Database lab 7 - PL/pgSQL Function Definitions

- There are four different procedural languages in the PostgreSQL database system:
  1) PL/pgSQL 2) PL/TcL 3)PL/Perl 4) PL/Phyton

- Advantages of the PL/pgSQL:
  - Functions and triggers can be created by using PL/pgSQL
  - Conditional expressions and operations that include the loops can be easily implemented
  - Complex calculations and queries (that cannot be performed by simple sql queries) can be implemented
  - PL/pgSQL provides users to write their own functions according to goal of the their study.
  - PL/pgSQL can use all of the datatypes and functions (such as min(), max(), etc) that exist in the standard SQL.

- A PL/pgSQL function does not have to return only one value. If anyone wants to get return values more than one, he/she should use **output**-type parameters. The parameters returned by the functions can be simple data types, or composite (such as records) data types. Besides functions can return nothing or they can return an instance such as a pointer of a result set.

- If a function returns nothing, we should write only "return" or "return void" at the end of the function.

For using the procedural language, we should firstly create the language as writing following in the editor:

"CREATE LANGUAGE plpgsql "

Function syntax of PL/pgSQL is given below: (Squared brackets, [ ], denote the optional cases.)

CREATE FUNCTION *function_name (parameter1 type, parameter2 type,..., [out] parameterN type)*
[RETURNS *datatype of the returned variable]* **AS [$$] ['']**
DECLARE
declarations;
BEGIN
expressions;
..
[RETURN] *[variable to be returned;]*
..
EXCEPTION
Situations except the regular cases;
END;
[$$] [''] LANGUAGE plpgsql;

Calling an existing function:
SELECT *function_name (values of the parameters);*
Dropping an existing function:
DROP FUNCTION *function_name (type_of_parameter1, type_of_parameter2, ..., type_of_parameterN);*

Different types of RETURN

For ending the running of a PL/pgSQL function, we should write "RETURN expression" at the end of the function. However if we want to change value of the several parameters in a function, or we make definitions with the type of "output" we do not use "return expression" or we only write "return".

Each function should have a "return expression" except the functions that use output parameters and functions that return nothing. Otherwise we get runtime errors.

Statements and loops, which can be used in Begin- End code blocks

<table>
<tr>
<td>Block structure:
[DECLARE]
BEGIN
….
[DECLARE]
BEGIN
…
END;

[EXCEPTION]
….
END;</td>
<td>Declare:
DECLARE
variable1 type :=
initial_value1;

Examples of variable declarations:
user_id integer;
quantity numeric(5);
url varchar;
my_var
tablename.columnname%TYPE;</td>
<td>If Statement:

IF condition THEN
Statements;
[ELSIF condition
THEN
Statements;]
[ELSE
statements;
]
END IF;</td>
<td>Case Statement:
CASE selector
WHEN
expression1
THEN satatement(s)1;
WHEN
expression2
THEN satatement(s)2;
….
WHEN
expressionN
THEN satatement(s)N;

[ELSE statement(s)N+1]
END;</td>
</tr>
<tr>
<td colspan="2">LOOP
Statement1;
….
EXIT [WHEN condition];
END LOOP;</td>
<td>WHILE condition LOOP
Statement1;
Statement2;
…
END LOOP;</td>
<td>FOR counter IN [REVERSE]
lower_limit..upper_limit LOOP
Statement1;
Statement2;
…
END LOOP;</td>
</tr>
<tr>
<td colspan="4">SELECT column1, column2, … column INTO var1, var2,… varN FROM table [WHERE]</td>
</tr>
</table>

Examples

1. Write a function, which gets two numeric parameters from the user and finds the sum of those parameters. Call the function with the parameters (22, 63).

2. Write a function to find the average salary of the employees, who is studying at the department, whose name is given as input parameter to the function.

3. Write a function to find the minimum and the maximum department number in the table “department”. Assign the value of the minimum number to the variable min_deptno, value of the maximum number to the variable max_deptno.

4. Write a function to find the number of the employees, who is working at the 6-th department. If the number of these employees is less than 10, raise the salary of these employees by 5%.

Answers

1. CREATE OR REPLACE FUNCTION example1 (num1 numeric, num2 numeric) RETURNS numeric AS '
DECLARE
total numeric;
BEGIN
total :=num1+num2;
return total;
END;
' LANGUAGE 'plpgsql';

NOT: After the AS and before the LANGUAGE, we can use the sign $$ instead of quotes (').
Calling the function: select example1 (22,63);

Solution without using Return (by using Out):

```
CREATE or replace FUNCTION example1 (num1 numeric, num2 numeric, out num3 numeric) AS '
BEGIN
        num3 :=num1+num2;
END;
' LANGUAGE  'plpgsql';
```

Dropping: DROP FUNCTION example1 (numeric, numeric)

2. CREATE OR REPLACE FUNCTION example2 (depname department.dname%type) RETURNS real AS '
DECLARE
 sal numeric;
 BEGIN
        SELECT AVG(salary) INTO sal FROM employee, department WHERE dno = dnumber AND dname = depname;
        RETURN sal;
 END;
' LANGUAGE 'plpgsql';

Calling: select example2('Sales'). **Dropping:** DROP FUNCTION example2 (department.dname%type)

3. CREATE OR REPLACE FUNCTION example3 (out min_deptno department.dnumber%type, out max_deptno department.dnumber%type) AS '
BEGIN
        Select min(dnumber), max(dnumber) INTO min_deptno, max_deptno from department;
END;
' LANGUAGE 'plpgsql';
Calling: select example3 (). **Dropping:** DROP FUNCTION example3 ()

4. CREATE OR REPLACE FUNCTION example4 () returns void AS '
DECLARE
num_worker numeric(3) := 0;
BEGIN
        Select count(*) into num_worker from employee where dno=6;
        IF (num_worker < 10) THEN
                Update employee SET salary=salary*1.05 where dno=6;
        END IF;
END;
' LANGUAGE 'plpgsql';
Calling: select example4(). **Dropping:** DROP FUNCTION example4()