# DBS311A Lab 5 *– E-Mail submission only, your document must have both CODE content and the OUTPUT after execution of the Procedure.*

# Subject of your mail must be like 311-Lab5 by Smith, John

# *Due by Saturday, March 13th by 7pm*

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.*

ANS

set serveroutput on

create or replace procedure Even\_Odd( the\_value in number)

is

begin

if mod(the\_value,2)=0

then

dbms\_output.put\_line('The number is even');

else

dbms\_output.put\_line('The number is odd');

end if;

end;

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

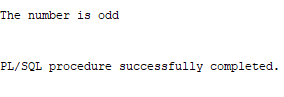
**Test with odd number**

**begin**

**Even\_Odd(13);**

**end;**

**output :**



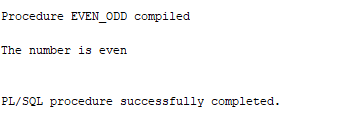
**Test with even number**

**begin**

**Even\_Odd(12);**

**end;**

**output:**



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 p\_empid of type NUMBER.

See the following example for employee# of 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.**

**ANS**

**set serveroutput on**

**create or replace procedure find\_employee (m\_value in number)**

**is**

**m\_first employees.first\_name%type;**

**m\_last employees.last\_name%type;**

**m\_email employees.email%type;**

**m\_phone employees.phone%type;**

**m\_hire employees.hire\_date%type;**

**m\_job employees.job\_title%type;**

**begin**

**select first\_name,last\_name,email,phone,hire\_date,job\_title**

**into m\_first,m\_last,m\_email,m\_phone,m\_hire,m\_job**

**from employees**

**where employee\_id = m\_value;**

**dbms\_output.put\_line('First\_name' || m\_first);**

**dbms\_output.put\_line('Last\_name' || m\_last);**

**dbms\_output.put\_line('Email' || m\_email);**

**dbms\_output.put\_line('Phone'|| m\_phone);**

**dbms\_output.put\_line('Hire Date' || m\_hire);**

**dbms\_output.put\_line('Job Title' || m\_job);**

**exception**

**when no\_data\_found**

**then**

**dbms\_output.put\_line('The Employee #id is not found ! ');**

**when others**

**then**

**dbms\_output.put\_line('Error occured');**

**end;**

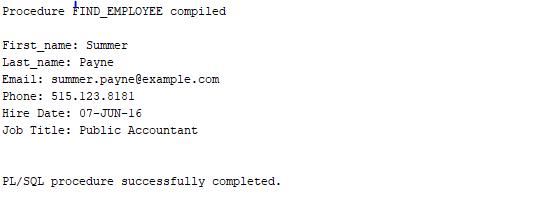
**when you pass in correct value :**

**begin**

**find\_employee(107);**

**end;**

**OUTPUT:**



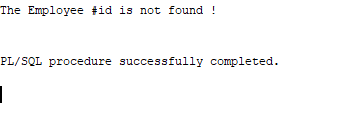
When incorect value is passed:

begin

find\_employee(895);

end;

Output



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:

* p\_catid IN NUMBER
* p\_amount IN NUMBER(9,2)

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.

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**

**ANS**

**create or replace procedure update\_price\_by\_cat (m\_category\_id IN products.category\_id%TYPE, m\_amount IN products.list\_price%TYPE)**

**as m\_value number;**

**BEGIN**

**SELECT COUNT(category\_id) INTO m\_value FROM products WHERE category\_id = m\_category\_id;**

**IF (m\_amount > 0 and m\_value > 0) THEN**

**UPDATE products SET LIST\_PRICE = LIST\_PRICE + m\_amount WHERE category\_id = m\_category\_id;**

**DBMS\_OUTPUT.PUT\_LINE('Rows updated :' || SQL%rowcount);**

**ELSE**

**DBMS\_OUTPUT.PUT\_LINE('The category entered does not exist or the price is less than zero');**

**END IF;**

**END;**

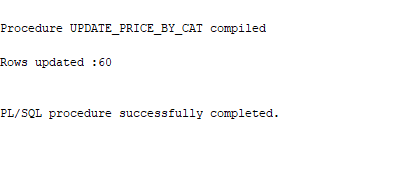
**Running with a valid data**

**BEGIN**

**update\_price\_by\_cat(4,5);**

**END;**

**Output:**



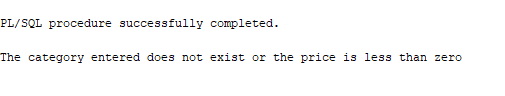
**Running with an invalid data**

**BEGIN**

**update\_price\_by\_cat(99,0);**

**END;**

**Output:**



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.

**ANS**

**set serveroutput on**

**create or replace procedure Update\_Price\_Under\_Avg**

**as m\_avg products.list\_price%TYPE;**

**m\_new number;**

**begin**

**select avg(list\_price) into m\_avg from products;**

**if m\_avg > 1000 then**

**m\_new := 1.01;**

**elsif m\_avg <= 1000 then**

**m\_new := 1.02;**

**end if;**

**update products set list\_price = list\_price \* m\_new where list\_price <= m\_avg;**

**dbms\_output.put\_line('Rows Updated =' || sql%rowcount);**

**EXCEPTION**

**when no\_data\_found then**

**dbms\_output.put\_line('No data found');**

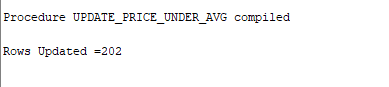
**when others then**

**dbms\_output.put\_line('There is a error in the code');**

**end;**

**Show your testing.**

**execute update\_price\_under\_avg;**



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
  + (ma x\_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.**

**ANS**

create or replace procedure Product\_price\_report

as avg\_pricess products.list\_price%Type;

avg\_price NUMBER;

max\_price NUMBER;

min\_price NUMBER;

cheap\_count NUMBER;

fair\_count NUMBER;

exp\_count NUMBER;

begin

select avg(list\_price), max(list\_price), min(list\_price) into avg\_price , max\_price , min\_price

from products;

select count(list\_price) into cheap\_count

from products where

list\_price < (avg\_price - min\_price)/2;

select count(list\_price) into exp\_count

from products where

list\_price > (max\_price - avg\_price) /2;

select count(list\_price) into fair\_count

from products where

list\_price between (avg\_price - min\_price) /2 and (max\_price - avg\_price) /2;

dbms\_output.put\_line('Cheap: '|| cheap\_count);

dbms\_output.put\_line('Fair: '|| fair\_count);

dbms\_output.put\_line('Expensive: '|| exp\_count);

EXCEPTION

when no\_data\_found then

dbms\_output.put\_line('No data found');

when others then

dbms\_output.put\_line('There is a error in the code');

end;

---------------------------------------------------

execute product\_price\_report;

output:

