show testing with one invalid employee Id and one valid Id.**

**CREATE OR REPLACE PROCEDURE Find\_Employee (employeeID NUMBER) AS**

**firstName VARCHAR2(255 BYTE);**

**lastName VARCHAR2(255 BYTE);**

**email VARCHAR2(255 BYTE);**

**phone VARCHAR2(255 BYTE);**

**hairDate VARCHAR2(255 BYTE);**

**jobTitle VARCHAR2(255 BYTE);**

**BEGIN**

**SELECT first\_name, last\_name,email,phone\_number,hire\_date,job\_id INTO firstName, lastName, email,phone,hairDate,jobTitle**

**FROM employees**

**WHERE employee\_id = employeeID;**

**DBMS\_OUTPUT.PUT\_LINE ('First name: ' || firstName);**

**DBMS\_OUTPUT.PUT\_LINE ('Last name: ' || lastName);**

**DBMS\_OUTPUT.PUT\_LINE ('Email: ' || email);**

**DBMS\_OUTPUT.PUT\_LINE ('Phone: ' || phone);**

**DBMS\_OUTPUT.PUT\_LINE ('Hire date: ' || hairDate);**

**DBMS\_OUTPUT.PUT\_LINE ('Job title: ' || jobTitle);**

**EXCEPTION**

**WHEN OTHERS**

**THEN**

**DBMS\_OUTPUT.PUT\_LINE ('Error!');**

**END;**

**/**

**DECLARE**

**ID NUMBER :=107;**

**BEGIN**

**Find\_Employee(ID);**

**END;**

**/**

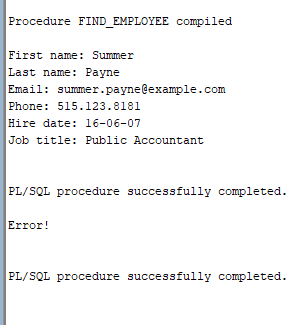
**DECLARE**

**ID NUMBER :=99999;**

**BEGIN**

**Find\_Employee(ID);**

**END;**

****

1. Every year, the company increases the price of all products in one category. For example, the company wants to increase the price (list\_price) 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 or shows 0 rows updated, if the input was an invalid category Id.

The procedure gets two parameters:

* category\_id 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.category\_id%type;

Or you need to see the table definition to find the type of the category\_id 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.

**Show testing with one invalid category Id and one valid Id.**

**Undo your Update > Rollback**

**CREATE OR REPLACE PROCEDURE Update\_Price\_by\_Cat (id products.category\_id%type,amount NUMBER) AS**

**Rows\_updated NUMBER;**

**BEGIN**

**UPDATE PRODUCTS SET List\_price = List\_price + amount**

**WHERE CATEGORY\_ID = id**

**AND LIST\_PRICE > 0;**

**Rows\_updated := sql%rowcount;**

**IF (Rows\_updated = 0) then**

**DBMS\_OUTPUT.PUT\_LINE ('Invalid Category ID');**

**ELSE**

**DBMS\_OUTPUT.PUT\_LINE ('The number of updated rows is: ' || Rows\_updated);**

**END IF;**

**EXCEPTION**

**WHEN OTHERS**

**THEN**

**DBMS\_OUTPUT.PUT\_LINE ('Error!');**

**END;**

**/**

**DECLARE**

**A NUMBER :=1;**

**B NUMBER :=5;**

**BEGIN**

**Update\_Price\_by\_Cat (A,B);**

**END;**

**/**

**DECLARE**

**A NUMBER :=99999;**

**B NUMBER :=5;**

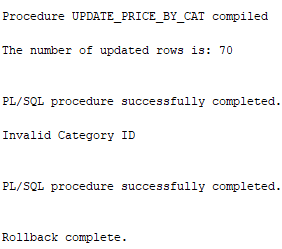
**BEGIN**

**Update\_Price\_by\_Cat (A,B);**

**END;**

**/**

**ROLLBACK;**

****

1. Every year, the company increase the price of products whose price is less than the average price of all products by 1%. (list\_price \* 1.01). Write a stored procedure named *Update\_Price\_Under\_Avg*. This procedure does 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.

**Show your testing.**

**Undo your Update > Rollback**

**CREATE OR REPLACE PROCEDURE Update\_Price\_Under\_Avg As**

**AvgPrice PRODUCTS.List\_price%type;**

**Rows\_updated NUMBER;**

**BEGIN**

**SELECT AVG(List\_price) INTO AvgPrice From PRODUCTS;**

**If AvgPrice <= 1000 THEN**

**UPDATE PRODUCTS SET List\_price = (List\_price \* 1.02)**

**WHERE List\_price < AvgPrice;**

**Rows\_updated := sql%rowcount;**

**ELSE**

**UPDATE PRODUCTS SET List\_price = (List\_price \* 1.01)**

**WHERE List\_price < AvgPrice;**

**Rows\_updated := sql%rowcount;**

**END IF;**

**DBMS\_OUTPUT.PUT\_LINE ('The number of updated rows is: ' || Rows\_updated);**

**EXCEPTION**

**WHEN OTHERS**

**THEN**

**DBMS\_OUTPUT.PUT\_LINE ('Error!');**

**END;**

**/**

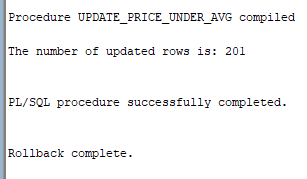
**BEGIN**

**Update\_Price\_Under\_Avg;**

**END;**

**/**

**ROLLBACK;**

****

1. 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 list price is less than
  + (avg\_price - min\_price) / 2

The product’s price is cheap.

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

The product’ price is expensive.

* If the list 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:

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

Cheap: 10

Fair: 50

Expensive: 18

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

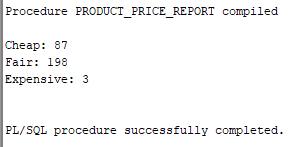
The procedure has no parameter. First, you need to find the average, minimum, and maximum prices (list\_price) in your database and store them into varibles avg\_price, min\_price, and max\_price.

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.

**Show your testing.**

****

**Note: Some of the above output displayed may not match exactly with your produced output. This is because the script file supplied to you was modified after creation of this lab requirements.**