PLSQL

Document Revision History

Date	Revision No.	Author	Summary of Changes
July 2016	1.0	Shraddha Jadhav and Rahul Kulkarni	Integration refinements as per integrated RDBMS LOT Structure

Table of Contents

Getting Started

Overview

This lab book is a guided tour for learning Oracle 9i. It comprises ‘To Do’ assignments. Follow the steps provided and work out the ‘To Do’ assignments.

Setup Checklist for Oracle 9i

Here is what is expected on your machine in order for the lab to work.

Minimum System Requirements

- Intel Pentium 90 or higher (P166 recommended)
- Microsoft Windows 95, 98, or NT 4.0, 2k, XP.
- Memory: 32MB of RAM (64MB or more recommended)

Please ensure that the following is done:

- Oracle Client is installed on every machine
- Connectivity to Oracle Server

Instructions

- For all coding standards refer Appendix A. All lab assignments should refer coding standards.
- Create a directory by your name in drive <drive>. In this directory, create a subdirectory Oracle 9i_assgn. For each lab exercise create a directory as lab <lab number>.

Learning More (Bibliography if applicable)

- Oracle10g - SQL - Student Guide - Volume 1 by Oracle Press
- Oracle10g - SQL - Student Guide - Volume 2 by Oracle Press
- Oracle10g database administration fundamentals volume 1 by Oracle Press
- Oracle10g Complete Reference by Oracle Press
- Oracle10g SQL with an Introduction to PL/SQL by Lannes L. Morris-Murphy

Problem Statement / Case Study

Table descriptions

Emp

Name	Null?	Type
Empno	Not Null	Number(4)
Ename	-	Varchar2(10)
job		Varchar2(9)
mgr		Number(4)
Hiredate		Date
Sal	-	Number(7,2)
Comm	-	Number(7,2)
Deptno	-	Number(2)

Designation_Masters

Name	Null?	Type
Design_code	Not Null	Number(3)
Design_name		Varchar2(50)

Department_Masters

Name	Null?	Type
Dept_Code	Not Null	Number(2)
Dept_name		Varchar2(50)

Student_Masters

Name	Null?	Type
Student_Code	Not Null	Number(6)
Student_name	Not Null	Varchar2(50)
Dept_Code		Number(2)
Student_dob		Date
Student_Address		Varchar2(240)

Student_Marks

Name	Null?	Type
Student_Code		Number(6)
Student_Year	Not Null	Number
Subject1		Number(3)
Subject2		Number(3)
Subject3		Number(3)

Staff_Masters

Name	Null?	Type
Staff_code	Not Null	Number(8)
Staff_Name	Not Null	Varchar2(50)
Design_code		Number
Dept_code		Number
HireDate		Date
Staff_dob		Date
Staff_address		Varchar2(240)
Mgr_code		Number(8)
Staff_sal		Number (10,2)

Book_Masters

Name	Null?	Type
Book_Code	Not Null	Number(10)
Book_Name	Not Null	Varchar2(50)
Book_pub_year		Number
Book_pub_author	Not Null	Varchar2(50)

Book_Transactions

Name	Null?	Type
Book_Code		Number
Student_code		Number
Staff_code		Number
Book_Issue_date	Not Null	Date
Book_expected_return_date	Not Null	Date
Book_actual_return_date		Date

Lab 1. Introduction to PL/SQL and Cursors

Goals	The following set of exercises are designed to implement the following • PL/SQL variables and data types • Create, Compile and Run anonymous PL/SQL blocks • Usage of Cursors
Time	1hr 30 min

2.1

Identify the problems(if any) in the below declarations:

```
DECLARE
V_Sample1 NUMBER(2);
V_Sample2 CONSTANT NUMBER(2) ;
V_Sample3 NUMBER(2) NOT NULL ;
V_Sample4 NUMBER(2) := 50;
V_Sample5 NUMBER(2) DEFAULT 25;
```

Example 1: Declaration Block

2.2
The following PL/SQL block is incomplete.
Modify the block to achieve requirements as stated in the comments in the block.

```
DECLARE --outer block
var_num1 NUMBER := 5;
BEGIN

DECLARE --inner block
var_num1 NUMBER := 10;
BEGIN
DBMS_OUTPUT.PUT_LINE('Value for var_num1:' ||var_num1);
--Can outer block variable (var_num1) be printed here.If Yes,Print the same.
END;
--Can inner block variable(var_num1)  be printed here.If Yes,Print the same.
END;
```

Example 2: PL/SQL block

2.3. Write a PL/SQL block to retrieve all staff (code, name, salary) under specific department number and display the result. (Note: The Department_Code will be accepted from user. Cursor to be used.**)**

2.4. Write a PL/SQL block to increase the salary by 30 % or 5000 whichever minimum for a given Department_Code.

2.5. Write a PL/SQL block to generate the following report for a given Department code

Student_Code Sudent_Name Subject1 Subject2 Subject3 Total Percentage Grade

Note: Display suitable error massage if wrong department code has entered and if there is no student in the given department.

For Grade:
Student should pass in each subject individually (pass marks 60).
Percent >= 80 then grade= A
Percent >= 70 and < 80 then grade= B
Percent >= 60 and < 70 then grade= C
Else D

2.6. Write a PL/SQL block to retrieve the details of the staff belonging to a particular department. Department code should be passed as a parameter to the cursor.

Lab 2. Exception Handling

Goals	Implementing Exception Handling ,Analyzing and Debugging
Time	1 hr

3.1: Modify the programs created in Lab2 to implement Exception Handling

3.2 The following PL/SQL block attempts to calculate bonus of staff for a given MGR_CODE. Bonus is to be considered as twice of salary. Though Exception Handling has been implemented but block is unable to handle the same.

Debug and verify the current behavior to trace the problem.

```
DECLARE
V_BONUS V_SAL%TYPE;
V_SAL STAFF_MASTER.STAFF_SAL%TYPE;

BEGIN
SELECT STAFF_SAL INTO V_SAL
FROM STAFF_MASTER
WHERE MGR_CODE=100006;

V_BONUS:=2*V_SAL;
DBMS_OUTPUT.PUT_LINE('STAFF SALARY IS ' || V_SAL);
DBMS_OUTPUT.PUT_LINE('STAFF BONUS IS ' || V_BONUS);

EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('GIVEN CODE IS NOT VALID.ENTER VALID CODE');
END;
```

Example 3: PL/SQL block

3.3 Rewrite the above block to achieve the requirement.

3.4
Predict the output of the following block ? What corrections would be needed to make it more efficient?

```
BEGIN
      DECLARE
      fname emp.ename%TYPE;
      BEGIN
      SELECT ename INTO fname
      FROM emp
      WHERE 1=2;

      DBMS_OUTPUT.PUT_LINE('This statement will print');
      EXCEPTION
      WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Some inner block error');
      END;

 EXCEPTION
 WHEN NO_DATA_FOUND THEN
 DBMS_OUTPUT.PUT_LINE('No data found in fname');

 WHEN OTHERS THEN
 DBMS_OUTPUT.PUT_LINE('Some outer block error');
 END;
```

Example 4: PL/SQL Block with Exception Handling

3.5 Debug the above block to trace the flow of control.
Additionally one can make appropriate changes in Select statement defined in the block to check the flow.

3.6: Write a PL/SQL program to check for the commission for an employee no 7369.
If no commission exists, then display the error message. Use Exceptions.

3.7: Write a PL/SQL block to drop any user defined table.

Lab 3. Procedures, Functions and Packages

Goals	The following set of exercises are designed to implement the following • Implement business logic using Database Programming like Procedures and Functions • Implement validations in Procedures and Functions • Working with Packages • Performance Tuning
Time	3.5 Hrs

Note: Procedures and functions should handle validations, pre-defined oracle server and user defined exceptions wherever applicable. Also use cursors wherever applicable.

4.1 Write a PL/SQL block to find the maximum salary of the staff in the given department. Note: Department code should be passed as parameter to the cursor.

4.2. Write a function to compute age. The function should accept a date and return age in years.

4.3. Write a procedure that accept staff code and update staff name to Upper case. If the staff name is null raise a user defined exception.

4.4 Write a procedure to find the manager of a staff. Procedure should return the following – Staff_Code, Staff_Name, Dept_Code and Manager Name.

4.5. Write a function to compute the following. Function should take Staff_Code and return the cost to company.
DA = 15% Salary, HRA= 20% of Salary, TA= 8% of Salary.
Special Allowance will be decided based on the service in the company.

< 1 Year	Nil
>=1 Year< 2 Year	10% of Salary
>=2 Year< 4 Year	20% of Salary
>4 Year	30% of Salary

4.6. Write a procedure that displays the following information of all staff

Staff_Name Department Name Designation Salary Status

Note: - Status will be (Greater, Lesser or Equal) respective to average salary of their own department. Display an error message Staff_Master table is empty if there is no matching record.

4.7. Write a procedure that accept Staff_Code and update the salary and store the old salary details in Staff_Master_Back (Staff_Master_Back has the same structure without any constraint) table.

Exp < 2 then no Update
Exp > 2 and < 5 then 20% of salary
Exp > 5 then 25% of salary

4.8. Create a procedure that accepts the book code as parameter from the user. Display the details of the students/staff that have borrowed that book and has not returned the same. The following details should be displayed

Student/Staff Code   Student/Staff Name   Issue Date Designation Expected Ret_Date

4.9. Write a package which will contain a procedure and a function.

Function: This function will return years of experience for a staff. This function will take the hiredate of the staff as an input parameter. The output will be rounded to the nearest year (1.4 year will be considered as 1 year and 1.5 year will be considered as 2 year).

Procedure: Capture the value returned by the above function to calculate the additional allowance for the staff based on the experience.
Additional Allowance = Year of experience x 3000
Calculate the additional allowance and store Staff_Code, Date of Joining, and Experience in years and additional allowance in Staff_Allowance table.

4.10. Write a procedure to insert details into Book_Transaction table. Procedure should accept the book code and staff/student code. Date of issue is current date and the expected return date should be 10 days from the current date. If the expected return date falls on Saturday or Sunday, then it should be the next working day.

4.11: Write a function named ‘get_total_records’, to pass the table name as a parameter, and get back the number of records that are contained in the table. Test your function with multiple tables.

4.12

Tune the following Oracle Procedure enabling to gain better performance.

Objective: The Procedure should update the salary of an employee and at the same time retrieve the employee's name and new salary into PL/SQL variables.

```
CREATE OR REPLACE PROCEDURE update_salary (emp_id NUMBER) IS
  v_name    VARCHAR2(15);
  v_newsal NUMBER;
BEGIN
 UPDATE emp_copy SET sal = sal * 1.1
 WHERE empno = emp_id;

 SELECT ename, sal INTO v_name, v_newsal
 FROM emp_copy
 WHERE empno = emp_id;

 DBMS_OUTPUT.PUT_LINE('Emp Name:' || v_name);
 DBMS_OUTPUT.PUT_LINE('Ename:' || v_newsal);
END;
```

Example 5: Oracle Procedure

4.13

The following procedure attempts to delete data from table passed as parameter.This procedure has compilation errors. Identify and correct the problem.

```
CREATE or REPLACE PROCEDURE gettable(table_name in varchar2) AS
BEGIN
DELETE FROM table_name;
END;
```

Example 6: Oracle Procedure

4.14

Write a procedure which prints the following report using procedure:

The procedure should take deptno as user input and appropriately print the emp details.

Also display :

Number of Employees,Total Salary,Maximum Salary,Average Salary

Note: The block should achieve the same without using Aggregate Functions.

Sample output for deptno 10 is shown below:

```
Employee Name :  CLARK
Employee Job : MANAGER
Employee Salary :  2450
Employee Comission :
*************************************
Employee Name :  KING
Employee Job : PRESIDENT
Employee Salary :  5000
Employee Comission :
*************************************
Employee Name :  MILLER
Employee Job : CLERK
Employee Salary :  1300
Employee Comission :
*************************************
Number of Employees :  3
Total Salary :  8750
Maximum Salary :  5000
Average Salary :  2916.67
-------------------------------------
```

Figure 1 :Report

4.15: Write a query to view the list of all procedures ,functions and packages from the Data Dictionary.

Lab 4. Case Study 1

Goals	Implementation of Procedures/Functions ,Packages with Testing and Review
Time	2.5hrs

Consider the following tables for the case study.

Customer_Masters

Name	Null?	Type
Cust_Id	Not Null	Number(6)
Cust_Name	Not Null	Varchar2(20)
Address		Varchar2(50)
Date_of_acc_creation		Date
Customer_Type		Char(3)

Note: Customer type can be either IND or NRI

Account_Masters Table

Name	Null?	Type
Account_Number	Not Null	Number(6)
Cust_ID		Number(6)
Account_Type		Char(3)
Ledger_Balance		Number(10)

Note: Account type can be either Savings (SAV) or Salary (SAL) account.
For savings account minimum amount should be 5000.

Transaction_Masters

Name	Null?	Type
Transaction_Id	Not Null	Number(6)
Account_Number		Number(6)
Date_of_Transaction		Date
From_Account_Number	Not Null	Number(6)
To_Account_Number	Not Null	Number(6)
Amount	Not Null	Number(10)
Transaction_Type	Not Null	Char(2)

Note: Transaction type can be either Credit (CR) or Debit (DB).

Procedure and function should be written inside a package.
All validations should be taken care.

5.1 Create appropriate Test Cases for the case study followed up by Self/Peer to Peer Review and close any defects for the same.

5.2Write a procedure to accept customer name, address, and customer type and account type. Insert the details into the respective tables.

5.3. Write a procedure to accept customer id, amount and the account number to which the customer requires to transfer money. Following validations need to be done

- Customer id should be valid
- From account number should belong to that customer
- To account number cannot be null but can be an account which need not exist in account masters (some other account)
- Adequate balance needs to be available for debit

5.4 Ensure all the Test cases defined are executed. Have appropriate Self/Peer to Peer

Code Review and close any defects for the same.

Lab 5. Case Study 2

Goals	Implementation of Procedures/Functions ,Packages with Testing and Review
Time	2.5hrs

Consider the following table (myEmp) structure for the case study

EmpNo Ename City Designation Salary

The following procedure accepts Task number and based on the same performs an appropriate task.

```
PROCEDURE run_task (task_number_in IN INTEGER)
IS
BEGIN
  IF task_number_in = 1
  THEN
    add_emp;
    --should add new emps in myEmp.
    --EmpNo should be inserted through Sequence.
    --All other data to be taken as parameters.Default location is Mumbai.
  END IF;
  IF task_number_in = 2
  THEN
    raise_sal;
    --should modify salary of an existing emp.
    --should take new salary and empno as input parameters
    --Should handle exception in case empno not found
    --upper limit of rasing salary is 30%. should raise exception appropriately
  END IF;
   IF task_number_in = 3
   THEN
    remove_emp;
         --should remove an existing emp
         --should take empno as parameter
      --Handle exception if empno not available
  END IF;
END run_task;
```

Example 7: Sample Oracle Procedure

However ,it has been observed the method adopted in above procedure is inefficient.

6.1

Create appropriate Test Cases for the case study followed up by Self/Peer to Peer Review and close any defects for the same.

6.2

Recreate the procedure (run_task) which is more efficient in performing the same.

6.3

Also, create relevant procedures (add_emp , raise_sal ,remove_emp)

with relevant logic (read comments)to verify the same.

6.4 Extend the above implementation using Packages

6.5) Ensure all the Test cases defined are executed. Have appropriate Self/Peer to Peer

Code Review and close any defects for the same.

Lab 6. Triggers

Goals	Triggers
Time	1 hr

1. Create a trigger called CHECK_SALARY_TRG on the staff_master table that fires before an INSERT or UPDATE operation on each row. The trigger must call the CHECK_SALARY procedure to carry out the business logic. The trigger should pass the new job ID and salary to the procedure parameters.

2. Update the CHECK_SALARY_TRG trigger to fire only when the job ID or salary values have actually changed.

    a. Implement the business rule using a WHEN clause to check whether the JOB_ID or SALARY values have changed.
    Note: Make sure that the condition handles the NULL in the OLD.column_name values if an INSERT operation is performed; otherwise, an insert operation will fail.

    b. Test the trigger by executing the EMP_PKG.ADD_EMPLOYEE procedure with the following parameter values: first_name='Eleanor', last name='Beh', email='EBEH', job='IT_PROG', sal=5000.

3. You are asked to prevent employees from being deleted during business hours.

    a. Write a statement trigger called DELETE_EMP_TRG on the EMPLOYEES table to prevent rows from being deleted during weekday business hours, which are from 9:00 a.m. to 6:00 p.m.

Lab 7. Oracle Supplied packages

Goals	• Understand oracle supplied packages
Time	1 Hours.

1.1. Create a procedure called EMPLOYEE_REPORT that generates an employee report in a file in the operating system, using the UTL_FILE package. The report should generate a list of employees who have exceeded the average salary of their departments.

a) Your program should accept two parameters. The first parameter is the output
directory. The second parameter is the name of the text file that is written.
Note: Use the directory location value UTL_FILE. Add an exception-handling section to handle errors that may be encountered when using the UTL_FILE package.

Solution :
-- The course has a directory alias provided called
"REPORTS_DIR" that
-- is associated with the /home/oracle/labs/plpu/reports
-- physical directory. Use the directory alias name
-- in quotes for the first parameter to create a file in the
appropriate directory.

```
CREATE OR REPLACE PROCEDURE employee_report(
  p_dir IN VARCHAR2, p_filename IN VARCHAR2) IS
  f UTL_FILE.FILE_TYPE;
  CURSOR cur_avg IS
    SELECT last_name, department_id, salary
    FROM employees outer
    WHERE salary > (SELECT AVG(salary)
                    FROM  employees inner
                    GROUP BY outer.department_id)
    ORDER BY department_id;
BEGIN
  f := UTL_FILE.FOPEN(p_dir, p_filename,'W');
  UTL_FILE.PUT_LINE(f, 'Employees who earn more than average salary:
');
  UTL_FILE.PUT_LINE(f, 'REPORT GENERATED ON ' ||SYSDATE);
  UTL_FILE.NEW_LINE(f);
  FOR emp IN cur_avg
  LOOP
    UTL_FILE.PUT_LINE(f,
    RPAD(emp.last_name, 30) || ' ' ||
    LPAD(NVL(TO_CHAR(emp.department_id,'9999'),'-'), 5) || ' ' ||
    LPAD(TO_CHAR(emp.salary, '$99,999.00'), 12));
  END LOOP;
  UTL_FILE.NEW_LINE(f);
  UTL_FILE.PUT_LINE(f, '*** END OF REPORT ***');
  UTL_FILE.FCLOSE(f);
END employee_report;
/
      EXECUTE employee_report('REPORTS_DIR','sal_rpt61.txt')
```

b). Invoke the program, using the second parameter with a name such as sal_rpt*xx*.txt, where *xx* represents your user number
(for example, 01, 15, and so on). The following is a sample output from the report file:

```
Employees who earn more than average salary:
REPORT GENERATED ON 26-FEB-04
Hartstein                20      $13,000.00
Raphaely                 30      $11,000.00
Marvis                   40      $6,500.00
...
        *** END OF REPORT ***
```

Note: The data displays the employee’s last name, department ID, and salary.

Lab 8. Design Considerations

Goals	• Understand Design Considerations
Time	1 Hours.

1.2. In the body of the package, define a private variable called emp_table based on the type defined in the specification to hold employee records. Implement the get_employees procedure to bulk fetch the data into the table.

```
CREATE OR REPLACE PACKAGE emp_pkg IS

  TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

  FUNCTION get_employee(p_emp_id
employees.employee_id%type)
    return employees%rowtype;

  FUNCTION get_employee(p_family_name
employees.last_name%type)
    return employees%rowtype;
```

1.3. Create a new procedure in the specification and body, called show_employees, that does not take arguments and displays the

contents of the private PL/SQL table variable (if any data exists).Hint:
Use the print_employee procedure.

```
CREATE OR REPLACE PACKAGE emp_pkg IS


  TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

  FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

  FUNCTION get_employee(p_family_name employees.last_name%type)
    return employees%rowtype;

  PROCEDURE get_employees(p_dept_id employees.department_id%type);

  PROCEDURE init_departments;

  PROCEDURE print_employee(p_rec_emp employees%rowtype);

  PROCEDURE show_employees;
```

```
-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  TYPE boolean_tab_type IS TABLE OF BOOLEAN
```

```
    INDEX BY BINARY_INTEGER;

  valid_departments boolean_tab_type;
  emp_table         emp_tab_type;

  FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
    RETURN BOOLEAN;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
  BEGIN
    IF valid_deptid(p_deptid) THEN
      INSERT INTO employees(employee_id, first_name, last_name,
email,
        job_id, manager_id, hire_date, salary, commission_pct,
department_id)
      VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
p_email,
        p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm, p_deptid);
    ELSE
      RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
Try again.');
    END IF;
  END add_employee;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
  BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email, p_deptid =>
p_deptid);
  END;

  PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
  BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p_empid;
  END get_employee;

  FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
  BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
  END;

  FUNCTION get_employee(p_family_name
employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
  BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
  END;

/* New get_employees procedure. */

  PROCEDURE get_employees(p_dept_id
employees.department_id%type) IS
  BEGIN
    SELECT * BULK COLLECT INTO emp_table
    FROM EMPLOYEES
    WHERE department_id = p_dept_id;
  END;

  PROCEDURE init_departments IS
  BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
      valid_departments(rec.department_id) := TRUE;
    END LOOP;
  END;

  PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id ||' '||
                        p_rec_emp.employee_id||' '||
                        p_rec_emp.first_name||' '||
                        p_rec_emp.last_name||' '||
                        p_rec_emp.job_id||' '||
                        p_rec_emp.salary);
  END;

  PROCEDURE show_employees IS
  BEGIN
    IF emp_table IS NOT NULL THEN
      DBMS_OUTPUT.PUT_LINE('Employees in Package table');
      FOR i IN 1 .. emp_table.COUNT
      LOOP
        print_employee(emp_table(i));
      END LOOP;
    END IF;
  END show_employees;

  FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
   RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
  BEGIN
    RETURN valid_departments.exists(p_deptid);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
    RETURN FALSE;
END valid_deptid;

BEGIN
  init_departments;
END emp_pkg;

/
SHOW ERRORS
```

Lab 9. Types of PLSQL pragma

Goals	• Understand Different type of PLSQL pragma.
Time	1 Hours.

1.4. The purpose of this example is to show the usage of predefined exceptions. Write a PL/SQL block to select the name of the employee with a given salary value.

a. Delete all records in the messages table. Use the DEFINE command to define a variable sal and initialize it to 6000.

b. In the declarative section declare two variables: ename of type employees.last_name and emp_sal of type employees.salary. Pass the value of the substitution variables to emp_sal.

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

d. 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>."

e. If the salary entered returns more than one row, 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>."

f. Handle any other exception with an appropriate exception handler and insert into the messages table the message "Some other error occurred."

g. Display the rows from the messages table to check whether the PL/SQL block has executed successfully. Sample output is shown below.

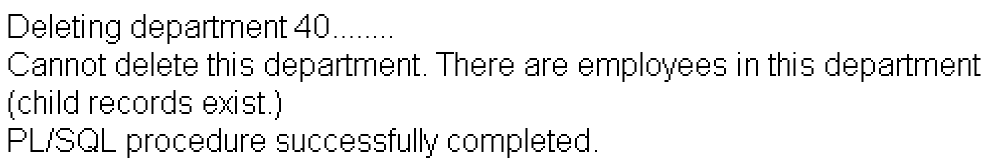
Solution :

```
SET VERIFY OFF
DECLARE
  v_ename     employees.last_name%TYPE;
  v_emp_sal  employees.salary%TYPE := 6000;
BEGIN
  SELECT     last_name
  INTO                  v_ename
  FROM                  employees
  WHERE      salary = v_emp_sal;
  INSERT INTO messages (results)
  VALUES (v_ename || ' - ' || v_emp_sal);

  EXCEPTION
  WHEN no_data_found THEN
  INSERT INTO messages (results)
  VALUES ('No employee with a salary of '|| TO_CHAR(v_emp_sal));
  WHEN too_many_rows THEN
  INSERT INTO messages (results)
  VALUES ('More than one employee with a salary of '||
TO_CHAR(v_emp_sal));
```

3.2. The purpose of this example is to show how to declare exceptions with a standard Oracle server error. Use the Oracle server error ORA-02292 (integrity constraint violated – child record found).

a. In the declarative section, declare an exception childrecord_exists. Associate the declared exception with the standard Oracle server error –02292.
b. In the executable section, display 'Deleting department 40.....'. Include a DELETE statement to delete the department with department_id 40
c. Include an exception section to handle the childrecord_exists exception and display the appropriate message. Sample output is shown below.



```
Deleting department 40........
Cannot delete this department. There are employees in this department
(child records exist.)
PL/SQL procedure successfully completed.
```

3.Load the script lab_07_04_soln.sql.

a. Observe the declarative section of the outer block. Note that the
no_such_employee exception is declared.

b. Look for the comment “RAISE EXCEPTION HERE.” If the value of emp_id is
not between 100 and 206, then raise the no_such_employee exception.

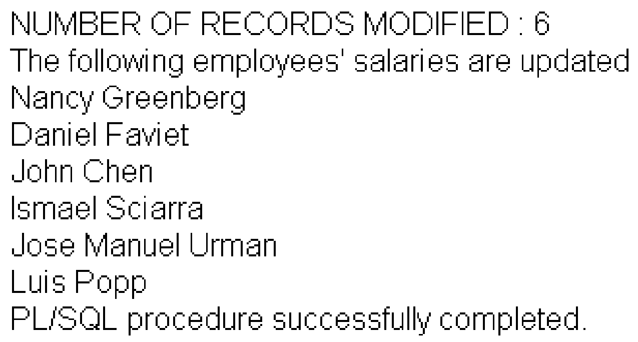
c. Look for the comment “INCLUDE EXCEPTION SECTION FOR OUTER
BLOCK” and handle the exceptions no_such_employee and too_many_rows.
Display appropriate messages when the exceptions occur.
The employees table has only one employee working in the HR department and therefore the code is written accordingly.
The too_many_rows exception is handled to indicate that the select statement retrieves more than one employee working in the HR department.

d.Close the outer block.

e.Save your script as lab_08_03_soln.sql.

f.Execute the script. Enter the employee number and the department number and observe the output. Enter different values and check for different conditions.
The sample output for employee ID 203 and department ID 100 is shown below.

```
SET SERVEROUTPUT ON
DECLARE
  e_childrecord_exists EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_childrecord_exists, -02292);
BEGIN
   DBMS_OUTPUT.PUT_LINE(' Deleting department 40........');
  delete from departments where department_id=40;
EXCEPTION
  WHEN e_childrecord_exists THEN
  DBMS_OUTPUT.PUT_LINE(' Cannot delete this department. There
are employees in this department (child records exist.) ');
END;
```



```
NUMBER OF RECORDS MODIFIED : 6
The following employees' salaries are updated
Nancy Greenberg
Daniel Faviet
John Chen
Ismael Sciarra
Jose Manuel Urman
Luis Popp
PL/SQL procedure successfully completed.
```

Lab 10. Collection elements

Goals	• Understand Different collection elements.
Time	1 Hour

a. Example of nested tables

```
DECLARE
   TYPE names_table IS TABLE OF VARCHAR2(10);
   TYPE grades IS TABLE OF INTEGER;

   names names_table;
   marks grades;
   total integer;
BEGIN
   names := names_table('Kavita', 'Pritam', 'Ayan', 'Rishav',
'Aziz');
   marks:= grades(98, 97, 78, 87, 92);
   total := names.count;
   dbms_output.put_line('Total '|| total || ' Students');
   FOR i IN 1 .. total LOOP
      dbms_output.put_line('Student:'||names(i)||', Marks:' ||
marks(i));
   end loop;
END;
/
```

Lab 11. Oracle 11g features

Goals	• Understand oracle 11g features.
Time	2 Hours.

1.5. Create the ADD_EMPLOYEE procedure to add an employee to the EMPLOYEES table. The row should be added to the EMPLOYEES table if the VALID_DEPTID function returns TRUE; otherwise, alert the user with an appropriate message. Provide the following parameters (with defaults specified in parentheses): first_name, last_name, email, job (SA_REP), mgr (145), sal (1000), comm (0), and deptid (30). Use the EMPLOYEES_SEQ sequence to set the employee_id column, and set hire_date to TRUNC(SYSDATE).

Solution :

```
CREATE OR REPLACE PROCEDURE add_employee(
   p_first_name employees.first_name%TYPE,
   p_last_name  employees.last_name%TYPE,
   p_email      employees.email%TYPE,
   p_job        employees.job_id%TYPE         DEFAULT 'SA_REP',
   p_mgr        employees.manager_id%TYPE     DEFAULT 145,
   p_sal        employees.salary%TYPE         DEFAULT 1000,
   p_comm       employees.commission_pct%TYPE DEFAULT 0,
   p_deptid     employees.department_id%TYPE  DEFAULT 30)
IS
BEGIN
 IF valid_deptid(p_deptid) THEN
   INSERT INTO employees(employee_id, first_name,
last_name, email,
     job_id, manager_id, hire_date, salary, commission_pct,
department_id)
   VALUES (employees_seq.NEXTVAL, p_first_name,
p_last_name, p_email,
     p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
 ELSE
   RAISE_APPLICATION_ERROR (-20204, 'Invalid department
ID. Try again.');
 END IF;
END add_employee;
```

Lab 12. Best practices of PL SQL code, Dynamic SQL

Goals	• Understand best practices of PL SQL code and dynamic sql.
Time	1 Hour.

6.1. Create a package called TABLE_PKG that uses Native Dynamic SQL to create or drop a table, and to populate, modify, and delete rows from the table.

a. Create a package specification with the following procedures:

```
PROCEDURE make(table_name VARCHAR2, col_specs VARCHAR2)
PROCEDURE add_row(table_name VARCHAR2, col_values VARCHAR2,
cols VARCHAR2 := NULL)
PROCEDURE upd_row(table_name VARCHAR2, set_values
VARCHAR2, conditions VARCHAR2 := NULL)
PROCEDURE del_row(table_name VARCHAR2,conditions
VARCHAR2 := NULL);
PROCEDURE remove(table_name VARCHAR2)
```

Ensure that subprograms manage optional default parameters with NULL values.

Solution :

```
CREATE OR REPLACE PACKAGE table_pkg IS
  PROCEDURE make(p_table_name VARCHAR2, p_col_specs
VARCHAR2);
  PROCEDURE add_row(p_table_name VARCHAR2,
p_col_values VARCHAR2,
    p_cols VARCHAR2 := NULL);
  PROCEDURE upd_row(p_table_name VARCHAR2,
p_set_values VARCHAR2,
    p_conditions VARCHAR2 := NULL);
  PROCEDURE del_row(p_table_name VARCHAR2,
p_conditions VARCHAR2 := NULL);
  PROCEDURE remove(p_table_name VARCHAR2);
END table_pkg;
/
SHOW ERRORS
```

b. Create the package body that accepts the parameters and dynamically constructs the appropriate SQL statements that are executed using Native Dynamic SQL, except for the remove procedure that should be written using the DBMS_SQL package.

Solution :

```
CREATE OR REPLACE PACKAGE BODY table_pkg IS
  PROCEDURE execute(p_stmt VARCHAR2) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(p_stmt);
    EXECUTE IMMEDIATE p_stmt;
  END;

  PROCEDURE make(p_table_name VARCHAR2, p_col_specs
VARCHAR2) IS
    v_stmt VARCHAR2(200) := 'CREATE TABLE '||
p_table_name ||
                          ' (' || p_col_specs || ')';
  BEGIN
    execute(v_stmt);
  END;
  PROCEDURE add_row(p_table_name VARCHAR2,
p_col_values VARCHAR2,
    p_cols VARCHAR2 := NULL) IS
    v_stmt VARCHAR2(200) := 'INSERT INTO '|| p_table_name;
  BEGIN
    IF p_cols IS NOT NULL THEN
      v_stmt := v_stmt || ' (' || p_cols || ')';
    END IF;
    v_stmt := v_stmt || ' VALUES (' || p_col_values || ')';
    execute(v_stmt);
  END;

  PROCEDURE upd_row(p_table_name VARCHAR2,
p_set_values VARCHAR2,
    p_conditions VARCHAR2 := NULL) IS
    v_stmt VARCHAR2(200) := 'UPDATE '|| p_table_name || '
SET ' || p_set_values;
  BEGIN
    IF p_conditions IS NOT NULL THEN
      v_stmt := v_stmt || ' WHERE ' || p_conditions;
    END IF;
    execute(v_stmt);
  END;

  PROCEDURE del_row(p_table_name VARCHAR2,
p_conditions VARCHAR2 := NULL) IS
    v_stmt VARCHAR2(200) := 'DELETE FROM '||
p_table_name;
  BEGIN
```

```
    IF p_conditions IS NOT NULL THEN
      v_stmt := v_stmt || ' WHERE ' || p_conditions;
    END IF;
    execute(v_stmt);
  END;

  PROCEDURE remove(p_table_name VARCHAR2) IS
    cur_id INTEGER;
    v_stmt VARCHAR2(100) := 'DROP TABLE '||p_table_name;
  BEGIN
    cur_id := DBMS_SQL.OPEN_CURSOR;
    DBMS_OUTPUT.PUT_LINE(v_stmt);
    DBMS_SQL.PARSE(cur_id, v_stmt, DBMS_SQL.NATIVE);
    -- Parse executes DDL statements,no EXECUTE is required.
    DBMS_SQL.CLOSE_CURSOR(cur_id);
  END;

END table_pkg;
/
SHOW ERRORS
```

Appendices

Appendix A: Oracle Standards

Key points to keep in mind:

1. Write comments in your stored Procedures, Functions and SQL batches generously, whenever something is not very obvious. This helps other programmers to clearly understand your code. Do not worry about the length of the comments, as it will not impact the performance.
2. Prefix the table names with owner names, as this improves readability, and avoids any unnecessary confusion.

Some more Oracle standards:

To be shared by Faculty in class

Appendix B: Coding Best Practices

1. Perform all your referential integrity checks and data validations by using constraints (foreign key and check constraints). These constraints are faster than triggers. So use triggers only for auditing, custom tasks, and validations that cannot be performed by using these constraints.
2. Do not call functions repeatedly within your stored procedures, triggers, functions, and batches. For example: You might need the length of a string variable in many places of your procedure. However do not call the LENGTH function whenever it is needed. Instead call the LENGTH function once, and store the result in a variable, for later use.