Stored Procedures: Funções em Linguagem SQL

Professor
Fernando Martins Medeiros

O que é?

Em muitos SGDBs temos o conceito de Stored Procedures, programas desenvolvidos em uma determinada linguagem de script e armazenados no servidor, onde serão processados. No PostgreSQL, as Stored Procedures são conhecidas com o nome de Functions.

Tipos de funções

No **PostgreSQL** podemos ter três tipos de funções:

Funções em Linguagem SQL: As funções em SQL não possuem variáveis e estruturas de comando (*if*, *for* etc). Elas apenas consistem em uma lista de comandos SQL (SELECT, INSERT, DELETE ou UPDATE), devendo retornar, obrigatoriamente, um determinado valor. Assim, o último comando deve ser sempre um SELECT. Essas funções são carregadas juntamente com o serviço do PostgreSQL, não necessitando de nenhuma carga de módulo adicional.

Funções de Linguagens Procedurais: Esse tipo de função utiliza variáveis e estruturas de comandos, além de executar ações SQL. Na versão atual do PostgreSQL temos quatro tipos de linguagens procedurais: PL/PgSQL, PL/Tcl, PL/Perl e PL/Python. A Linguagem PL/PgSQL é a mais utilizada, pois é bem estruturada e fácil de aprender. As linguagens PL/Tcl, PL/Perl e PL/Python têm sintaxe semelhante às linguagens das quais elas herdam sua implementação. Sua utilização será explorada nas próximas edições da SQL Magazine.

Funções Externas: No PostgreSQL podemos utilizar funções desenvolvidas em uma linguagem externa, como C++. A vantagem é que passamos a contar com o poder de uma linguagem de programação completa, possibilitando a implementação de rotinas complexas no banco de dados. As funções devem ser empacotadas em bibliotecas compartilhadas que, por sua vez, devem ser registradas no SGBD.

O Comando Create Function

```
1   CREATE OR REPLACE FUNCTION incrementar(integer)
2   RETURNS integer AS
3   $$
4
5       SELECT $1 + 1;
6
7   $$
8   LANGUAGE sql;
9
10  SELECT incrementar(10);
```

O Comando Create Function

```
1   CREATE FUNCTION quantidade() RETURNS void AS $$
2   DECLARE
3       quantidade integer := 30;
4   BEGIN
5       RAISE NOTICE 'Aqui a quantidade é %', quantidade;
6       quantidade := 50;
7       RAISE NOTICE 'Aqui a quantidade é %', quantidade;
8   END;
9
10  $$ LANGUAGE plpgsql;
11
12  SELECT quantidade();
```

O Comando Create Function

```
1   CREATE FUNCTION quantidade() RETURNS void AS $$
2   DECLARE
3       quantidade integer := 30;
4   BEGIN
5       RAISE NOTICE 'Aqui a quantidade é %', quantidade;
6       quantidade := 50;
7       RAISE NOTICE 'Aqui a quantidade é %', quantidade;
8   END;
9
10  $$ LANGUAGE plpgsql;
11
12  SELECT quantidade();
```

Retorno de valores

```
1  CREATE FUNCTION quantidade() RETURNS void AS $$
2  DECLARE
3     quantidade integer := 30;
4  BEGIN
5     RAISE NOTICE 'Aqui a quantidade é %', quantidade;
6     quantidade := 50;
7     RAISE NOTICE 'Aqui a quantidade é %', quantidade;
8  END;
9
10 $$ LANGUAGE plpgsql;
11
12 SELECT quantidade();
```

```
1  CREATE OR REPLACE FUNCTION incrementar(integer)
2  RETURNS integer AS
3  $$
4
5      SELECT $1 + 1;
6
7  $$
8  LANGUAGE sql;
9
10 SELECT incrementar(10);
```

Como usar?

```sql
SELECT * FROM function_name(val1,val2);
```

```sql
SELECT function_name(val1,val2);
```

```sql
SELECT function_name(val1,val2) FROM table;
```

Estruturas condicionais

```
IF expressão-boleana THEN
 comandos
 ELSE
 comandos
 END IF;
```

Como exemplo, tendo V_COUNT como um número inteiro, temos:

```
IF v_count > 0 THEN
 INSERT INTO users_count(count)
 VALUES(v_count);
 return ''v'';
 ELSE
 return ''f'';
 END IF;
```