Database Management Lab 8: PL/pgSQL Record/Cursor and Trigger Definitions

While writing a PL/Pgsql function, we use quotation mark (‘) after the expression CREATE FUNCTION ... AS. If we want to use the quotation mark (‘) also in the body section of the function, it will cause a problem. There are several solutions of this problem. In the first solution, instead of using only one quote (‘), we can use two quatation marks consecutively (‘’) in the body block. Or as in the second solution we can use another delimeter (after the expression AS) to start the definition of the function. An alternative delimeter, which can be used to start the definition of the functions, is $$. At the end of the definition of the function, again we should use the same delimeter, $$.

If the body section of the function includes $$ characters, we should define another delimeter at the begining and at the end of the function. For example we can use bla_bla tag. These tags are case sensitive (although pl/pgsql is not a case-sensitive language). If we use TAG at the beginning of the function, we cannot use tag at the end of the function, we should use TAG also in the end.

Record / Cursor Definitions

- PL/pgSQL functions do not have to return variables with simple data types, they may return variables with composite datatypes (such as records). In this case, we should define the return type as record.
- If we use local variables with composite types in the functions, we can also define these local variables as record types.
- If a list is returned at the end of the function, we should define a **CURSOR** and return the result list by this cursor.

Record definition: CREATE TYPE my_record_type as (field1 type1, field2 type2, …); **Ex:** CREATE TYPE sum_prod AS (sum int, product int) We can use this record type for definition of the local variables in the functions or to return variables from the function: my_record_variable my_record_type;	**CURSOR definition:** cursor_name [[NO] SCROLL] CURSOR [(arguments)] FOR sql_query; **Ex:** my_cur CURSOR FOR select * from emp;
CURSOR usage: OPEN my_cursor; FETCH [direction { FROM | IN }] cursor INTO target; Target can be a “record” or “variable1, variable2, ..” MOVE [direction { FROM | IN }] cursor; CLOSE my_cursor;	**Some properties of the Cursors** If we define the cursor by SCROLL property, proceeding by the reverse order is possible for the cursor; If we define the cursor by NO SCROLL, we could not proceed in reverse order in the cursor.

Providing information and error messages to the users on the screen:

Expression RAISE can be used to give information messages to the users:

RAISE [level] ’format’ [, expression [, ...]] [USING option = expression [, ...]];
RAISE [level] condition_name [USING option = expression [, ...]];
RAISE [level] SQLSTATE ’sqlstate’ [USING option = expression [, ...]];
RAISE [level] USING option = expression [, ...];
RAISE ;

"Level" option illustrates the type of the message. It can be one of these: DEBUG, LOG, INFO, NOTICE, WARNING and EXCEPTION. The default value of the level is EXCEPTION. We can use NOTICE or INFO to give messages to the users.
Example: RAISE NOTICE 'Salary here is %', sal_variable;

Examples

1. Write a function which takes the ssn of an employee as an input parameter, and returns his/her name, name of his/her department and his/her salary. And also write these information on the screen. Call the function by using the SSN of an employee.
2. Write a function which takes department number as an input parameter and shows the names of the employees, who are working at this department, on the screen. Call the function for a particular department.
3. Write a function which takes the department number as input parameter and returns the avarage salary of the employees, who are working at this department. Do not use the function SUM() for the solution.

Answers

1. Before the definition of the function, let us define a record type which has 3 fields (name of the employee, name of the department of the employee, salary of the employee):

```
CREATE TYPE my_record AS (name_emp varchar(20), dep_name varchar(20), sal_emp numeric);
```

Function definition:

```
CREATE OR REPLACE FUNCTION example1 (eno employee.ssn%type) RETURNS my_record AS '
DECLARE
emprec my_record;
BEGIN
        select fname, dname, salary into emprec from employee e, department d where ssn = eno and e.dno = d.dnumber;
        raise notice "Name of the employee: %, name of his-her department: %, his-her salary: % TL. ", emprec.name_emp, emprec.dep_name, emprec.sal_emp ;
        return emprec;
END;
' LANGUAGE  'plpgsql';
```

Calling: select example1('123456789'); **Dropping:** DROP FUNCTION example1 (employee.ssn%type); drop type my_record;

2.

```
CREATE OR REPLACE FUNCTION example2 (dnum numeric) RETURNS void AS '
DECLARE
        emp_cur CURSOR FOR select fname, lname from employee where dno = dnum;
BEGIN
    FOR emp_rec IN emp_cur  LOOP
        RAISE INFO  "Employee name is % %", emp_rec.fname, emp_rec.lname;
    END LOOP;
END;
' LANGUAGE  'plpgsql';
```

Calling: select example2(6); **Dropping:** DROP FUNCTION example2 (numeric);
NOTE: We could write "RAISE NOTICE message;" instead of "RAISE INFO message" in the LOOP.

3.

```
CREATE OR REPLACE FUNCTION example3 (dnum numeric) RETURNS numeric AS '
DECLARE
        sum_salary numeric;
        emp_cur CURSOR FOR select salary from employee where dno = dnum;
BEGIN
        sum_salary := 0;
        FOR emp_rec IN emp_cur  LOOP
                sum_salary := sum_salary  + emp_rec.salary;
        END LOOP;
        RETURN toplam_maas;
END;
' LANGUAGE  'plpgsql';
```

Calling: select example3 (6); **Dropping:** DROP FUNCTION example3(numeric);

TRIGGER DEFINITION

Triggers are also stored into the database such as the functions. However triggers are not called by the main functions, they are induced automatically for specific cases in the database. DML commands (INSERT, UPDATE, DELETE) induce the triggers. Their definition in PL/pgSQL is:

CREATE TRIGGER *trigger_name* { **BEFORE** | **AFTER** } { ***event1*** [OR ***event2*** OR ...] } **ON** *table_name* **[FOR [EACH] { ROW | STATEMENT }]** [**WHEN** (*condition*)]
EXECUTE PROCEDURE *trigger_func_name(arguments*);

Return type of the trigger function should be TRIGGER. Example:

CREATE OR REPLACE FUNCTION trig_func () *RETURNS TRIGGER* AS '
BEGIN
Statements;
[RETURN [NULL | OLD | NEW];]
END;
' LANGUAGE 'plpgsql';

To give error messages in the trigger functions:

RAISE EXCEPTION 'Error message: you cannot do bla bla operation on bla bla table, etc...';

Part	Description	Possible values
Trigger timing	When should the trigger start? Before or after the DML command?	**Before/After**
Trigger event	DML command which induces the trigger	**Insert/Update/Delete**
Trigger type	Working number/type of the trigger body block	**Statement/Row**

Trigger tipi, trigger fonksiyonunun, bir SQL sorgusu için **sadece bir kez** mi, yoksa trigger olayından etkilenen **her bir satır** için mi çalışacağını belirler. Varsayılanı "FOR EACH **STATEMENT**"tır.

NEW: It is used in the body block of the trigger function. It is avaliable only for row-level triggers. It is a record type variable which involves the values of the new record of insert or update operations. Its value is NULL for the statement-level type triggers and for the Delete operations.
OLD: It is used in the body block of the trigger function. It is avaliable only for row-level triggers. It is a record type variable which involves the values of the old record of delete or update operations. Its value is NULL for the statement-level type triggers and for the Insert operations.

Trigger dropping:
DROP TRIGGER *trigger_func_name* **ON** *tablo_name* [CASCADE | RESTRICT]

CASCADE: Automatically drops the objects which depend on the trigger.
RESTRICT: If there are objects depend on trigger, trigger cannot be dropped. This is default one.

EXAMPLES

1. We can insert records into the table employee only in workdays and working hours.
2. Write the trigger that provides the change of dno values in the table employee, when anybody change the value of a dnumber in the department table.
3. Write the triger which does not allow increment of salary more than10% and decrement of it
4. Add a new column to the table department. Name of new column is total_salary, type of it is integer. Write a trigger which provides the properly change of column total_salary of the table department, when the salaries of the employees of corresponding department change.

ANSWERS

1. CREATE OR REPLACE FUNCTION trig_func_example1() RETURNS TRIGGER AS '
BEGIN
if (to_char(now(), "DY") IN ("SAT", "SUN") OR to_char(now(), "HH24") NOT between "08" and "18") then
RAISE EXCEPTION "You can insert records only in weekdays and working hours.";
RETURN NULL;
else RETURN NEW;
end if;
END;

```
' LANGUAGE  'plpgsql';
```
CREATE TRIGGER **example1** BEFORE insert ON employee FOR EACH ROW EXECUTE PROCEDURE **trig_func_example1 ();**

To test trigger: INSERT INTO employee (ssn, fname, lname) values('999000555', 'Ali', 'Veli')
Dropping trigger and function of the trigger:
First:DROP TRIGGER example1 on employee; **then:**DROP FUNCTION trig_func_example1()

2.
```
CREATE OR REPLACE FUNCTION trig_func_example2() RETURNS TRIGGER AS '
BEGIN
        update employee set dno = new.dnumber where dno = old.dnumber;
        RETURN NEW;
END;
' LANGUAGE  'plpgsql';
```

CREATE TRIGGER example2 AFTER update ON department
FOR EACH ROW EXECUTE PROCEDURE trig_func_example2();

Dropping trigger and function of the trigger:
First:DROP TRIGGER example2 on department; **then:**DROP FUNCTION trig_func_example2()

3.
```
CREATE OR REPLACE FUNCTION trig_func_ex3() RETURNS TRIGGER AS '
BEGIN
        if (old.salary > new.salary or new.salary>1.1*old.salary) then
                Raise exception "You cannot decrease the salary or increase it more than %%10.";
                Return old;
        Else    return new;
        end if;
END;
' LANGUAGE  'plpgsql';
```

CREATE TRIGGER ex3 BEFORE update ON employee FOR EACH ROW EXECUTE PROCEDURE trig_func_ex3();

To test: update employee set salary = salary * 0.9 where ssn='123456789';
First: DROP TRIGGER ex3 on employee; **then:** DROP FUNCTION trig_func_ex3();

4.
```
ALTER TABLE department ADD COLUMN total_salary INTEGER default 0;
CREATE OR REPLACE FUNCTION trig_func_ex4() RETURNS TRIGGER AS $ex4$
BEGIN
   if  (TG_OP = 'DELETE') then
         update department set total_salary=total_salary-old.salary where dnumber=old.dno;
   elsif (TG_OP = 'UPDATE') then
          update department set total_salary=total_salary-old.salary+new.salary where dnumber=old.dno;
   else
         update department set total_salary=total_salary+new.salary where dnumber=new.dno;
   end if;                  RETURN NEW;
END;
$ex4$ LANGUAGE  'plpgsql';
```

CREATE TRIGGER ex4 AFTER insert or update or delete ON employee FOR EACH ROW EXECUTE PROCEDURE trig_func_ex4();

First: DROP TRIGGER ex4 on employee; **then:** DROP FUNCTION trig_func_ex4();
Drop the new column from department: ALTER TABLE department **DROP COLUMN** total_salary;