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.

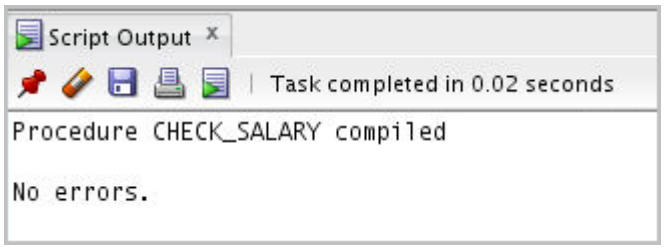
Solution 9-1: Creating Statement and Row Triggers

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:
      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.

      Open sol_09.sql script from /home/oracle/labs/plpu/soln directory. Uncomment and select the code under Task 1_a. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

```
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
```

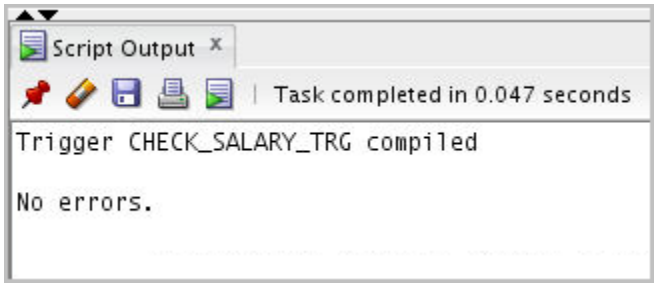


b. Create a trigger called `CHECK_SALARY_TRG` on the `EMPLOYEES` table that fires before an `INSERT` or `UPDATE` operation on each row:

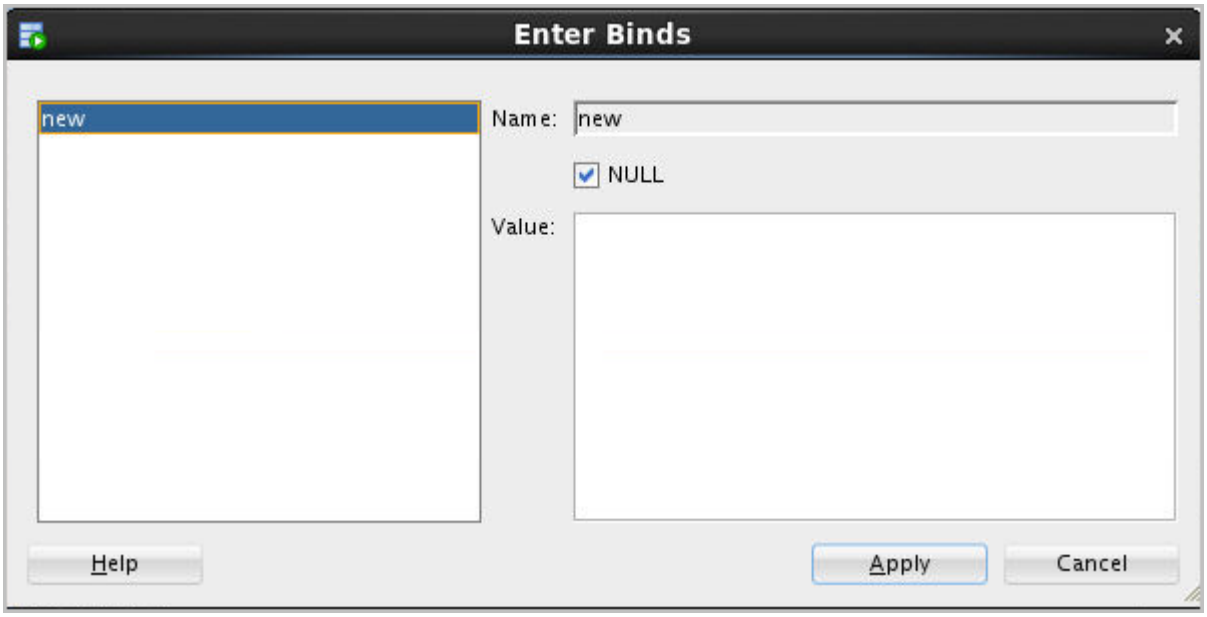
1) The trigger must call the `CHECK_SALARY` procedure to carry out the business logic.
2) The trigger should pass the new job ID and salary to the procedure parameters.

Uncomment and select the code under Task 1_b. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

```
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
```



Note: The trigger compilation might ask for values of bind variables while compiling. You may encounter a wizard as the one below. Click Apply.



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?

      Uncomment and select the code under Task 2_a. Click the Run Script icon (or press F5) 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)
```

```
Script Output
Task completed in 0.067 seconds
Error starting at line : 43 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 $12008
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 24
ORA-06512: at "ORA61.EMP_PKG", line 40
ORA-06512: at line 1
```

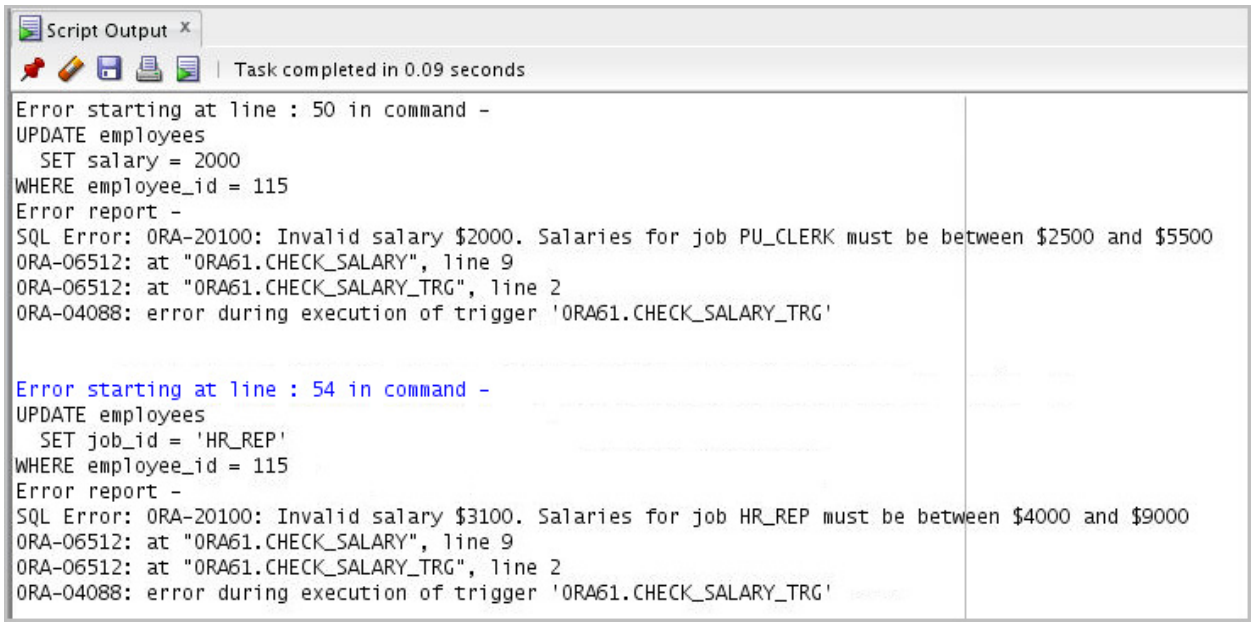
      The trigger raises an exception because the `EMP_PKG.ADD_EMPLOYEE` procedure invokes an overloaded version of itself that uses the default salary of $1,000 and a default job ID of `SA_REP`. However, the `JOBS` table stores a minimum salary of $6,000 for the `SA_REP` type.

   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?

Uncomment and select the code under Task 2_b. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

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

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



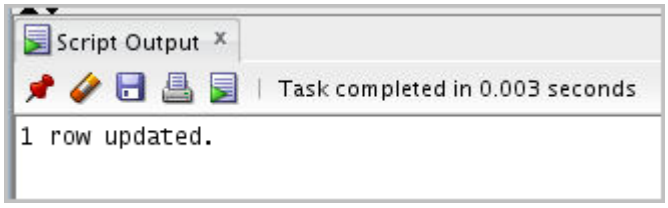
The first update statement fails to set the salary to $2,000. The check salary trigger rule fails the update operation because the new salary for employee 115 is less than the minimum allowed for the `PU_CLERK` job ID.

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?

Uncomment and select the code under Task 2_c. Click the Run Script icon (or press F5) 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;
```



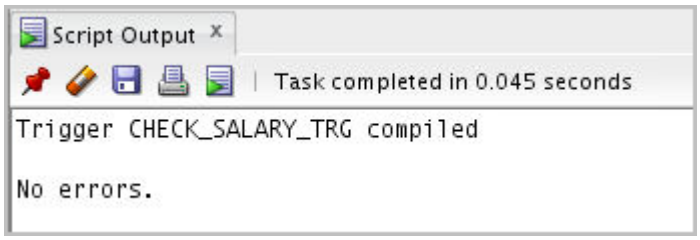
The update operation is successful because the new salary falls within the acceptable range for the current job ID.

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.

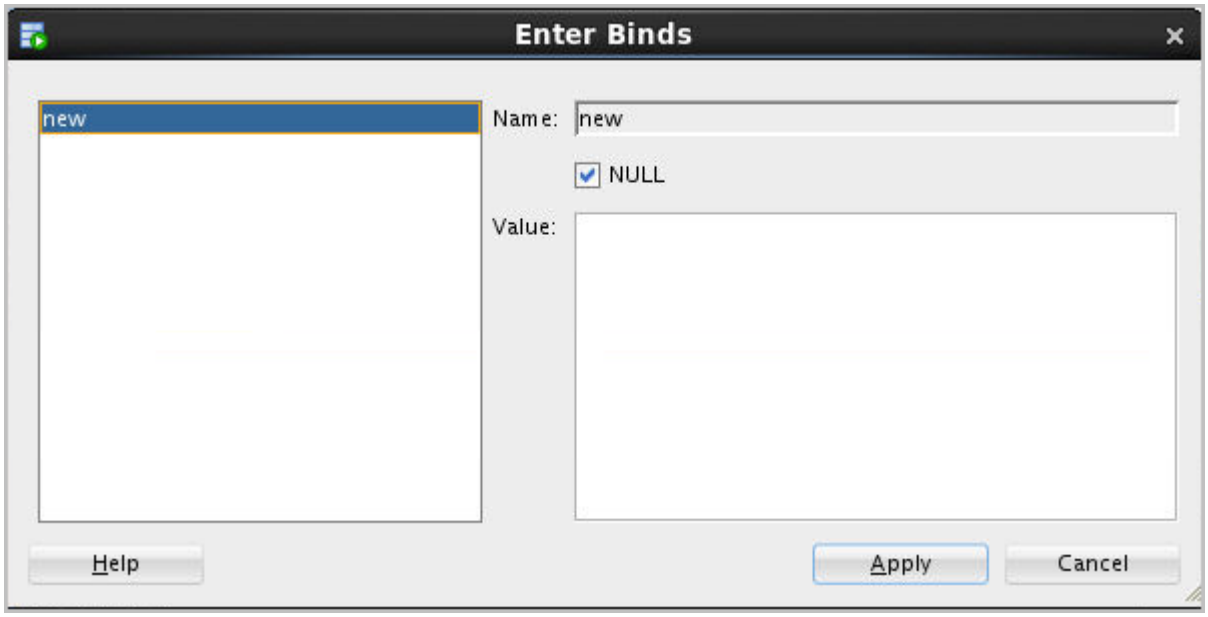
      Note: Make sure that the condition handles the `NULL` in the `OLD.column_name` values if an `INSERT` operation is performed; otherwise, an insert operation will fail.

      Uncomment and select the code under Task 3_a. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

```
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
```



Note: The trigger compilation might ask for values of bind variables while compiling. You may encounter a wizard as the one below. Click Apply.

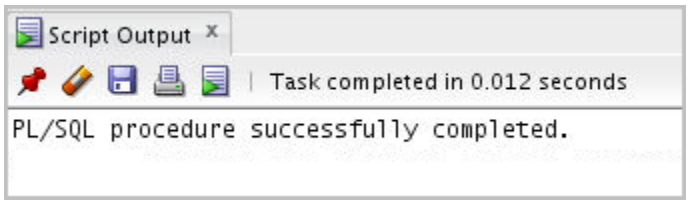


b. Test the trigger by executing the `EMP_PKG.ADD_EMPLOYEE` procedure with the following parameter values:

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

Uncomment and select the code under Task 3_b. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

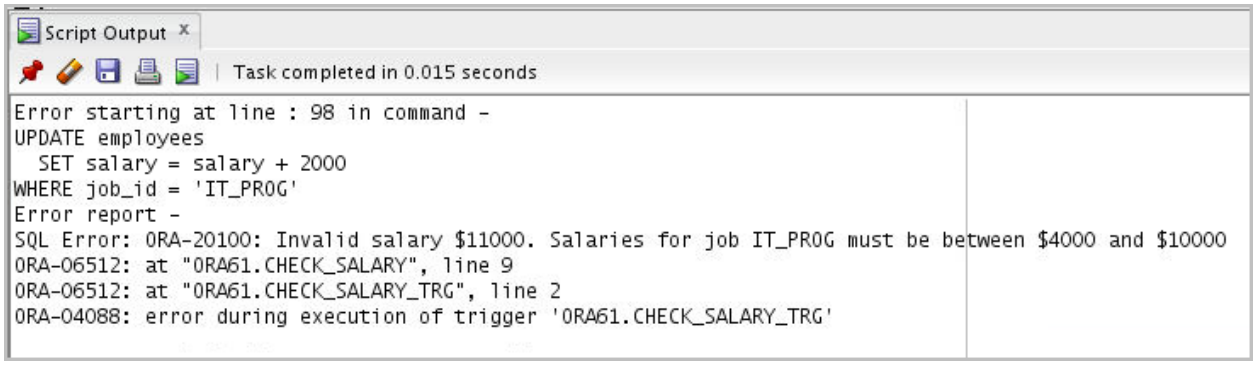
```
BEGIN
  emp_pkg.add_employee('Eleanor', 'Beh', 'EBEH',
                       p_job => 'IT_PROG', p_sal => 5000);
END;
/
```



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

Uncomment and select the code under Task 3_c. Click the Run Script icon (or press F5) 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';
```



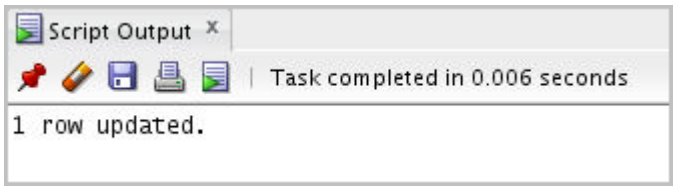
An employee's salary in the specified job type exceeds the maximum salary for that job type. No employee salaries in the `IT_PROG` job type are updated.

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

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

Uncomment and select the code under Task 3_d. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

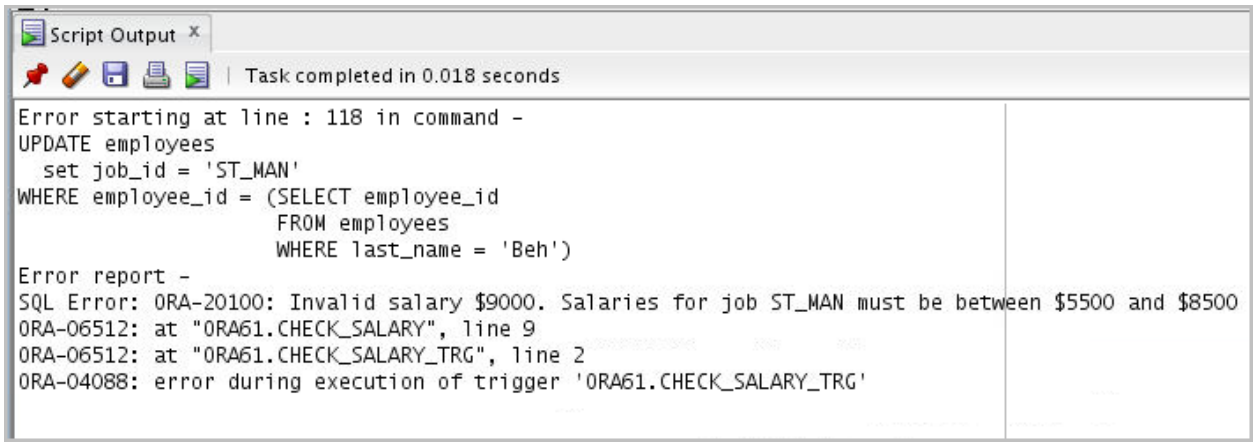
```
UPDATE employees
  SET salary = 9000
WHERE employee_id = (SELECT employee_id
                     FROM employees
                     WHERE last_name = 'Beh');
```



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

Uncomment and select the code under Task 3_e. Click the Run Script icon (or press F5) 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');
```

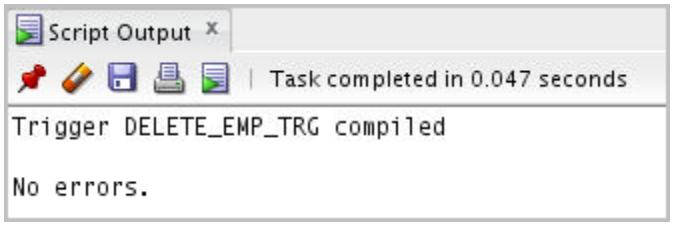


The maximum salary of the new job type is less than the employee's current salary; therefore, the update operation fails.

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.

      Uncomment and select the code under Task 4_a. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

```
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
```

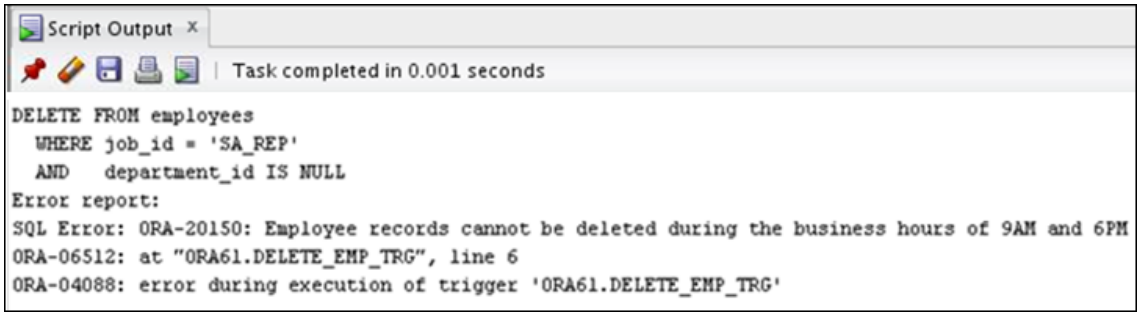


b. Attempt to delete employees with `JOB_ID` of `SA_REP` who are not assigned to a department.

Hint: This is employee Grant with ID 178.

Uncomment and select the code under Task 4_b. Click the Run Script icon (or press F5) on the SQL Worksheet toolbar to run the script. The code and the results are shown below.

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



Note: Depending on the current time on your host machine in the classroom, you may or may not be able to perform the delete operations. For example, in the screen capture above, the delete operation failed as it was performed outside the allowed business hours (based on the host machine time).