# Lab 6 - Stored Procedures/Iterative Statements

In this Lab, learners will create PL/SQL stored procedures and/or user defined functions to perform a set of given tasks. The idea of this lab is to create object-oriented objects to provide answers and/or output to software. This will include both objects that directly supply output, but also objects that perform tasks that support other objects that provide the output.

## Submission

* Your submission will be a single SQL file with appropriate header and commenting. Please ensure your file runs when the entire file is executed in SQL Developer.
* All objects must include exception handling where appropriate, and specifically the type of exceptions, and only those types, that may occur within the object coding.
* All coding must adhere to the posted course style guide

## Notes

A parameter can be

* IN parameter
* 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 OTHERS exception checked just in case any error occurs that has not been anticipated at the time you write the code.

Do not forget to run SET SERVEROUTPUT ON; before each coding session.

## Tasks

1. Write a user-defined function, called *fncCalcFactorial*, that gets an integer number *n* and calculates and returns its factorial.   
   **Example**:  
   0! = 1  
   2! = fact(2) = 2 \* 1 = 2  
   3! = fact(3) = 3 \* 2 \* 1 = 6

4! = fact(4) = 4 \* 3 \* 2 \* 1 = 24  
. . .  
n! = fact(n) = n \* (n-1) \* (n-2) \* . . . \* 1  
  
Create a non-saved procedure that executes the above function and outputs the result for 3 different input values.

1. The company wants to calculate the employees’ annual salary (not pay). For this question we will state that the salary in the database is the initial salary of the employee, not the current salary. Here are the business rules:  
    - The first year of employment, the amount of salary is the amount found in the salary column.  
    - Every year after that, on their hiring anniversary, salary is increased by 4%.  
    - Employees start their job with 2 weeks vacation annually, after 3 years they earn an additional week each year to a maximum of 6 weeks.  
     
   Write a stored procedure named *spCalcCurrentSalary* which gets an employee ID and for that employee calculates the current salary based on the number of FULL years the employee has been working in the company. (Use a loop construct to calculate the salary) and the number of weeks vacation they will receive.  
   The procedure calculates and prints the salary.

Sample output:  
 First Name: first\_name   
 Last Name: last\_name  
 Hire Date: Nov. 1, 2017  
 Salary: $9999,99  
 Vacation Weeks: 3

If the employee does not exist, the procedure should display an appropriate message, use implicate cursor attributes to determine this.  
Create a non-saved procedure that executes the above stored procedure and outputs the result for 3 different input values.

1. Write a stored procedure named *spDepartmentsReport* to print the department ID, department name, the city where the department is located, and the number of employees working in the department. Print the output to the Script Output window in the following format (data incorrect in sample):  
     
   DeptID Department City NumEmp  
    10 Adminstration Seattle 4  
    120 Accounting Toronto 2  
    …   
   Include ALL departments, even if there are no employees, and show the output as 0 for those with no department (not null).  
   Create a non-saved procedure that executes the above stored procedure and outputs the results as specified.
2. Using the SportLeagues tables:
   1. Create a user defined function, called *spDetermineWinningTeam* that receives a gameID and determines the teamID of the winning team.  
      - if the game has not yet been played, return -1  
      - if the game is a tie, return 0
   2. Create an SQL statement that outputs ALL teams and the total number of games they have won using the UDF created in part A.   
      - Note you will need aggregate functions here!  
      - How are you going to handle the ties and games not played yet??? I am curious to see what you guys come up with here!!!!