Practices for Lesson 9: Creating Triggers

Chapter 9

Practices for Lesson 9: Overview

Overview

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

Note:

1. Before starting this practice, execute `/home/oracle/labs/plpu/code_ex/cleanup_scripts/cleanup_09.sql` script.
2. If you missed a step in a practice, please run the appropriate solution script for that practice step before proceeding to the next step or the next practice.

Practice 9-1: Creating Statement and Row Triggers

Overview

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

Note: Execute `cleanup_09.sql script` from `/home/oracle/labs/plpu/code_ex/cleanup_scripts/` before performing the following tasks.

Task

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:
      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.
   b. Create a trigger 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.
2. Test the `CHECK_SALARY_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?
   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?
   c. 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.
   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:
      - `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?

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

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.

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.