**Zhunagali Kanat IT2-2003**

**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\_emp

after delete on departments

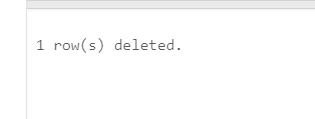
for each row

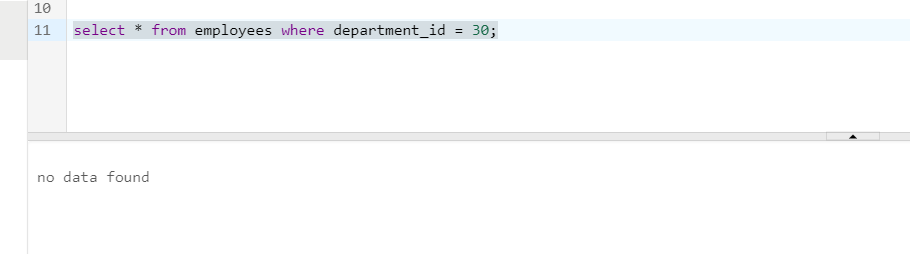
begin

delete from employees where department\_id = :old.department\_id;

end;

delete from departments where department\_id = 30;





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 err\_sal

before insert on employees

for each row

begin

if :new.job\_id = 'SA\_REP' and :new.salary not between 2000 and 7000 then

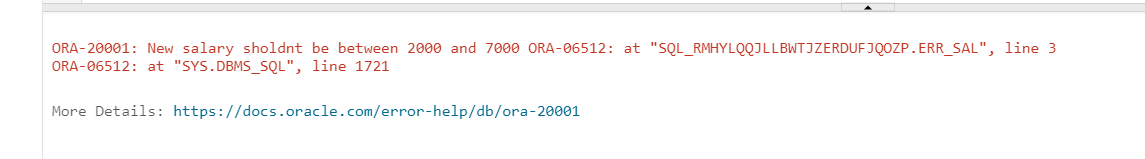
raise\_application\_error(-20001, 'New salary sholdnt be between 2000 and 7000');

end if;

end;

insert into employees (employee\_id, last\_name, job\_id, salary, email, hire\_date)

values (12333, 'Temp emp', 'SA\_REP', 1999, 'TEMPe3EMP', TRUNC(SYSDATE));



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 emp\_inst\_trigger

after insert on employees

for each row

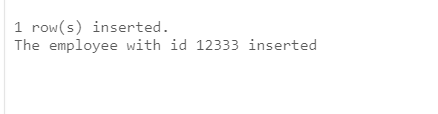
begin

dbms\_output.put\_line('The employee with id ' || :new.employee\_id || ' inserted');

end;

insert into employees (employee\_id, last\_name, job\_id, salary, email, hire\_date)

values (12333, 'Temp emp', 'SA\_REP', 1999, 'TEMPe3EMP', TRUNC(SYSDATE));



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 check\_emp\_in\_dep\_trigger

before delete on departments

for each row

declare

v\_count number;

begin

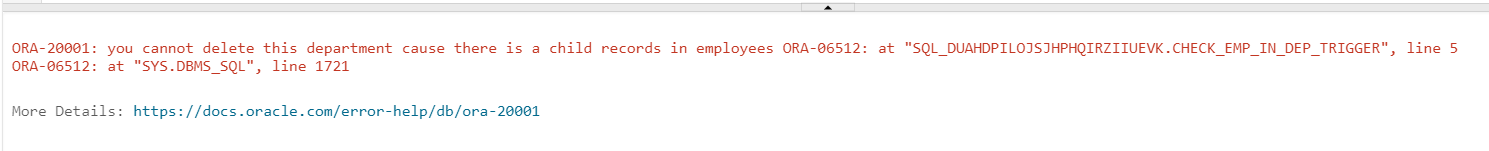
select count(\*) into v\_count from employees where department\_id = :old.department\_id;

if v\_count > 0 then raise\_application\_error(-20001, 'you cannot delete this department cause there is a child records in employees');

end if;

end;

delete from departments where department\_id = 30;



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 check\_emp\_in\_dep\_trigger

before update on employees

for each row

begin

if :old.salary = :new.salary then raise\_application\_error(-20001, 'Salary is equal to old value!!!');

end if;

end;

update employees set salary = 24000 where employee\_id = 100;



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 dml\_trigger

before insert or update or delete on employees

for each row

begin

if inserting then

dbms\_output.put\_line('The new id is inseted ' || :new.employee\_id);

end if;

if updating then

dbms\_output.put\_line('The employee with id '|| :old.employee\_id || ' is updated');

end if;

if deleting then

dbms\_output.put\_line('The employee with id ' || :old.employee\_id || ' is deleting');

end if;

end;

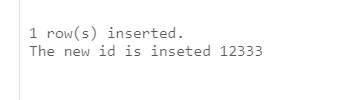
insert into employees (employee\_id, last\_name, job\_id, salary, email, hire\_date)

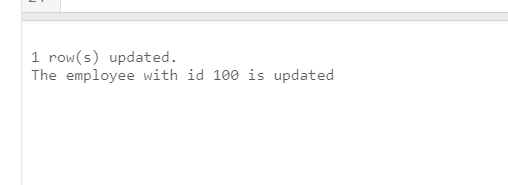
values (12333, 'Temp emp', 'SA\_REP', 1999, 'TEMPe3EMP', TRUNC(SYSDATE));

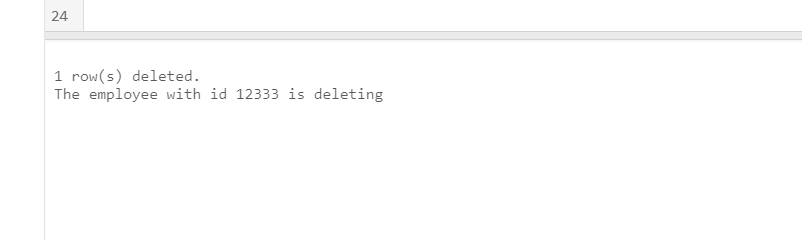
update employees set salary = 2404 where employee\_id = 100;

delete from employees where employee\_id = 12333;

select \* from 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(

emp\_job\_id in jobs.job\_id%type,

emp\_sal in employees.salary%type

) is max\_s jobs.min\_salary%type;

min\_s jobs.max\_salary%type;

begin

select min\_salary, max\_salary into min\_s, max\_s from jobs where job\_id = emp\_job\_id;

if emp\_sal not between min\_s and max\_s then raise\_application\_error(-20001, 'Invalid salary ' || emp\_sal || '. Salaries for job ' || emp\_job\_id || ' must be between ' || min\_s || ' and ' || max\_s);

end if;

end check\_salary;

create or replace trigger check\_salary\_trg

before insert or update on employees

for each row

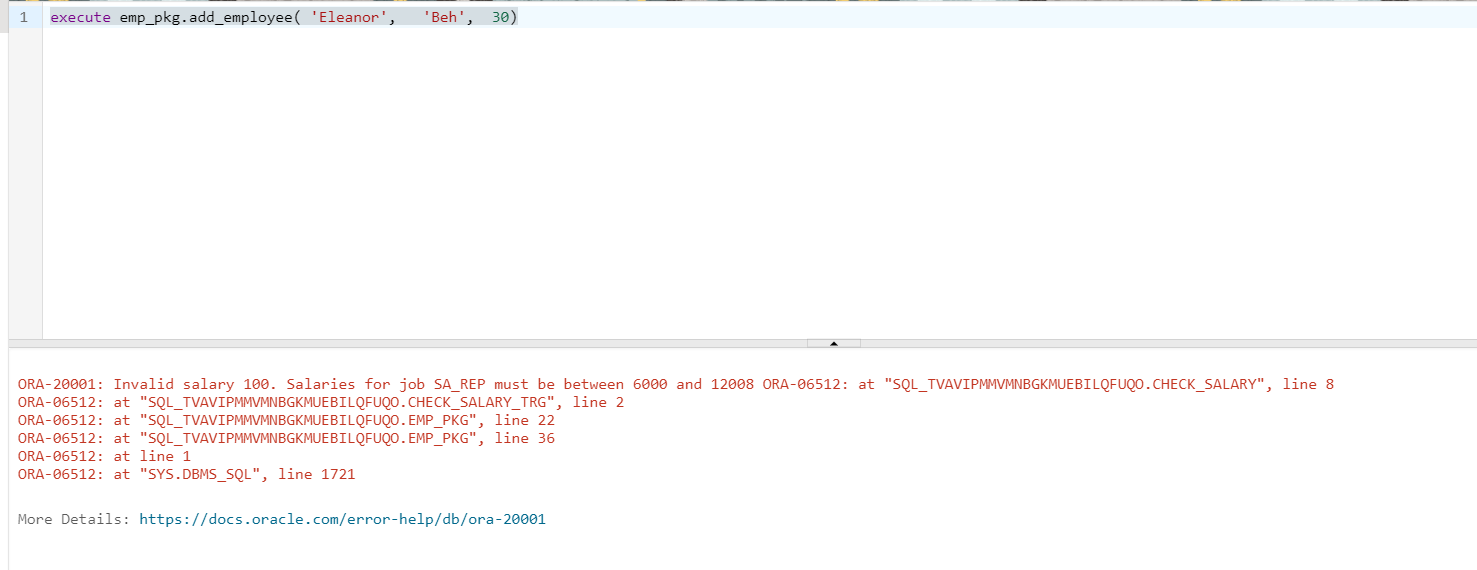
begin

check\_salary(:new.job\_id, :new.salary);

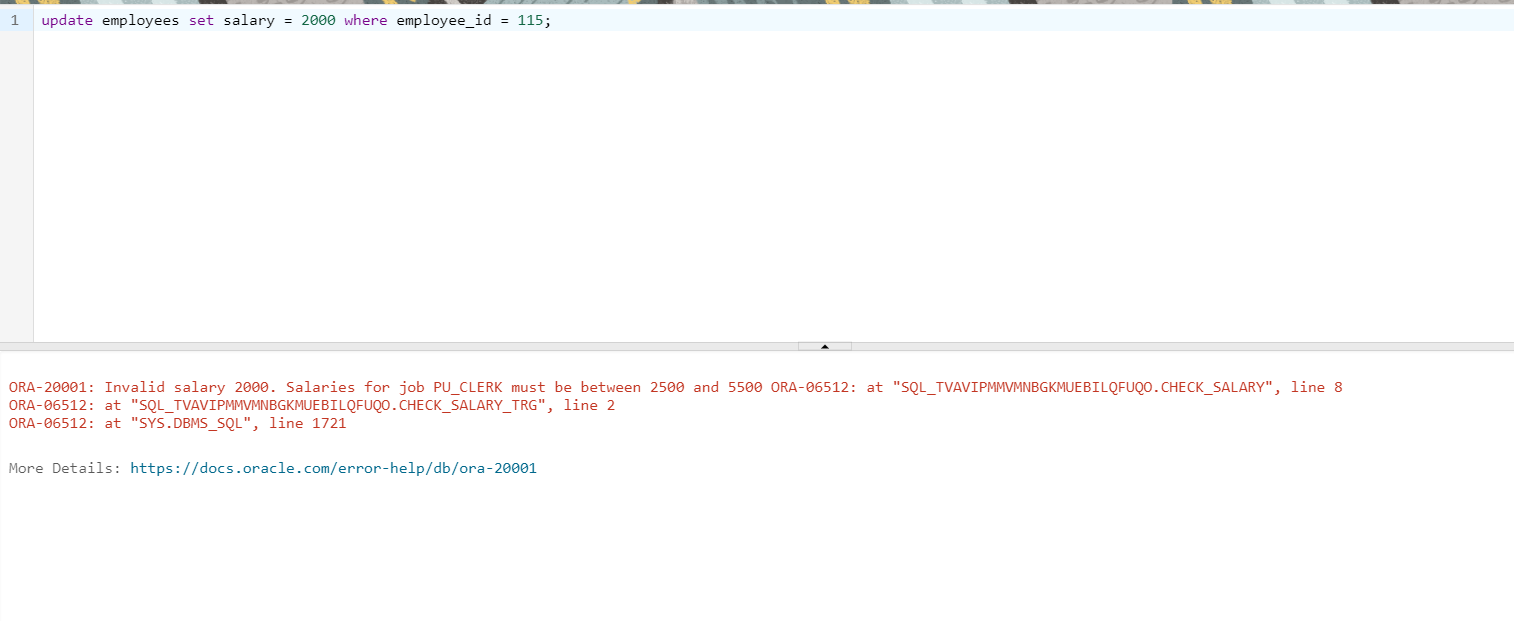
end;

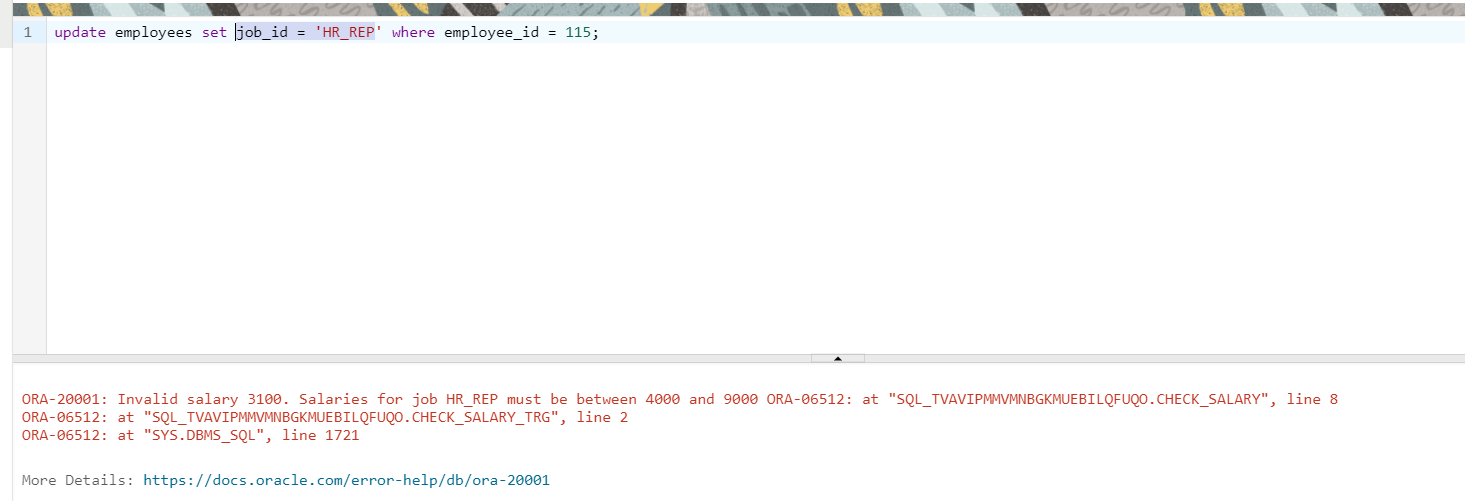
2. Test the CHECK\_SAL\_TRG using the following cases:

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

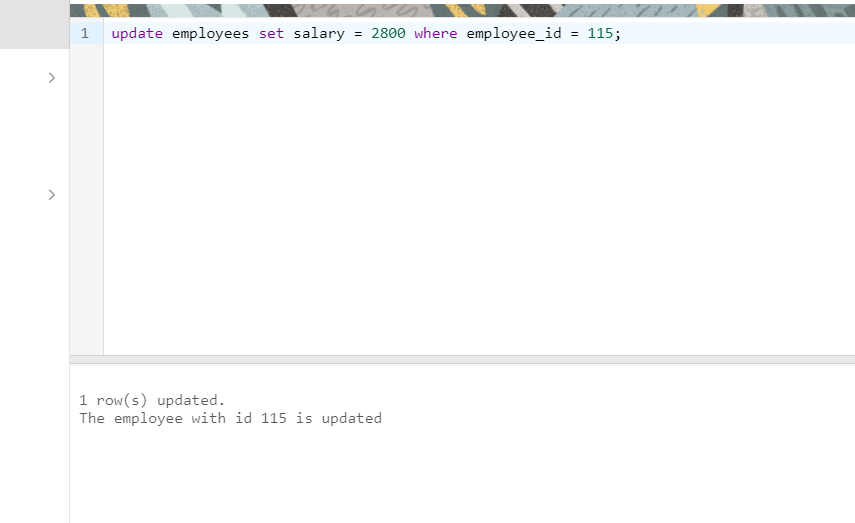


b. Update the salary of employee 115 to $2,000. In a separate update operation, change the employee job ID to HR\_REP. What happens in each case?





* + - 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.

create or replace trigger check\_salary\_trg

before insert or update on employees

for each row

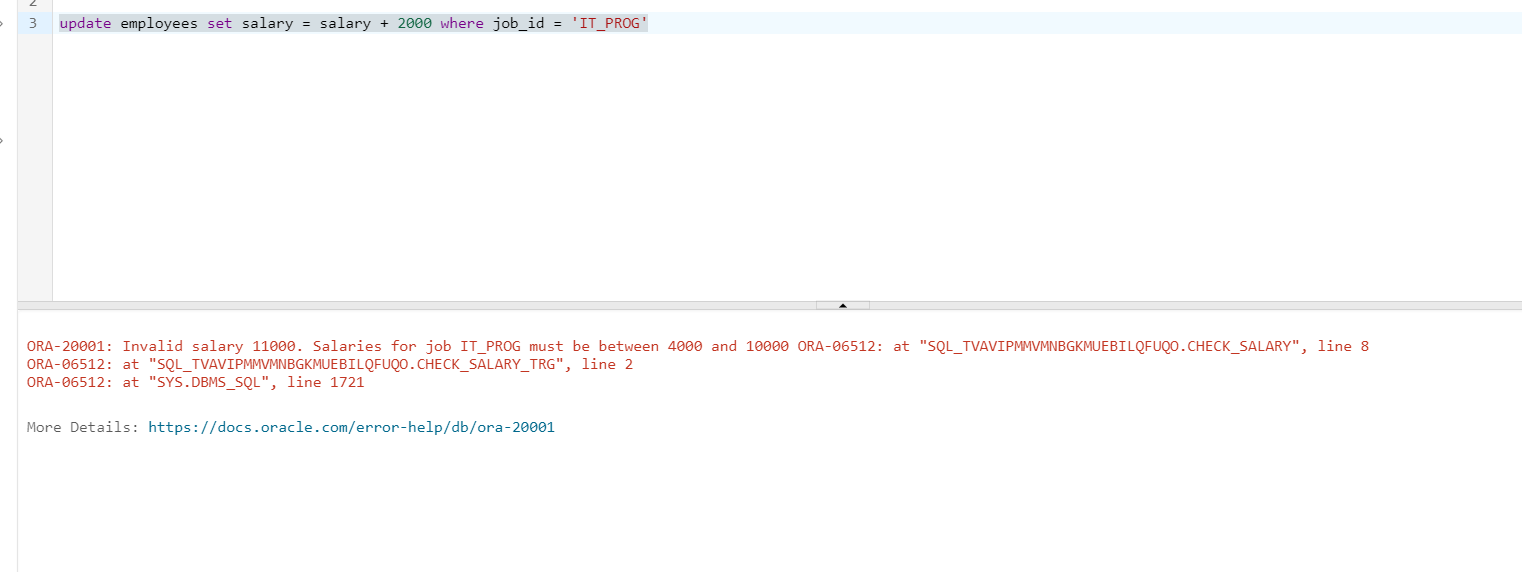
when (new.job\_id is not null and (new.job\_id != old.job\_id or new.salary != old.salary))

begin

check\_salary(:new.job\_id, :new.salary);

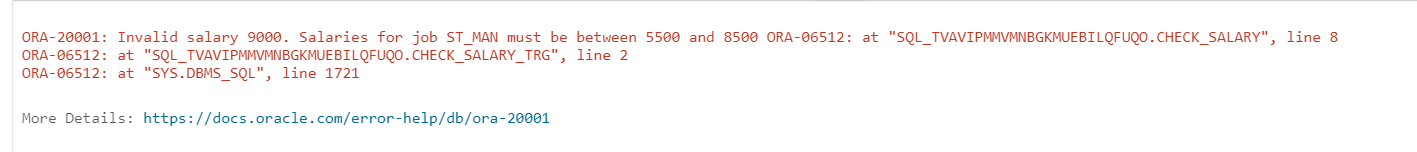
end;

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
2. 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?

