Database Systems
PL/pgSql Functions

Week 7 – Lab

Functions

- A PostgreSQL function or a stored procedure is a set of SQL and procedural commands such as declarations, assignments, loops, flow-of-control etc. stored on the database server and can be involved using the SQL interface. And it is also known as PostgreSQL stored procedures.
- We can create PostgreSQL functions in serval languages, for example, SQL, PL/pgSQL, C, Python etc.
- It enables us to perform operations, which would generally take various commands and round trips in a function within the database.

Why?

- PL/pgSQL is easy to learn and simple to use.
- PL/pgSQL comes with PostgreSQL by default. The user-defined functions and stored procedures developed in PL/pgSQL can be used like any built-in functions and stored procedures.
- PL/pgSQL inherits all user-defined types, functions, and operators.
- PL/pgSQL has many features that allow you to develop complex functions and stored procedures.
- PL/pgSQL can be defined to be trusted by the PostgreSQL database server.

Functions

Syntax

```
CREATE [OR REPLACE] FUNCTION function_name (arguments)
RETURNS return_datatype
LANGUAGE plpgsql
AS $variable_name$
DECLARE
declaration;
[...] -- variable declaration
 BEGIN
< function_body >
[...]  -- logic
RETURN { variable_name | value }
END;
$$
```

```
-- Example
do $$
declare
   n integer:= 10;
   fib integer := 0;
   counter integer := 0 ;
   i integer := 0 ;
   j integer := 1 ;
begin
          if (n < 1) then
                    fib := 0 ;
          end if;
          loop
                    exit when counter = n ;
                    counter := counter + 1 ;
                    select j, i + j into i,j ;
          end loop;
          fib := i;
    raise notice '%', fib;
end; $$
```

For Loop (1/2)

For loop contains a counting variable which is not necessary to declare outside the for a loop. It can be declared in the for loop statement itself. This counting variable has START VALUE and an END VALUE as its range for which it will iterate.

Syntax

```
FOR [counting variable name] IN [REVERSE] [START
VALUE] .. [END VALUE] [BY step value] LOOP
[code/statements to repeat];
END LOOP;
```

For Loop (2/2)

```
CREATE OR REPLACE FUNCTION displayTable(int) RETURNS void AS $$
DECLARE
tableOf int:=$1;
BEGIN
FOR counter IN 1..10
LOOP
RAISE NOTICE '%', tableOf*counter;
END LOOP;
END;
$$ LANGUAGE plpgsql;
```

```
postgres=# select displayTable(5);
NOTICE:  5
NOTICE:  10
NOTICE:  15
NOTICE:  20
NOTICE:  25
NOTICE:  30
NOTICE:  35
NOTICE:  40
NOTICE:  45
NOTICE:  50
 displaytable
--------------

(1 row)
```

While Loop (1/2)

The PostgreSQL WHILE LOOP is used when we want to execute the same block of code statements several times. This continues execution of WHILE LOOP body until a condition defined in the WHILE LOOP evaluates to false.

Syntax

```
WHILE [condition] LOOP
[statements] END LOOP
```

While Loop (2/2)

```
1 1
2 2
3 3
4 4
5 5
6 6
```

```
DECLARE -- (1) Declare your variables
i_start INT := 0;
i_increment INT := 1;
i_current INT := 1;
i_end INT := 6;
BEGIN -- (2) Tell Postgres the scope of our program, nested in a BEGIN..END
container.
  WHILE i_current <= i_end -- (3) Define our loop within a LOOP..END LOOP
container.
        LOOP -- (4) Code to be executed. In this case, we are doing two things:
         -- (a) Incrementing our i_current variable.
         i_current := i_current + i_increment;
         -- (b) Printing the value of i_current.
         RAISE NOTICE i_current;    -- info tidbit: in most other forms of TSQL, we
                                    use "Print" instead of "Raise Notice".
        END LOOP;
END;
```

IF Statement

IF statements

- Simple IF statements
  - `IF condition THEN statement; END IF;`
- IF-THEN-ELSE statements
  -
```
IF condition THEN
    statements;
ELSE
    additional statements;
END IF;
```

Case Statement

```
SELECT last_name, job_id, salary,
CASE job_id
WHEN 'ACCOUNT' THEN 1.10*salary
WHEN 'IT_PROG' THEN 1.15*salary
WHEN 'SALES' THEN 1.20*salary
ELSE salary END "REVISED_SALARY" FROM employees;
```

Example For function on DVD rental

```
CREATE OR REPLACE FUNCTION get_actor_by_id(id integer)
 RETURNS TABLE(first_name VARCHAR(45), last_name VARCHAR(45)) AS
$$
BEGIN
 RETURN QUERY

 SELECT actor.first_name, actor.last_name
 FROM actor
 WHERE actor_id = id;

END; $$

LANGUAGE plpgsql;
```

```
select get_actor_by_id(55);
```

Exercise 1

- From DVD rental database Table **address** have column address as text (you want to convert address to Longitude/latitude and create new two columns in **address** table).
- Steps:
  - Create Function to retrieve addresses that contains “11” and city_id between 400-600.
  - Call the function from Python script and use geoPy to generate longitude and latitude and Create Query to update Address table with new values.
  - Any area that geoPy not recognize put 0,0 for both values.
  - Submit you python code/output query for Address Table/SQL Query with screenshots for Database function. (prefer **README.md** file)

Exercise 2

- In Customer Table we have less than 600 customers and you don't have backend developer to do paging.. Your job is to create function to retrieve customers order by address_id and it will take two parameters (start,end).
- For example: retrievecustomers(10,40)
  - 10 is customer number 10 in the query.
  - If start or end is negative number or more than 600 show an error message describe the error.
- Submit SQL query to create the function with example.

Useful resources

- https://www.linuxtopia.org/online_books/database_guides/Practical_PostgreSQL_database/PostgreSQL_x20238_002.htm
- https://www.postgresqltutorial.com/plpgsql-loop-statements/
- https://www.educba.com/postgresql-for-loop/?source=leftnav
- https://www.postgresqltutorial.com/plpgsql-for-loop/
- https://kb.objectrocket.com/postgresql/postgres-if-statement-1351

See you next week