**Practice 9**

**Triggers. Part 1**

1. Create a row trigger that will fire after DELETE operation in the Departments table. The trigger must delete the corresponding records in the Employees table. For example, you delete the department 30 from the Departments table; the trigger must delete all the employees that work in the department 30.

CREATE OR REPLACE TRIGGER DELETE\_EMPLOYEES\_AFTER\_DEPARTMENT

AFTER DELETE ON departments

FOR EACH ROW

BEGIN

DELETE FROM employees

WHERE department\_id = :OLD.department\_id;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

END DELETE\_EMPLOYEES\_AFTER\_DEPARTMENT;

/

1. Create a row trigger that will fire before INSERT operation into the Employees table. The trigger must raise an error if the new value of salary for ‘SA\_REP’ job\_id is not between 2000 and 7000.

CREATE OR REPLACE TRIGGER CHECK\_SALARY\_RANGE\_BEFORE\_INSERT

BEFORE INSERT ON employees

FOR EACH ROW

BEGIN

IF :NEW.job\_id = 'SA\_REP' AND (:NEW.salary < 2000 OR :NEW.salary > 7000) THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Salary for SA\_REP job must be between 2000 and 7000.');

END IF;

END CHECK\_SALARY\_RANGE\_BEFORE\_INSERT;

/

1. Create a row trigger that will fire after INSERT operation into the Employees table. The trigger must output the line “The employee with id \_\_\_\_ is inserted”.

CREATE OR REPLACE TRIGGER AFTER\_INSERT\_EMPLOYEE

AFTER INSERT ON employees

FOR EACH ROW

BEGIN

DBMS\_OUTPUT.PUT\_LINE('The employee with id ' || :NEW.employee\_id || ' is inserted');

END AFTER\_INSERT\_EMPLOYEE;

/

1. Create a row trigger that will fire before Delete operation in the Departments table. The trigger must raise an error if there are any child records in the Employees table. For example, you delete the department 30 from the Departments table; the trigger must check if there are any employees in the Employees table that work in the department 30; if so, the trigger must raise an error.

CREATE OR REPLACE TRIGGER BEFORE\_DELETE\_DEPARTMENT

BEFORE DELETE ON departments

FOR EACH ROW

DECLARE

v\_employee\_count NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_employee\_count

FROM Employees

WHERE department\_id = :OLD.department\_id;

IF v\_employee\_count > 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Cannot delete department. Employees exist in the department.');

END IF;

END BEFORE\_DELETE\_DEPARTMENT;

/

1. Create a row trigger that will fire before UDPATE of salary operation in the Employees table. The trigger must raise an error if the new value of salary equals its old value.

CREATE OR REPLACE TRIGGER BEFORE\_UPDATE\_SALARY

BEFORE UPDATE OF salary ON employees

FOR EACH ROW

BEGIN

IF :NEW.salary = :OLD.salary THEN

RAISE\_APPLICATION\_ERROR(-20001, 'New salary value must be different from the current value.');

END IF;

END BEFORE\_UPDATE\_SALARY;

/

1. Create a row trigger that will fire on Employees table before any DML operation (insert, update, delete). The trigger must output the corresponding message depending on the operation. For example, before DELETE operation the following line must appear “The employee with id \_\_\_\_ is deleted”.

CREATE OR REPLACE TRIGGER BEFORE\_DML\_EMPLOYEES

BEFORE INSERT OR UPDATE OR DELETE ON Employees

FOR EACH ROW

BEGIN

IF INSERTING THEN

DBMS\_OUTPUT.PUT\_LINE('The employee with id ' || :NEW.employee\_id || ' is inserted');

ELSIF UPDATING THEN

DBMS\_OUTPUT.PUT\_LINE('The employee with id ' || :OLD.employee\_id || ' is updated');

ELSIF DELETING THEN

DBMS\_OUTPUT.PUT\_LINE('The employee with id ' || :OLD.employee\_id || ' is deleted');

END IF;

END BEFORE\_DML\_EMPLOYEES;

/

**Part 2**

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.

* + - 1. Create a procedure called CHECK\_SALARY as follows (separately not in any package):
         1. The procedure accepts two parameters, one for an employee’s job ID string and the other for the salary.
         2. The procedure uses the job ID to determine the minimum and maximum salary for the specified job.
         3. 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.
      2. Create a trigger (separately not in any package) called CHECK\_SALARY\_TRG on the EMPLOYEES table that fires before an INSERT or UPDATE operation on each row:
         1. The trigger must call the CHECK\_SALARY procedure to carry out the business logic.
         2. The trigger should pass the new job ID and salary to the procedure parameters.

CREATE OR REPLACE PROCEDURE CHECK\_SALARY(

p\_job\_id IN VARCHAR2,

p\_salary IN NUMBER

) AS

v\_min\_salary NUMBER;

v\_max\_salary NUMBER;

BEGIN

SELECT MIN\_SALARY, MAX\_SALARY

INTO v\_min\_salary, v\_max\_salary

FROM JOBS

WHERE JOB\_ID = p\_job\_id;

IF p\_salary < v\_min\_salary OR p\_salary > v\_max\_salary THEN

RAISE\_APPLICATION\_ERROR(

-20001,

'Invalid salary ' || p\_salary || '. Salaries for job ' || p\_job\_id || ' must be between ' || v\_min\_salary || ' and ' || v\_max\_salary

);

END IF;

END CHECK\_SALARY;

/

CREATE OR REPLACE TRIGGER CHECK\_SALARY\_TRG

BEFORE INSERT OR UPDATE ON EMPLOYEES

FOR EACH ROW

BEGIN

CHECK\_SALARY(:NEW.JOB\_ID, :NEW.SALARY);

END CHECK\_SALARY\_TRG;

/

2. Test the CHECK\_SAL\_TRG using the following cases:

a. Using your EMP\_PKG.ADD\_EMPLOYEE procedure, add employee Eleanor Beh to department 30. What happens and why?

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?

* + - 1. Update the salary of employee 115 to $2,800. What happens?

3. Update the CHECK\_SALARY\_TRG trigger to fire only when the job ID or salary values have actually changed.

* + - 1. 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.

1. 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

c. Update employees with the IT\_PROG job by incrementing their salary by $2,000. What happens?

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

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

* + - 1. Change the job of Eleanor Beh to ST\_MAN using another UPDATE statement with a subquery. What happens?

CREATE OR REPLACE TRIGGER CHECK\_SALARY\_TRG

BEFORE INSERT OR UPDATE OF JOB\_ID, SALARY ON EMPLOYEES

FOR EACH ROW

WHEN ((NEW.JOB\_ID IS NOT NULL AND (NEW.JOB\_ID <> OLD.JOB\_ID OR OLD.JOB\_ID IS NULL))

OR (NEW.SALARY IS NOT NULL AND (NEW.SALARY <> OLD.SALARY OR OLD.SALARY IS NULL)))

BEGIN

CHECK\_SALARY(:NEW.JOB\_ID, :NEW.SALARY);

END CHECK\_SALARY\_TRG;

/

BEGIN

EMP\_PKG.ADD\_EMPLOYEE('Eleanor', 'Beh', 'EBEH', 'IT\_PROG', 5000);

COMMIT;

UPDATE EMPLOYEES

SET SALARY = SALARY + 2000

WHERE JOB\_ID = 'IT\_PROG';

COMMIT;

UPDATE EMPLOYEES

SET SALARY = 9000

WHERE EMPLOYEE\_ID = (SELECT EMPLOYEE\_ID FROM EMPLOYEES WHERE LAST\_NAME = 'Beh');

COMMIT;

UPDATE EMPLOYEES

SET JOB\_ID = 'ST\_MAN'

WHERE EMPLOYEE\_ID = (SELECT EMPLOYEE\_ID FROM EMPLOYEES WHERE LAST\_NAME = 'Beh');

COMMIT;

END;

/

SELECT EMPLOYEE\_ID FROM EMPLOYEES WHERE LAST\_NAME = 'Beh';