COEN 178 **Intro to Database Systems** **Spring 2022**

Lab 5 (100 pts)

Objectives: Learn

- PLSQL Subprograms and cursors.
- SQL random number generator

Write PLSQL procedures to insert values for salary using a **random number generator** (this will come in handy if you want to generate "fake values" for demo purposes).

In this part, we will write a few PLSQL procedures (and a function), cursors and use a random number generator.

Exercise 1

We will write a simple procedure to display a message passed as a parameter.

```
Create or Replace Procedure DisplayMessage(message in VARCHAR)
As
BEGIN
     DBMS_OUTPUT.put_line('Hello '||message);

END;
/
Show Errors;
```

a) Create the procedure above. **You can do it in one of two ways**:

1) Copy and paste the code at SQL prompt.
2) Paste it into a text file (test.sql, for example) and give command **start test.sql** (must include the path name of the file) at the SQL prompt.

Q1) Did the procedure compile without errors?

b) At the sql prompt run the following:

```
SET SERVEROUTPUT ON
```

c) Now, at the SQL prompt, call the procedure with command,

```
exec DisplayMessage('include a message');
```

Q2) What is displayed?

Exercise 2

Let us try to generate a random integer in the range 10 to 100 in SQL.

Type the following query at SQL prompt.

```
Select ROUND(DBMS_RANDOM.value (10,100)) from DUAL;
```

Q3) What is displayed?

Exercise 3

For this exercise , you will use **AlphaCoEmp** table that you have created in the last lab. Please make sure that the table is still loaded with data (from last assignment). Otherwise, please use the same steps that you did last time, to load the data from Staff table.

Now, using an update SQL statement, make the salaries of all employees to 0.

We will write a PLSQL procedure to insert salaries (random values in a range) for all employees in AlphaCoEmp table.

Since we want to set the salaries of all employees in the table, we need to write the query which fetches all the employees. Therefore, we need to use a **cursor** to hold and point to the rows fetched. For each row fetched, we will generate a random number for a salary within the range (passed in as parameters) and update the table.

```
Create or Replace Procedure setSalaries(low in INTEGER, high in INTEGER)
As
Cursor Emp_cur IS
      Select * from AlphaCoEmp;
      -- Local variables
      l_emprec Emp_cur%rowtype;

      l_salary AlphaCoEmp.salary%type;
BEGIN
      for l_emprec IN Emp_cur
      loop
            l_salary := ROUND(dbms_random.value(low,high));

             update AlphaCoEmp
             set salary = l_salary
             where name = l_emprec.name;

   END LOOP;
   commit;
END;
/
show errors;
```

Create this procedure (you can copy and paste it at the prompt in console window)
Once you get a clean compilation of the procedure, execute it with salary values of your choice.

Example:

```
exec setSalaries (50000,100000);
```

Now, do a select * on AlphaCoEmp table. Do you see the salaries (where previously there was a 0 for this column).

Exercise 4

Now that you have salaries set in **AlphaCoEmp** table, write and run a SQL query to display the names of people with salaries between a low and a high value of your choice (between 80000 and 100000, for example).

Exercise 5

You must try writing a procedure on your own. It will be similar to the procedure **setSalaries(),** except now you need to be able to update the salary of the person (name passed as parameter) to a randomly generated value in a range (passed as a parameter). Complete the procedure below.

```
Create or Replace Procedure setEmpSalary(p_name in VARCHAR, low in
INTEGER, high in INTEGER)
As
     /* Define the local variables you need */


BEGIN
     /* since name is a primary key, set the salary
     Of the employee where name = p_name.

     With an update statement, set the salary of that employee
     With a randomly generated value between the low and high
passed
     In as parameters
     */


END;
/
show errors;
```

Test your procedure to change the salary of a name that you select from **AlphaCoEmp** table.

Exercise 6

In this exercise, you will **complete** a PLSQL function that returns the salary where name of the employee is passed in as a parameter. Remember, functions return something to the calling code whereas procedures are like void functions.

Complete the function below.

```
Create or Replace FUNCTION getEmpSalary(p_name in VARCHAR)

Return NUMBER IS

     /* Define the local variables you need.
     You need a variable to hold the salary returned    */

     l_salary  ;

BEGIN
     /* Two things are missing from the below statement (hint: how
are you saving the result into l_salary?)*/
     Select salary
     from AlphaCoEmp

     return l_salary;
END;
/
show errors;
```

Test your function using the query below. For the parameter, give an employee name of your choice that is in the AlphaCoEmp table.

```
Select getEmpSalary('')

From Dual;
```

Q4) Did your function work correctly?

Exercise 7

In this exercise, we will try to assign job titles to employees randomly selected from a hardcoded set of titles, in AlphaCoEmp table, using a **PLSQL procedure**. Since we want to assign salaries based on titles, we will store titles and salaries in two **PLSQL VArrays** and randomly select an index into the arrays and use the title and its associated salary from the same index number. For example, the salary for the title stored at index 1 in the **titles array** will be at index 1 in the **salaries array**.

Note: In PLSQL VArrays, subscripts start at 1, not 0.

Examine the code in the procedure below, to assign job titles and salaries.

```
Create or Replace Procedure assignJobTitlesAndSalaries
As
      type titlesList IS VARRAY(5) OF AlphaCoEmp.title%type;
      type salaryList IS VARRAY(5) of AlphaCoEmp.salary%type;
      v_titles titlesList;
      v_salaries salaryList;
Cursor Emp_cur IS
      Select * from AlphaCoEmp;
      l_emprec Emp_cur%rowtype;
      l_title AlphaCoEmp.title%type;
      l_salary AlphaCoEmp.salary%type;
      l_randomnumber INTEGER := 1;
```

```
BEGIN
/* Titles are stored in the v_titles array  */
/* Salaries for each title are stored in the v_salaries array.
The salary of v_titles[0] title is at v_salaries[0].
*/
v_titles := titlesList('advisor', 'director', 'assistant', 'manager',
'supervisor');

v_salaries := salaryList(130000, 100000, 600000, 500000, 800000);

/* use a for loop to iterate through the set  */
for l_emprec IN Emp_cur
LOOP
     /* We get a random number between 1-5 both inclusive */
     l_randomnumber := dbms_random.value(1,5);

     /* Get the title using the random value as the index into the
     v_tiles array */
     l_title := v_titles(l_randomnumber);
     /* Get the salary using the same random value as the index into
     the v_salaries array */
     l_salary := v_salaries(l_randomnumber);

     /* Update the employee title and salary using the l_title
     and l_salary */
     update AlphaCoEmp
     set title = l_title
     where name = l_emprec.name;

     update AlphaCoEmp
     set salary = l_salary
     where name = l_emprec.name;
```

```
END LOOP;

commit;
END;
/
Show errors;
```

Run the code and if it compiles without errors, run the command,

a) exec assignJobTitlesAndSalaries at SQL prompt.

b) Run a Select * on AlphaCoEmp table and check if titles and salaries are assigned.

c) Now, modify the above procedure and include one more job title and a salary for the title in the code.

d) Run the procedure.

e) Execute the procedure and make sure it is working ok.

Exercise 8

Now, we will write a **PLSQL function** that **calculates the salary raise based on the current salary and the percent raise**. PLSQL functions return a value.

The function **calcSalaryRaise()** calculates raise amount as follows:

- • Takes the name and percent salary (an integer) as parameters.
- • Fetches the employee's salary from AlphaCoEmp Table and calculates the amount of raise.
- • If that employee is also is in Emp_Work table, adds an additional 1000 to the raise.

```
Create or Replace Function calcSalaryRaise( p_name in
AlphaCoEmp.name%type, percentRaise IN NUMBER)
RETURN NUMBER
IS
      l_salary AlphaCoEmp.salary%type;
      l_raise AlphaCoEmp.salary%type;
      l_cnt Integer;
BEGIN
      -- Find the current salary of p_name from AlphaCoEMP table.
      Select salary into l_salary from AlphaCoEmp
      where name = p_name;
      -- Calculate the raise amount
      l_raise := l_salary * (percentRaise/100);
```

```
        -- Check if the p_name is in Emp_Work table.
        -- If so, add a $1000 bonus to the
        -- raise amount
        Select count(*) into l_cnt from Emp_Work
        where name = p_name;
        if l_cnt >= 1 THEN
        l_raise := l_raise + 1000;
End IF;
        /* return a value from the function */
        return l_raise;

END;
/
Show Errors;
```

Run the function (copy and paste it at SQL prompt or run it from a script file). Once it compiles without errors, you can execute it by calling it, as shown below.

a) **If you want to test the function and see if it is working ok, call it as follows:**

```
Select calcSalaryRaise('Stone',2) from Dual;
```

Note: You can give any name that is in the AlphaCoEmp table.

Q5) What is the output? ----------

b) Call the function as part of a more useful SQL query

```
Select name, title, salary CURRENTSALARY, (salary +
trunc(calcSalaryRaise(name,2))) NEWSALARY
from AlphaCoEmp where upper(name) = upper('Stone');
```

Q6) What is the output?----------

c) If you examine the code of the function, we are comparing (string compare) the name with the parameter, p_name without checking the case. Modify the code so that both strings are compared with each other, regardless of either's casing.

Test and make sure your function work correctly after modifications.

Exercise 9

a) Let us create a table called **EmpStats** as follows:

```
Create table EmpStats (title VARCHAR(20) Primary KEY,empcount INTEGER,
lastModified DATE);
```

b) The function (incomplete) below, **counts the number of employees from AlphaCoEmp table by title, where title is passed as a parameter and returns the count**.

Complete the function and run it.

```
Create or Replace Function countByTitle(p_title in
AlphaCoEmp.title%type)
RETURN NUMBER IS
      l_cnt Integer;
BEGIN
      /* Complete the query below */
      Select into l_cnt from AlphaCoEmp
      Group by
      Having
      return l_cnt;
END;
/
```

c) Run the SQL commands below and <u>show the output</u>.

```
select countByTitle('director') from Dual;
```

```
select countByTitle('advisor') from Dual;
```

Exercise 10

If the function, **countByTitle()** in Exercise 9 is working ok, we will write a procedure to store the number of employees by title in the table, EmpStats. We will call the function countByTitle() in this procedure.

```
CREATE or REPLACE procedure saveCountByTitle
AS
l_advisor_cnt integer := 0;
BEGIN
l_advisor_cnt := countByTitle('advisor');

delete from EmpStats; -- Any previously loaded data is deleted
/* inserting count of employees with title, 'advisor'.*/
insert into EmpStats values ('advisor',l_advisor_cnt,SYSDATE);
END;
/
Show errors;
```

The above procedure stores the count of employees with the title, 'advisor'.

a) Complete the procedure to store the count of employees for every title you have in AlphaCoEmp table.

b) Execute the procedure.

c) Show the data in the EmpStats table (do a Select *).

Q7) What is the count shown for the title, 'advisor'? ----------

Run the queries and capture the results in **lab5_output.log**, using *spool*.