# ITE 5220 Oracle Database Programming using PL/SQL

## Lab Exercise 5[Chapter 7 & 8]

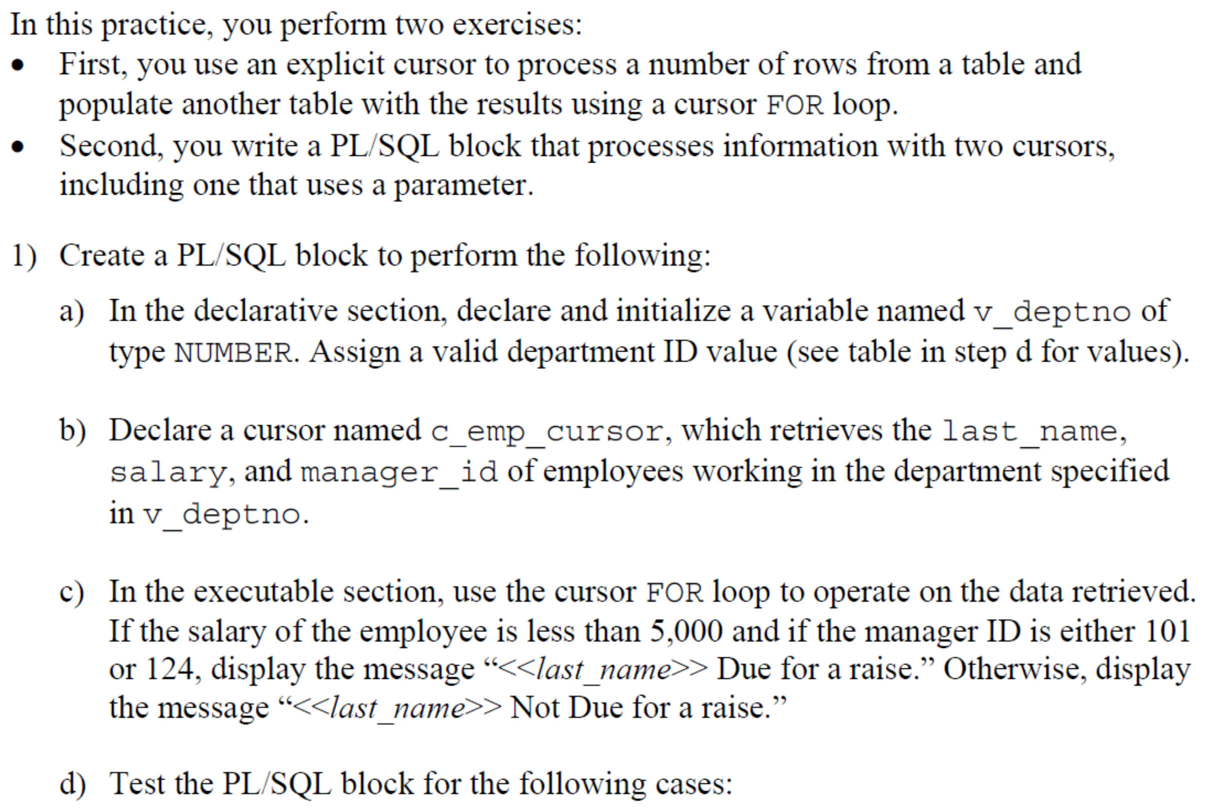
## 8 POINTS

**Agenda:**

**To do this lab you will have to use your laptops.**

**You have to capture the output and write your findings about the output.**

**Practice 1:[3 Points]**
**Using Explicit Cursors**



In this practice, you perform two exercises:

- First, you use an explicit cursor to process a number of rows from a table and populate another table with the results using a cursor `FOR` loop.
- Second, you write a PL/SQL block that processes information with two cursors, including one that uses a parameter.

1) Create a PL/SQL block to perform the following:

   a) In the declarative section, declare and initialize a variable named `v_deptno` of type `NUMBER`. Assign a valid department ID value (see table in step d for values).

   b) Declare a cursor named `c_emp_cursor`, which retrieves the `last_name`, `salary`, and `manager_id` of employees working in the department specified in `v_deptno`.

   c) In the executable section, use the cursor `FOR` loop to operate on the data retrieved. If the salary of the employee is less than 5,000 and if the manager ID is either 101 or 124, display the message "<<*last_name*>> Due for a raise." Otherwise, display the message "<<*last_name*>> Not Due for a raise."

   d) Test the PL/SQL block for the following cases:

| Department ID | Message |
|---|---|
| 10 | Whalen Due for a raise |
| 20 | Hartstein Not Due for a raise<br>Fay Not Due for a raise |
| 50 | Weiss Not Due for a raise<br>Fripp Not Due for a raise<br>Kaufling Not Due for a raise<br>Vollman Not Due for a raise. . .<br>. . .<br>OConnell Due for a raise<br>Grant Due for a raise |
| 80 | Russell Not Due for a raise<br>Partners Not Due for a raise<br>Errazuriz Not Due for a raise<br>Cambrault Not Due for a raise<br>. . .<br>Livingston Not Due for a raise<br>Johnson Not Due for a raise |

2) Next, write a PL/SQL block that declares and uses two cursors—one without a parameter and one with a parameter. The first cursor retrieves the department number and the department name from the `departments` table for all departments whose ID number is less than 100. The second cursor receives the department number as a parameter, and retrieves employee details for those who work in that department and whose `employee_id` is less than 120.

   a) Declare a cursor `c_dept_cursor` to retrieve `department_id` and `department_name` for those departments with `department_id` less than 100. Order by `department_id`.

   b) Declare another cursor `c_emp_cursor` that takes the department number as parameter and retrieves the following data from the `employees` table: `last_name`, `job_id`, `hire_date`, and `salary` of those employees who work in that department, with `employee_id` less than 120.

   c) Declare variables to hold the values retrieved from each cursor. Use the `%TYPE` attribute while declaring variables.

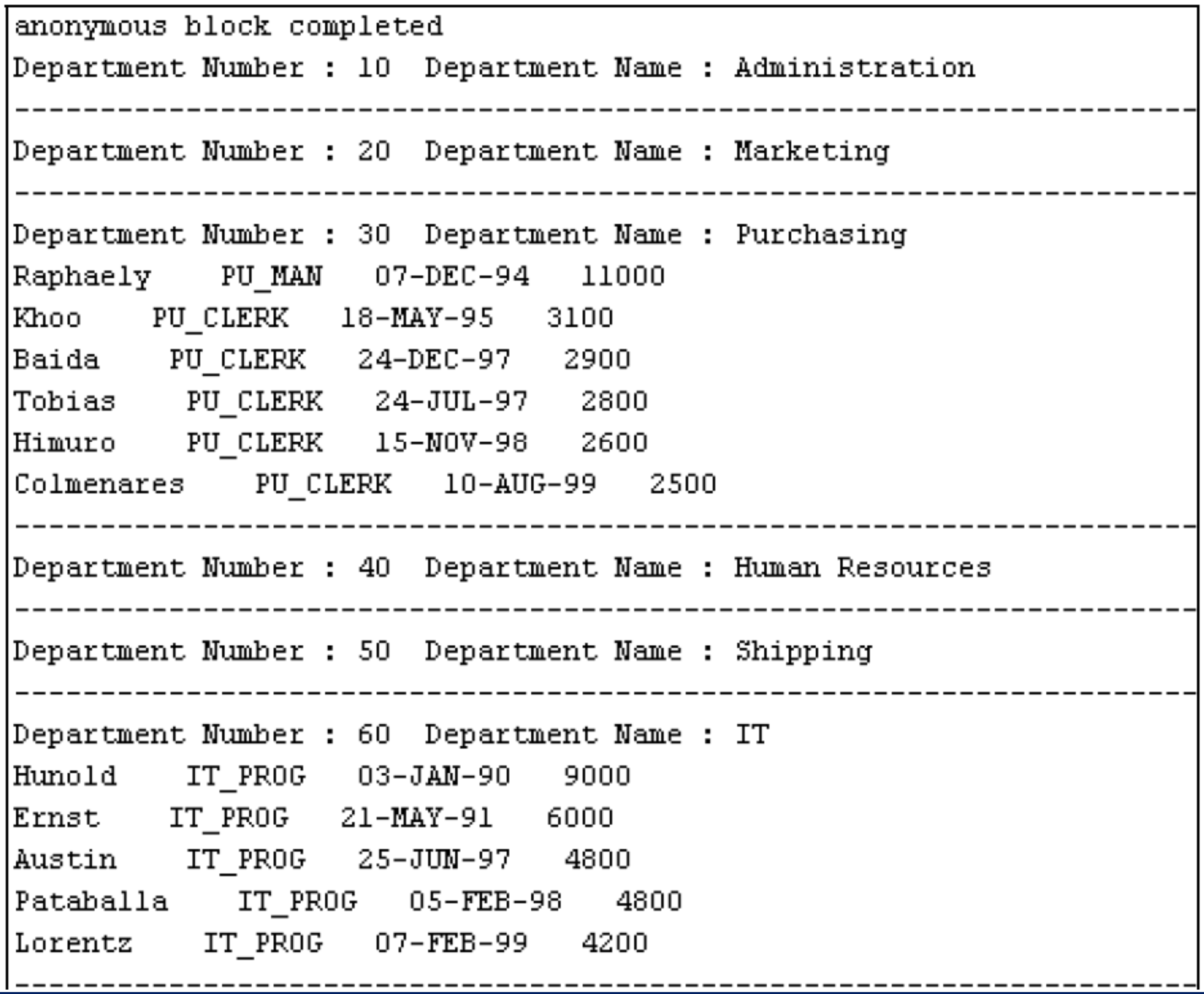
   d) Open `c_dept_cursor` and use a simple loop to fetch values into the variables declared. Display the department number and department name. Use the appropriate cursor attribute to exit the loop.

   e) Open `c_emp_cursor` by passing the current department number as a parameter. Start another loop and fetch the values of `emp_cursor` into variables, and print all the details retrieved from the `employees` table.

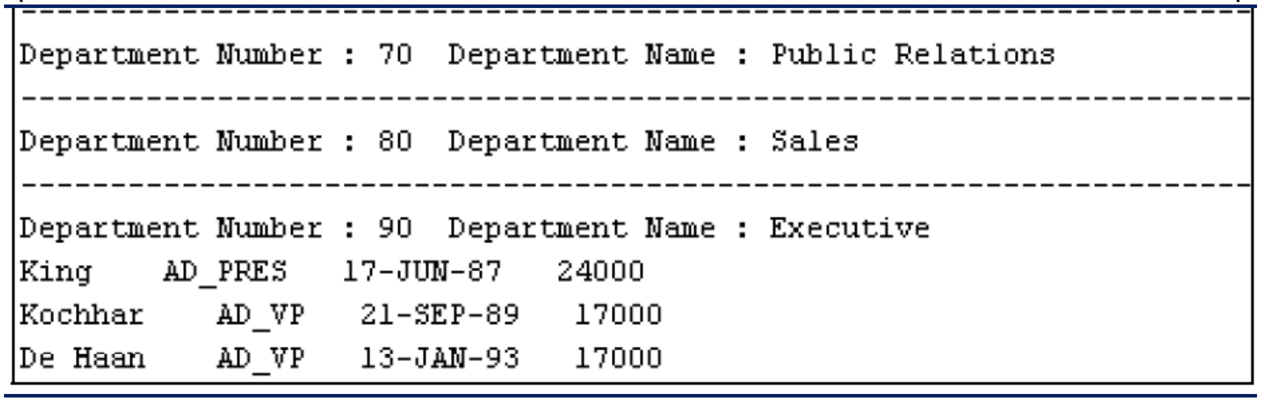
      **Note**
      - Check whether `c_emp_cursor` is already open before opening the cursor.
      - Use the appropriate cursor attribute for the exit condition.
      - When the loop completes, print a line after you have displayed the details of each department, and close `c_emp_cursor`.

   f) End the first loop and close `c_dept_cursor`. Then end the executable section.

   g) Execute the script. The sample output is as follows:



```
anonymous block completed
Department Number : 10  Department Name : Administration
---------------------------------------------------------------------
Department Number : 20  Department Name : Marketing
---------------------------------------------------------------------
Department Number : 30  Department Name : Purchasing
Raphaely    PU_MAN   07-DEC-94   11000
Khoo    PU_CLERK   18-MAY-95   3100
Baida    PU_CLERK   24-DEC-97   2900
Tobias    PU_CLERK   24-JUL-97   2800
Himuro    PU_CLERK   15-NOV-98   2600
Colmenares    PU_CLERK   10-AUG-99   2500
---------------------------------------------------------------------
Department Number : 40  Department Name : Human Resources
---------------------------------------------------------------------
Department Number : 50  Department Name : Shipping
---------------------------------------------------------------------
Department Number : 60  Department Name : IT
Hunold    IT_PROG   03-JAN-90   9000
Ernst    IT_PROG   21-MAY-91   6000
Austin    IT_PROG   25-JUN-97   4800
Pataballa    IT_PROG   05-FEB-98   4800
Lorentz    IT_PROG   07-FEB-99   4200
---------------------------------------------------------------------
```



```
---------------------------------------------------------------------
Department Number : 70  Department Name : Public Relations
---------------------------------------------------------------------
Department Number : 80  Department Name : Sales
---------------------------------------------------------------------
Department Number : 90  Department Name : Executive
King    AD_PRES   17-JUN-87   24000
Kochhar    AD_VP   21-SEP-89   17000
De Haan    AD_VP   13-JAN-93   17000
```

## Practice 2:[3 Points]

Create a PL/SQL block that uses an explicit cursor to determine the top *n* salaries of employees.

1) Run the `lab_07-2.sql` script to create the `top_salaries` table for storing the salaries of the employees.

2) In the declarative section, declare the `v_num` variable of the `NUMBER` type that holds a number `n`, representing the number of top *n* earners from the `employees` table. For example, to view the top five salaries, enter `5`. Declare another variable `sal` of type `employees.salary`. Declare a cursor, `c_emp_cursor`, which retrieves the salaries of employees in descending order. Remember that the salaries should not be duplicated.

3) In the executable section, open the loop and fetch the top *n* salaries, and then insert them into the `top_salaries` table. You can use a simple loop to operate on the data. Also, try and use the `%ROWCOUNT` and `%FOUND` attributes for the exit condition.

   **Note:** Make sure that you add an exit condition to avoid having an infinite loop.

4) After inserting data into the `top_salaries` table, display the rows with a `SELECT` statement. The output shown represents the five highest salaries in the `employees` table.

```
SALARY
----------------------
24000
17000
17000
14000
13500
```

5) Test a variety of special cases such as `v_num = 0` or where `v_num` is greater than the number of employees in the `employees` table. Empty the `top_salaries` table after each test.

## Practice 3:[2 Points]
## Handling Predefined Exceptions

In this practice, you write a PL/SQL block that applies a predefined exception in order to process only one record at a time. The PL/SQL block selects the name of the employee with a given salary value.

1) Execute the command in the `lab_05_01.sql` file to re-create the `messages` table.

2) In the declarative section, declare two variables: `v_ename` of type `employees.last_name` and `v_emp_sal` of type `employees.salary`. Initialize the latter to 6000.

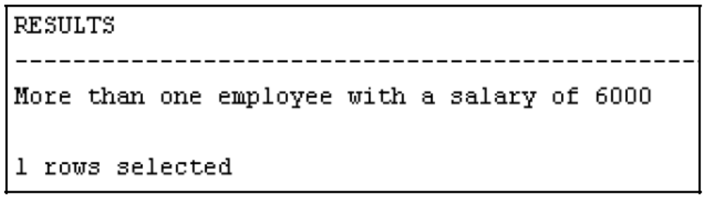
3) In the executable section, retrieve the last names of employees whose salaries are equal to the value in `v_emp_sal`. If the salary entered returns only one row, insert into the `messages` table the employee's name and the salary amount.
   **Note:** Do not use explicit cursors.

4) If the salary entered does not return any rows, handle the exception with an appropriate exception handler and insert into the `messages` table the message "No employee with a salary of <*salary*>."

5) If the salary entered returns multiple rows, handle the exception with an appropriate exception handler and insert into the `messages` table the message "More than one employee with a salary of <*salary*>."

6) Handle any other exception with an appropriate exception handler and insert into the `messages` table the message “Some other error occurred.”

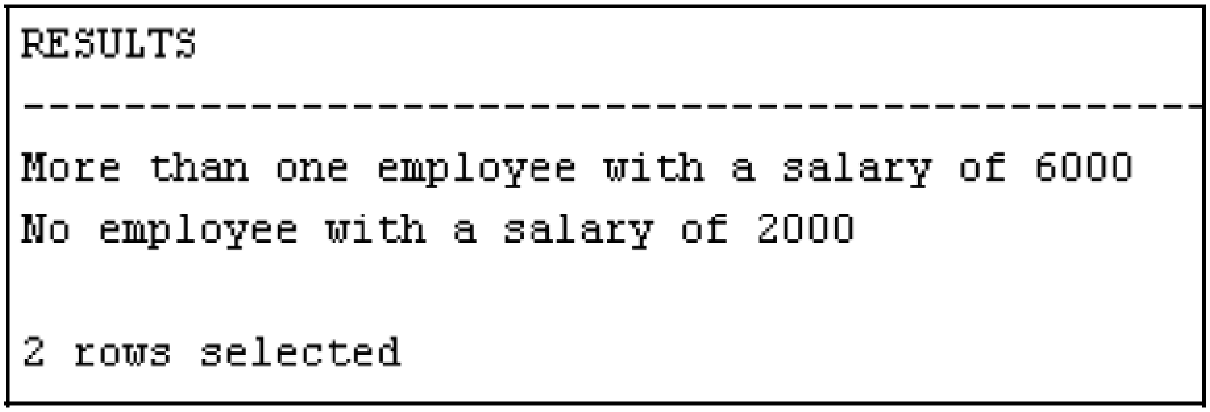
7) Display the rows from the `messages` table to check whether the PL/SQL block has executed successfully. The output is as follows:



```
RESULTS
-----------------------------------------------
More than one employee with a salary of 6000

1 rows selected
```

8) Change the initialized value of `v_emp_sal` to 2000 and re-execute. Output is as follows:



```
RESULTS
-----------------------------------------------
More than one employee with a salary of 6000
No employee with a salary of 2000

2 rows selected
```

## Practice 4: [2 Points]
## Handling Standard Oracle Server Exceptions

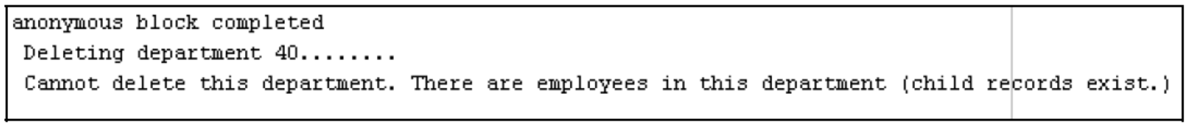
In this practice, you write a PL/SQL block that declares an exception for the Oracle Server error `ORA-02292 (integrity constraint violated – child record found)`. The block tests for the exception and outputs the error message.

1) In the declarative section, declare an exception `e_childrecord_exists`. Associate the declared exception with the standard Oracle Server error `–02292`.

2) In the executable section, display "Deleting department 40...." Include a `DELETE` statement to delete the department with the `department_id` 40.

3) Include an exception section to handle the `e_childrecord_exists` exception and display the appropriate message.

   The sample output is as follows:



```
anonymous block completed
 Deleting department 40........
 Cannot delete this department. There are employees in this department (child records exist.)
```