ITE 5220 Oracle Database Programming using PL/SQL

Lab Exercise 9[Chapter 9 and 10 (Triggers)]

12 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: Creating Statement and Row Triggers[6 Points]

In this practice, you create statement and row triggers. You also create procedures that are invoked from within the triggers.

1) The rows in the `JOBS` table store a minimum and maximum salary allowed for different `JOB_ID` values. You are asked to write code to ensure that employees' salaries fall in the range allowed for their job type, for insert and update operations.
   a) Create a procedure called `CHECK_SALARY` as follows:
      i) The procedure accepts two parameters, one for an employee's job ID string and the other for the salary.
      ii) The procedure uses the job ID to determine the minimum and maximum salary for the specified job.
      iii) If the salary parameter does not fall within the salary range of the job, inclusive of the minimum and maximum, then it should raise an application exception, with the message "`Invalid salary <sal>. Salaries for job <jobid> must be between <min> and <max>`". Replace the various items in the message with values supplied by parameters and variables populated by queries. Save the file.

Open the `sol_09_01_a.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below. To compile the procedure, right-click the procedure's name in the Object Navigation tree, and then select Compile.

```
CREATE OR REPLACE PROCEDURE check_salary (p_the_job VARCHAR2,
p_the_salary NUMBER) IS
  v_minsal jobs.min_salary%type;
  v_maxsal jobs.max_salary%type;
BEGIN
  SELECT min_salary, max_salary INTO v_minsal, v_maxsal
  FROM jobs
  WHERE job_id = UPPER(p_the_job);
  IF p_the_salary NOT BETWEEN v_minsal AND v_maxsal THEN
    RAISE_APPLICATION_ERROR(-20100,
      'Invalid salary $' ||p_the_salary ||'. '||
      'Salaries for job '|| p_the_job ||
      ' must be between $'|| v_minsal ||' and $' || v_maxsal);
  END IF;
END;
/
SHOW ERRORS
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

```
PROCEDURE check_salary Compiled.
No Errors.
```

Messages - Log

```
CHECK_SALARY Compiled
```

b) Create a trigger called CHECK_SALARY_TRG on the EMPLOYEES table that fires before an INSERT or UPDATE operation on each row:

i) The trigger must call the CHECK_SALARY procedure to carry out the business logic.

ii) The trigger should pass the new job ID and salary to the procedure parameters.

Open the `sol_09_01_b.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below. To compile the trigger, right-click the trigger's name in the Object Navigation tree, and then select Compile.

```
CREATE OR REPLACE TRIGGER check_salary_trg
BEFORE INSERT OR UPDATE OF job_id, salary
ON employees
FOR EACH ROW
BEGIN
  check_salary(:new.job_id, :new.salary);
END;
/
SHOW ERRORS
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

```
TRIGGER check_salary_trg Compiled.
No Errors.
```

Messages - Log

```
CHECK_SALARY_TRG Compiled
```

2) Test the CHECK_SAL_TRG trigger using the following cases:

a) Using your EMP_PKG.ADD_EMPLOYEE procedure, add employee Eleanor Beh to department 30. What happens and why?

Open the sol_09_02_a.sql file in the D:\labs\PLPU\solns folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

```
EXECUTE emp_pkg.add_employee('Eleanor', 'Beh', 30)
```

```
Error starting at line 1 in command:
EXECUTE emp_pkg.add_employee('Eleanor', 'Beh', 30)
Error report:
ORA-20100: Invalid salary $1000. Salaries for job SA_REP must be between $6000 and $12000
ORA-06512: at "ORA61.CHECK_SALARY", line 9
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'
ORA-06512: at "ORA61.EMP_PKG", line 35
ORA-06512: at "ORA61.EMP_PKG", line 51
ORA-06512: at line 1
```

The trigger raises an exception because the `EMP_PKG.ADD_EMPLOYEE` procedure invokes an overloaded version of itself that uses the default salary of $1,000 and a default job ID of `SA_REP`. However, the `JOBS` table stores a minimum salary of $ 6,000 for the `SA_REP` type.

b) Update the salary of employee 115 to $2,000. In a separate update operation, change the employee job ID to `HR_REP`. What happens in each case?

Open the `sol_09_02_b.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below. To compile the package, right-click the package's name in the Object Navigation tree, and then select Compile.

```
UPDATE employees
  SET salary = 2000
WHERE employee_id = 115;

UPDATE employees
  SET job_id = 'HR_REP'
WHERE employee_id = 115;
```

```
Error starting at line 1 in command:
UPDATE employees
  SET salary = 2000
WHERE employee_id = 115
Error report:
SQL Error: ORA-20100: Invalid salary $2000. Salaries for job PU_CLERK must be between $2500 and $5500
ORA-06512: at "ORA61.CHECK_SALARY", line 9
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'


Error starting at line 5 in command:
UPDATE employees
  SET job_id = 'HR_REP'
WHERE employee_id = 115
Error report:
SQL Error: ORA-20100: Invalid salary $3100. Salaries for job HR_REP must be between $4000 and $9000
ORA-06512: at "ORA61.CHECK_SALARY", line 9
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'
```

The first update statement fails to set the salary to $2,000. The check salary trigger rule fails the update operation because the new salary for employee 115 is less than the minimum allowed for the `PU_CLERK` job ID.

The second update fails to change the employee's job because the current employee's salary of $3,100 is less than the minimum for the new `HR_REP` job ID.

c) Update the salary of employee 115 to $2,800. What happens?

Open the `sol_09_02_c.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

```
UPDATE employees
  SET salary = 2800
WHERE employee_id = 115;
```

```
1 rows updated
```

The update operation is successful because the new salary falls within the acceptable range for the current job ID.

3) 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.

      Open the `sol_09_03_a.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below. To compile the trigger, right-click the trigger's name in the Object Navigation tree, and then click Compile.

```
CREATE OR REPLACE TRIGGER check_salary_trg
BEFORE INSERT OR UPDATE OF job_id, salary
ON employees FOR EACH ROW
WHEN (new.job_id <> NVL(old.job_id,'?') OR
      new.salary <> NVL(old.salary,0))
BEGIN
  check_salary(:new.job_id, :new.salary);
END;
/
SHOW ERRORS
```

```
TRIGGER check_salary_trg Compiled.
No Errors.
```

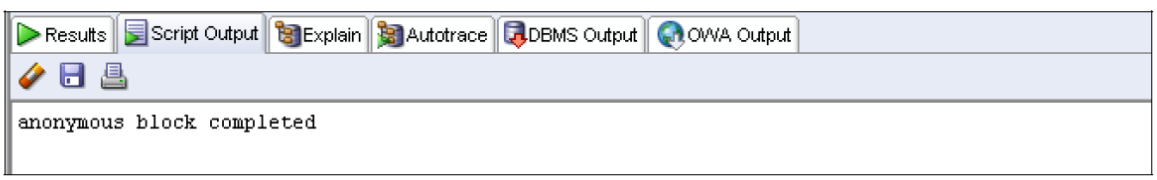
```
CHECK_SALARY_TRG Compiled
```

b) Test the trigger by executing the `EMP_PKG.ADD_EMPLOYEE` procedure with the following parameter values:

- `p_first_name: 'Eleanor'`
- `p_last name: 'Beh'`
- `p_Email: 'EBEH'`
- `p_Job: 'IT_PROG'`
- `p_Sal: 5000`

Open the `sol_09_03_b.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

```
BEGIN
  emp_pkg.add_employee('Eleanor', 'Beh', 'EBEH',
                       job => 'IT_PROG', sal => 5000);
END;
/
```



c) Update employees with the `IT_PROG` job by incrementing their salary by $2,000. What happens?

Open the `sol_09_03_c.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

```
UPDATE employees
  SET salary = salary + 2000
WHERE job_id = 'IT_PROG';
```

```
Error starting at line 1 in command:
UPDATE employees
  SET salary = salary + 2000
WHERE job_id = 'IT_PROG'
Error report:
SQL Error: ORA-20100: Invalid salary $11000. Salaries for job IT_PROG must be between $4000 and $10000
ORA-06512: at "ORA61.CHECK_SALARY", line 9
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'
```

An employee's salary in the specified job type exceeds the maximum salary for that job type. No employee salaries in the `IT_PROG` job type are updated.

d) Update the salary to $9,000 for `Eleanor Beh`.

Open the `sol_09_03_d.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

```
UPDATE employees
  SET salary = 9000
WHERE employee_id = (SELECT employee_id
                     FROM employees
                     WHERE last_name = 'Beh');
```

```
1 rows updated
```

Hint: Use an `UPDATE` statement with a subquery in the `WHERE` clause. What happens?

e) Change the job of `Eleanor Beh` to `ST_MAN` using another `UPDATE` statement with a subquery. What happens?

Open the `sol_09_03_e.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

```
UPDATE employees
  set job_id = 'ST_MAN'
WHERE employee_id = (SELECT employee_id
                     FROM employees
                     WHERE last_name = 'Beh');
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

```
Error starting at line 1 in command:
UPDATE employees
  set job_id = 'ST_MAN'
WHERE employee_id = (SELECT employee_id
                     FROM employees
                     WHERE last_name = 'Beh')
Error report:
SQL Error: ORA-20100: Invalid salary $9000. Salaries for job ST_MAN must be between $5500 and $8500
ORA-06512: at "ORA61.CHECK_SALARY", line 9
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'
```

The maximum salary of the new job type is less than the employee's current salary; therefore, the update operation fails.

4) 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 AM to 6:00 PM.

Open the `sol_09_04_a.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below. To compile the trigger, right-click the trigger's name in the Object Navigation tree, and then click Compile.

```
CREATE OR REPLACE TRIGGER delete_emp_trg
BEFORE DELETE ON employees
DECLARE
  the_day VARCHAR2(3) := TO_CHAR(SYSDATE, 'DY');
  the_hour PLS_INTEGER := TO_NUMBER(TO_CHAR(SYSDATE, 'HH24'));
BEGIN
   IF (the_hour BETWEEN 9 AND 18) AND (the_day NOT IN
('SAT','SUN')) THEN
     RAISE_APPLICATION_ERROR(-20150,
      'Employee records cannot be deleted during the business
       hours of 9AM and 6PM');
   END IF;
END;
/
SHOW ERRORS
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

```
TRIGGER delete_emp_trg Compiled.
No Errors.
```

```
Messages - Log
DELETE_EMP_TRG Compiled
```

b) Attempt to delete employees with `JOB_ID` of `SA_REP` who are not assigned to a department.

Hint: This is employee Grant with ID 178.

Open the `sol_09_04_b.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below. To compile the trigger, right-click the trigger's name in the Object Navigation tree, and then click Compile.

```
DELETE FROM employees
WHERE job_id = 'SA_REP'
  AND   department_id IS NULL;
```

```
Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Error starting at line 1 in command:
DELETE FROM employees
  WHERE job_id = 'SA_REP'
  AND   department_id IS NULL
Error report:
SQL Error: ORA-20150: Employee records cannot be deleted during the business hours of 9AM and 6PM
ORA-06512: at "ORA61.DELETE_EMP_TRG", line 6
ORA-04088: error during execution of trigger 'ORA61.DELETE_EMP_TRG'
```

Practice 2: Managing Data Integrity Rules and Mutating Table Exceptions [6 Points]

In this practice, you implement a simple business rule for ensuring data integrity of employees' salaries with respect to the valid salary range for their jobs. You create a trigger for this rule. During this process, your new triggers cause a cascading effect with triggers created in the practice section of the previous lesson. The cascading effect results in a mutating table exception on the `JOBS` table. You then create a PL/SQL package and additional triggers to solve the mutating table issue.

1) Employees receive an automatic increase in salary if the minimum salary for a job is increased to a value larger than their current salaries. Implement this requirement through a package procedure called by a trigger on the `JOBS` table. When you attempt to update the minimum salary in the `JOBS` table and try to update the employees' salaries, the `CHECK_SALARY` trigger attempts to read the `JOBS` table, which is subject to change, and you get a mutating table exception that is resolved by creating a new package and additional triggers.

    a. Update your `EMP_PKG` package (that you last updated in Practice 8) as follows:

        i. Add a procedure called `SET_SALARY` that updates the employees' salaries.

        ii. The `SET_SALARY` procedure accepts the following two parameters: The job ID for those salaries that may have to be updated, and the new minimum salary for the job ID

Open the `sol_10_01_a.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown as follows. To compile the trigger, right-click the package's name in the Object Navigation tree, and then click Compile. The newly added code is highlighted in bold letters in the following code box.

```
-- Package SPECIFICATION

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;

  /* New set_salary procedure */

  PROCEDURE set_salary(p_jobid VARCHAR2, p_min_salary NUMBER);

END emp_pkg;
/
SHOW ERRORS

-- 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

    PROCEDURE audit_newemp IS
      PRAGMA AUTONOMOUS_TRANSACTION;
      user_id VARCHAR2(30) := USER;
    BEGIN
      INSERT INTO log_newemp (entry_id, user_id, log_time,
name)
      VALUES (log_newemp_seq.NEXTVAL, user_id,
sysdate,p_first_name||' '||p_last_name);
      COMMIT;
    END audit_newemp;

  BEGIN -- add_employee
    IF valid_deptid(p_deptid) THEN
      audit_newemp;
      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;

  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;

/* New set_salary procedure */

PROCEDURE set_salary(p_jobid VARCHAR2, p_min_salary NUMBER) IS
    CURSOR cur_emp IS
      SELECT employee_id
      FROM employees
      WHERE job_id = p_jobid AND salary < p_min_salary;
  BEGIN
    FOR rec_emp IN cur_emp
    LOOP
      UPDATE employees
        SET salary = p_min_salary
      WHERE employee_id = rec_emp.employee_id;
    END LOOP;
  END set_salary;

BEGIN
  init_departments;
END emp_pkg;


/
SHOW ERRORS
```

```
PACKAGE emp_pkg Compiled.
No Errors.
PACKAGE BODY emp_pkg Compiled.
No Errors.
```

b. Create a row trigger named `UPD_MINSALARY_TRG` on the `JOBS` table that invokes the `EMP_PKG.SET_SALARY` procedure, when the minimum salary in the `JOBS` table is updated for a specified job ID.

Open the `sol_10_01_b.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below. To compile the trigger, right-click the trigger's name in the Object Navigation tree, and then click Compile. The code and the results are shown below.

```
CREATE OR REPLACE TRIGGER upd_minsalary_trg
AFTER UPDATE OF min_salary ON JOBS
FOR EACH ROW
BEGIN
  emp_pkg.set_salary(:new.job_id, :new.min_salary);
END;
/
SHOW ERRORS
```

```
TRIGGER upd_minsalary_trg Compiled.
No Errors.
```

c. Write a query to display the employee ID, last name, job ID, current salary, and minimum salary for employees who are programmers—that is, their `JOB_ID` is `'IT_PROG'`. Then, update the minimum salary in the `JOBS` table to increase it by $1,000. What happens?

Open the `sol_10_01_c.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

```
SELECT employee_id, last_name, salary
FROM employees
WHERE job_id = 'IT_PROG';

UPDATE jobs
  SET min_salary = min_salary + 1000
WHERE job_id = 'IT_PROG';
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

```
EMPLOYEE_ID          LAST_NAME                 SALARY
-------------------- ------------------------- ----------------------
103                  Hunold                    9000
104                  Ernst                     6000
105                  Austin                    4800
106                  Pataballa                 4800
107                  Lorentz                   4200
214                  Beh                       9000

6 rows selected


Error starting at line 5 in command:
UPDATE jobs
  SET min_salary = min_salary + 1000
WHERE job_id = 'IT_PROG'
Error report:
SQL Error: ORA-04091: table ORA61.JOBS is mutating, trigger/function may not see it
ORA-06512: at "ORA61.CHECK_SALARY", line 5
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'
ORA-06512: at "ORA61.EMP_PKG", line 143
ORA-06512: at "ORA61.UPD_MINSALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.UPD_MINSALARY_TRG'
04091. 00000 -  "table %s.%s is mutating, trigger/function may not see it"
*Cause:    A trigger (or a user defined plsql function that is referenced in
           this statement) attempted to look at (or modify) a table that was
           in the middle of being modified by the statement which fired it.
*Action:   Rewrite the trigger (or function) so it does not read that table.
```

The update of the `min_salary` column for job `'IT_PROG'` fails because the `UPD_MINSALARY_TRG` trigger on the `JOBS` table attempts to update the employees' salaries by calling the `EMP_PKG.SET_SALARY` procedure. The `SET_SALARY` procedure causes the `CHECK_SALARY_TRG` trigger to fire (a cascading effect). The `CHECK_SALARY_TRG` calls the `CHECK_SALARY` procedure, which attempts to read the `JOBS` table data, this encountering the mutating table exception on the `JOBS` table, which is the table that is subject to the original update operation.

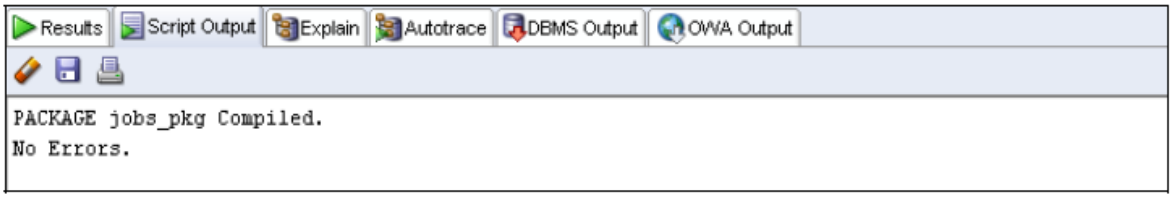
2) To resolve the mutating table issue, create a JOBS_PKG package to maintain in memory a copy of the rows in the JOBS table. Next, modify the CHECK_SALARY procedure to use the package data rather than issue a query on a table that is mutating to avoid the exception. However, you must create a BEFORE INSERT OR UPDATE statement trigger on the EMPLOYEES table to initialize the JOBS_PKG package state before the CHECK_SALARY row trigger is fired.

   a. Create a new package called JOBS_PKG with the following specification:

```
PROCEDURE initialize;
FUNCTION get_minsalary(jobid VARCHAR2) RETURN NUMBER;
FUNCTION get_maxsalary(jobid VARCHAR2) RETURN NUMBER;
PROCEDURE set_minsalary(jobid VARCHAR2,min_salary
                        NUMBER);
PROCEDURE set_maxsalary(jobid VARCHAR2,max_salary
                        NUMBER);
```

Open the `sol_10_02_a.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below. To compile the package's specification, right-click the package's name or body in the Object Navigator tree, and then Select Compile.

```
CREATE OR REPLACE PACKAGE jobs_pkg IS
  PROCEDURE initialize;
  FUNCTION get_minsalary(p_jobid VARCHAR2) RETURN NUMBER;
  FUNCTION get_maxsalary(p_jobid VARCHAR2) RETURN NUMBER;
  PROCEDURE set_minsalary(p_jobid VARCHAR2, p_min_salary
NUMBER);
  PROCEDURE set_maxsalary(p_jobid VARCHAR2, p_max_salary
NUMBER);
END jobs_pkg;
/
SHOW ERRORS
```



   b. Implement the body of JOBS_PKG as follows:

      i. Declare a private PL/SQL index-by table called `jobs_tab_type` that is indexed by a string type based on the `JOBS.JOB_ID%TYPE`.

ii. Declare a private variable called `jobstab` based on the `jobs_tab_type`.

iii. The `INITIALIZE` procedure reads the rows in the `JOBS` table by using a cursor loop, and uses the `JOB_ID` value for the `jobstab` index that is assigned its corresponding row.

iv. The `GET_MINSALARY` function uses a `p_jobid` parameter as an index to the `jobstab` and returns the `min_salary` for that element.

v. The `GET_MAXSALARY` function uses a `p_jobid` parameter as an index to the `jobstab` and returns the `max_salary` for that element.

vi. The `SET_MINSALARY` procedure uses its `p_jobid` as an index to the `jobstab` to set the `min_salary` field of its element to the value in the `min_salary` parameter.

vii. The `SET_MAXSALARY` procedure uses its `p_jobid` as an index to the `jobstab` to set the `max_salary` field of its element to the value in the `max_salary` parameter.

Open the `sol_10_02_b.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below. To compile the package's body, right-click the package's name or body in the Object Navigator tree, and then Select Compile.

```
CREATE OR REPLACE PACKAGE BODY jobs_pkg IS
  TYPE jobs_tab_type IS TABLE OF jobs%rowtype
    INDEX BY jobs.job_id%type;
  jobstab jobs_tab_type;

  PROCEDURE initialize IS
  BEGIN
    FOR rec_job IN (SELECT * FROM jobs)
    LOOP
      jobstab(rec_job.job_id) := rec_job;
    END LOOP;
  END initialize;

  FUNCTION get_minsalary(p_jobid VARCHAR2) RETURN NUMBER IS
  BEGIN
    RETURN jobstab(p_jobid).min_salary;
  END get_minsalary;

  FUNCTION get_maxsalary(p_jobid VARCHAR2) RETURN NUMBER IS
  BEGIN
    RETURN jobstab(p_jobid).max_salary;
  END get_maxsalary;
```

```
  PROCEDURE set_minsalary(p_jobid VARCHAR2, p_min_salary
NUMBER) IS
  BEGIN
    jobstab(p_jobid).max_salary := p_min_salary;
  END set_minsalary;

  PROCEDURE set_maxsalary(p_jobid VARCHAR2, p_max_salary
NUMBER) IS
  BEGIN
    jobstab(p_jobid).max_salary := p_max_salary;
  END set_maxsalary;

END jobs_pkg;
/
SHOW ERRORS
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

```
PACKAGE BODY jobs_pkg Compiled.
No Errors.
```

Messages - Log

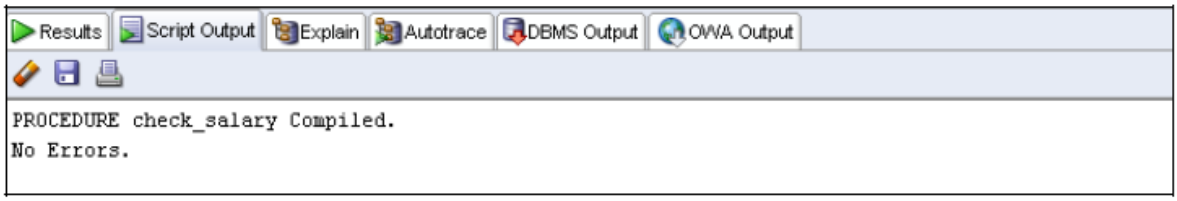
```
JOB_PKG Compiled
```

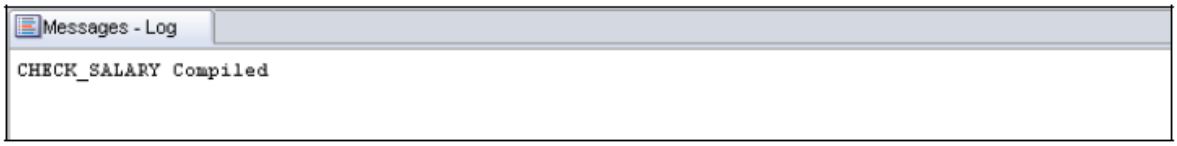
c. Copy the `CHECK_SALARY` procedure from Practice 10, Exercise 1a, and modify the code by replacing the query on the `JOBS` table with statements to set the local `minsal` and `maxsal` variables with values from the `JOBS_PKG` data by calling the appropriate `GET_*SALARY` functions. This step should eliminate the mutating trigger exception.

Open the `sol_10_02_c.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below. To compile the procedure, right-click the procedure's name in the Object Navigator, and then select Compile.

```
CREATE OR REPLACE PROCEDURE check_salary (p_the_job VARCHAR2,
p_the_salary NUMBER) IS
  v_minsal jobs.min_salary%type;
  v_maxsal jobs.max_salary%type;
BEGIN
  /*
```

```
  ** Commented out to avoid mutating trigger exception on the
JOBS table
  SELECT min_salary, max_salary INTO v_minsal, v_maxsal
  FROM jobs
  WHERE job_id = UPPER(p_the_job);
  */
  v_minsal := jobs_pkg.get_minsalary(UPPER(p_the_job));
  v_maxsal := jobs_pkg.get_maxsalary(UPPER(p_the_job));
  IF p_the_salary NOT BETWEEN v_minsal AND v_maxsal THEN
    RAISE_APPLICATION_ERROR(-20100,
      'Invalid salary $'||p_the_salary||'. '||
      'Salaries for job '|| p_the_job ||
      ' must be between $'|| v_minsal ||' and $' || v_maxsal);
  END IF;
END;
/
SHOW ERRORS
```





d. Implement a `BEFORE INSERT OR UPDATE` statement trigger called `INIT_JOBPKG_TRG` that uses the `CALL` syntax to invoke the `JOBS_PKG.INITIALIZE` procedure to ensure that the package state is current before the DML operations are performed.

Open the `sol_10_02_d.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below. To compile the trigger, right-click the trigger's name in the Object Navigator, and then select Compile.

```
CREATE OR REPLACE TRIGGER init_jobpkg_trg
BEFORE INSERT OR UPDATE ON jobs
CALL jobs_pkg.initialize
/
SHOW ERRORS
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

```
TRIGGER init_jobpkg_trg Compiled.
No Errors.
```

Messages - Log

```
INIT_JOBPKG_TRG Compiled
```

e. Test the code changes by executing the query to display the employees who are programmers, and then issue an update statement to increase the minimum salary of the `IT_PROG` job type by 1,000 in the `JOBS` table. Follow this up with a query on the employees with the `IT_PROG` job type to check the resulting changes. Which employees' salaries have been set to the minimum for their jobs?

Open the `sol_10_02_e.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

```
SELECT employee_id, last_name, salary
FROM employees
WHERE job_id = 'IT_PROG';

UPDATE jobs
  SET min_salary = min_salary + 1000
WHERE job_id = 'IT_PROG';

SELECT employee_id, last_name, salary
FROM employees
WHERE job_id = 'IT_PROG';
```

```
EMPLOYEE_ID            LAST_NAME                 SALARY
---------------------- ------------------------- ----------------------
103                    Hunold                    9000
104                    Ernst                     6000
105                    Austin                    4800
106                    Pataballa                 4800
107                    Lorentz                   4200
214                    Beh                       9000

6 rows selected

1 rows updated
EMPLOYEE_ID            LAST_NAME                 SALARY
---------------------- ------------------------- ----------------------
103                    Hunold                    9000
104                    Ernst                     6000
105                    Austin                    5000
106                    Pataballa                 5000
107                    Lorentz                   5000
214                    Beh                       9000

6 rows selected
```

The employees with last names `Austin`, `Pataballa`, and `Lorentz` have all had their salaries updated. No exception occurred during this process, and you implemented a solution for the mutating table trigger exception.

3) Because the `CHECK_SALARY` procedure is fired by `CHECK_SALARY_TRG` before inserting or updating an employee, you must check whether this still works as expected.

   a. Test this by adding a new employee using `EMP_PKG.ADD_EMPLOYEE` with the following parameters: `('Steve', 'Morse', 'SMORSE', and sal => 6500)`. What happens?

Open the `sol_10_03_a.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

```
EXECUTE emp_pkg.add_employee('Steve', 'Morse', 'SMORSE', p_sal
=> 6500)
```

```
anonymous block completed
```

b. To correct the problem encountered when adding or updating an employee:

   i. Create a `BEFORE INSERT OR UPDATE` statement trigger called `EMPLOYEE_INITJOBS_TRG` on the `EMPLOYEES` table that calls the `JOBS_PKG.INITIALIZE` procedure.

   ii. Use the `CALL` syntax in the trigger body.

c. Test the trigger by adding employee Steve Morse again. Confirm the inserted record in the `EMPLOYEES` table by displaying the employee ID, first and last names, salary, job ID, and department ID.

Open the `sol_10_03_c.sql` file in the `D:\labs\PLPU\solns` folder, or copy and paste the following code in the SQL Worksheet area. Click the Run Script (F5) icon on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

```
EMPLOYEE_ID          FIRST_NAME           LAST_NAME            SALARY               JOB_ID
-------------------- -------------------- -------------------- -------------------- ------
222                  Steve                Morse                6500                 SA_REP

1 rows selected
```