COEN 178 Intro to Database Systems Winter 2019

Lab 4 (100 pts)

Objectives: Learn

- SQL Integrity Constraints
- PLSQL Procedures and cursors.
- SQL random number generator

In this lab, we will create two new tables and load them with data from **staff_2010** table that you have created in the earlier labs.

The following is the plan of what you will do today.

a) Create two new tables with foreign key constraints

b) Load the new tables with a subset of data from staff_2010 table (we will avoid the tedium of having to load them by hand with hardcoded values)

c) Test the referential integrity enforcement.

d) 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).

Part 1

Question 1

Create a table, **AlphaCoEmp** as follows.

```
CREATE TABLE AlphaCoEmp(Name VARCHAR(25) Primary Key, Title VARCHAR(20)
DEFAULT NULL, Salary Number(10,2) DEFAULT 0);
```

Inserting values:

We will load the table, AlphaCoEmp, with employee last names from Staff_2010.

Now run the query below to insert the last names from table, **Staff_2010**.

```
INSERT INTO AlphaCoEmp (name) SELECT last from Staff_2010;
```

Did you see any errors? The errors are because of the primary key constraint which is being violated by some duplicate last names in the Staff_2010 table. How do you fix this query so that it loads unique last names from Staff_2010?

(**Hint**: Think **distinct**, modify and rerun the query)

Did it run now? How many rows are created?

Do a Select * from AlphaCoEmp and check the results displayed. What was displayed for title and salary?

Question 2

Let us define a table called **Emp_Work** as follows:

```
Create Table Emp_Work(name VARCHAR(25) Primary Key, Project
VARCHAR(20) default NULL,

Constraint FK_AlphaCo

Foreign Key (name) REFERENCES AlphaCoEmp(name));
```

Note the foreign key constraint defined in this table with a name.

Let us insert employee names that start with **A** or **G** or **S** from the **AlphaCoEmp** table into **Emp_Work** table. We will use regular expressions to grab names that start with A or G or S. (You are free to use your choice for letters).

```
insert into Emp_Work(name) Select Name from AlphaCoEmp where

REGEXP_LIKE(name,'(^[ags])','i');
```

Study the Regex that describes a pattern where the string starts with (^) a [ags] (a or g or s).

a) **What is the 'i' for?**

Run the query.

b) **How many rows are inserted into Emp_Work table?**

Now, run a query to display the names that start with a (or A) or g(or G) or s (or S) from Emp_Work table.

```
Select Name from AlphaCoEmp where REGEXP_LIKE(name,'(^[ags])','i');
```

Pick a name from the displayed list and delete it from the table, AlphaCoEmp.

```
Example:

Delete from AlphaCoEmp

Where name='Smith';
```

c) **Did your deletion work? Explain why it did not work.**

Question 3

Let us change the **Emp_Work** table definition such that when a primary key that has a reference from a foreign key is deleted, it should delete the row with that foreign key as well.

We will use the **alter Table** command and define the constraint, on **delete cascade**.

Remember how you defined this table earlier.

Create Table Emp_Work(name VARCHAR(20) Primary Key, Project VARCHAR(20) default NULL,

Constraint FK_AlphaCo

Foreign Key (name) REFERENCES AlphaCoEmp(Name));

Now, we need to drop this constraint **FK_AlphaCo** and add the constraint with delete cascade. We do dropping and adding the new constraint with Alter Table statement.

Run this statement to drop the constraint.

```
Alter table Emp_Work

drop constraint FK_AlphaCo;
```

Add the new constraint as follows:

```
Alter table Emp_Work

add constraint FK_AlphaCo

FOREIGN KEY (name)

references AlphaCoEmp(name)

on delete cascade;
```

a) **Is the table altered?**

Now try to delete the name you tried earlier, from AlphaCoEmp table.

Example:

```
    DELETE from AlphaCoEmp

    Where name='Smith';
```

b) Did you succeed this time?

c) Check if the name “Smith” is in the Emp_Work table. It should have been deleted automatically if our constraint worked. Did it work?

Part 2

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

Question 4

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;
```

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

a) Copy and paste the code at SQL prompt.

b) 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.

----

a) **Did the procedure compile without errors?**

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

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

c) **What is displayed?**

Question 5

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;
```

a) **What is displayed?**

We will write a procedure to insert salaries (random values in a range) for all employees in AlphaCoEmp table. Remember we have loaded a default value of 0 for salary, when we created the 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).

Question 6

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).

Question 7

Now 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, select 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
     */



   commit;
END;
/
show errors;
```

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

Question 8 **(If you are not able to complete this exercise and run it during this lab time, please complete it and submit it before the start of the next lab).**

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 where as 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
     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;
```

Did your function work correctly?

Run the queries and capture the results in **lab4_output.txt**, using *spool*.