Funções, Procedures e Triggers

Programando Diretamente no Banco de Dados.

O que são?

O que são Funções?

- São rotinas e/ou subrotinas programadas com comportamentos específicos.
- Podem ou não receber parâmetros e podem ou não retornar algum resultado, limitando-se sempre em um e **somente um** valor retornado.
- Geralmente são mais objetivas e curtas.

O que são Procedures?

- Também são rotinas e/ou subrotinas programadas com comportamentos específicos.
- Conceitualmente diferenciam-se das funções normais por retornar mais de um valor.
- Em alguns bancos são criadas com sintaxes específicas, não é o caso do postgres onde utilizamos a mesma sintaxe da criação de funções com apenas um retorno.
- Pode acontecer de ser mais longas (ter mais comportamento) para atingir o resultado final.

O que são Triggers?

- Como o nome já sugere, são gatilhos que podem ser disparados sempre que alguma ação for executada em uma tabela no banco de dados.
- Em alguns bancos são criadas com sintaxes específicas, não é o caso do postgres onde utilizamos a mesma sintaxe da criação de funções para criar uma função gatilhada e vinculamos um gatilho a esta função.

Stored Functions

Como criar Funções e Procedures no PostgreSQL

Exemplos

Exemplo 1

Função c/ parâmetros e retorno

Exemplo 1 (função c/ parâmetros e retorno)

```
create or replace function somar(arg1 int, arg2 int)
    returns int
    language plpgsql as
$$
begin
    return arg1 + arg2;
end;
$$
```

Exemplo 1 (função c/ parâmetros e retorno)

- Para criar uma function usamos os comando DDL **CREATE.**
- Para substituir uma função já existente usamos o **REPLACE.**
- É uma prática comum o uso o **CREATE or REPLACE**
- O recurso que será criado no banco é um recurso chamado **FUNCTION.**
- Toda função deverá ter um nome (no exemplo nossa função se chama **somar**)
- Ela poderá ou não ter parâmetros/argumentos.
- O tipos que usamos em funções são os mesmos tipos suportados em criação de atributos em tabelas.

Exemplo 1 (função c/ parâmetros e retorno)

- Nós definimos o tipo do retorno através da instrução **returns <tipo_desejado>.**
- Podemos escrever nossas funções no postgres em mais de uma linguagem, e essa linguagem deve ser especificada através da instrução **language.**
- A instrução **as** indica o “como” as funções serão criadas.
- O duplo cifrão ($$) indica o início e fim do escopo da função (similar ao { } do java)
- O resultado final da função é retornado através da instrução **return.** (Não confundir com o **returns** que determina o tipo do retorno).

Languages

Languages

- c
  - podemos escrever o conteúdo de uma função usando a linguagem de programação C.
- sql
  - comandos que já vimos até agora (SELECT, INSERT, etc)
- **plpgsql**
  - além dos comandos que já vimos até agora.
  - suporte a recursos de programação (if, while, for, etc), também conhecida como estrutura de controle.
  - possibilita “programar” no banco de dados.

Estrutura de Controle

Estrutura de Controle Condicionais

Estrutura de Controle Condicionais

- IF-THEN, IF-THEN-ELSE e IF-THEN-ELSEIF
- Simple CASE e Searched CASE
- LOOP e EXIT
- WHILE
- FOR (int variant) e FOR (sobre resultado de consultas)

IF-THEN

IF *boolean-expression* THEN

statements

END IF;

IF-THEN-ELSE

IF *boolean-expression* THEN

statements

ELSE

statements

END IF;

IF-THEN-ELSEIF

IF *boolean-expression* THEN

statements

ELSIF *boolean-expression* THEN

statements

ELSE

statements

END IF;

Simple CASE

```
CASE search-expression
 WHEN expression THEN
      statements
 WHEN expression THEN
      statements
 ELSE
      statements
END CASE;
```

Searched CASE

```
CASE
 WHEN boolean-expression THEN
      statements
 WHEN boolean-expression THEN
      statements
 ELSE
      statements
END CASE;
```

LOOP e EXIT

```
LOOP
    IF count > 0 THEN
        EXIT; -- exit loop
    END IF;
END LOOP;
```

WHILE

WHILE *boolean-expression* LOOP

statements

END LOOP

FOR (Integer variant)

```
FOR i IN 1..10 LOOP
 statements
END LOOP;
```

FOR (Sobre resultados de consulta)

FOR *target* IN *query* LOOP

statements

END LOOP

Exemplo 2

Função c/ declaração de variáveis

Exemplo 2 (função c/ declaração de variáveis)

```
create or replace function somar(arg1 int, arg2 int)
   returns int
   language plpgsql as
$$
declare
   bonus int = 10;
begin
   return arg1 + arg2 + bonus;
end;
$$
```

Exemplo 2 (função c/ declaração de variáveis)

- Uso da instrução **declare** antes do **begin** para definição de variáveis.
- A declaração de uma variável se dá pelo seu nome primeiramente logo seguido do seu tipo.
- Uma variável pode ser inicializada no momento da sua declaração, bem como pode ser inicializada em outro momento dentro da execução da função.

Exemplo 3

Função sem retorno

Exemplo 3 (função sem retorno)

```
create or replace function somar(arg1 int, arg2 int)
    returns void
    language plpgsql as
$$
declare
    resultado int;
begin
    resultado = arg1 + arg2 + bonus;
end;
$$
```

Exemplo 3 (função sem retorno)

- Remoção do uso do **return** ao final da função para retornar o valor.
- Uso do **void** na declaração do tipo do retorno (**returns**), para indicar que a função não irá retornar nenhum valor.

Exemplo 4

Função com múltiplos retornos (conhecidas também como procedures)

Exemplo 4 (função com múltiplo retorno)

```
create or replace function somar(in arg1 int, in arg2
                int, out resultado int, out bonus int)
    returns record
    language plpgsql as
$$
begin
    bonus = 10;
    resultado = arg1 + arg2 + bonus;
end;
$$
```

Exemplo 4 (função com múltiplo retorno)

- Surgimento dos argumentos **IN** e **OUT** para indicar quais parâmetros são de entrada (IN) e quais são de saída (OUT)
- Definição do tipo **record** como o **returns** da função, que irá permitir que o múltiplo retorno funcione corretamente.
- Não é necessário o uso do **return** ao final da função para retornar os valores, pois ao atribuí-los ao parâmetro do tipo **OUT,** a função que executou receberá o valor dos mesmos através desses parâmetros.

Stored Triggers

Como criar Funções ‘Gatilhadas’ e Dispara-las no PostgreSQL

Exemplos

Stored Triggers (Exemplo)

```
create or replace function somar()
    returns trigger
    language plpgsql as
$$
begin
    -- seu comportamento aqui
    return null;
end;
$$;
```

Stored Triggers (Exemplo)

- Uso obrigatório do **returns** do tipo **trigger**, isso irá indicar que é uma função que pode ser “gatilhada”.
- O retorno com o **return** é obrigatório, podendo ser **null** quando a função for chamada por processos que não precisem do seu valor, ou então um resultado válido que será utilizado, por exemplo, para inserção na tabela do banco de dados.

Stored Triggers (Exemplo)

```
create trigger somar trigger insert
   after insert on minha_tabela
 for each statement
     execute procedure somar();
```

Stored Triggers (Exemplo)

- A função por si só não será gatilhada é necessário criar o gatilho (trigger) e definir quando ele será executado e qual tabela será observada as ações para o gatilho ser disparado.
- Existem duas possibilidades de **momentos: after** (após) e **before** (antes).
- Definido o momento, é necessário definir a ação observada, podendo ser **insert, update ou delete.**
- Por fim é necessário especificar se será disparado para cada nova linha adicionada ou para cada comando executado. Lembrando que posso ter um **insert** único que ensira mais de uma linha em uma tabela (por exemplo).

Live Code!

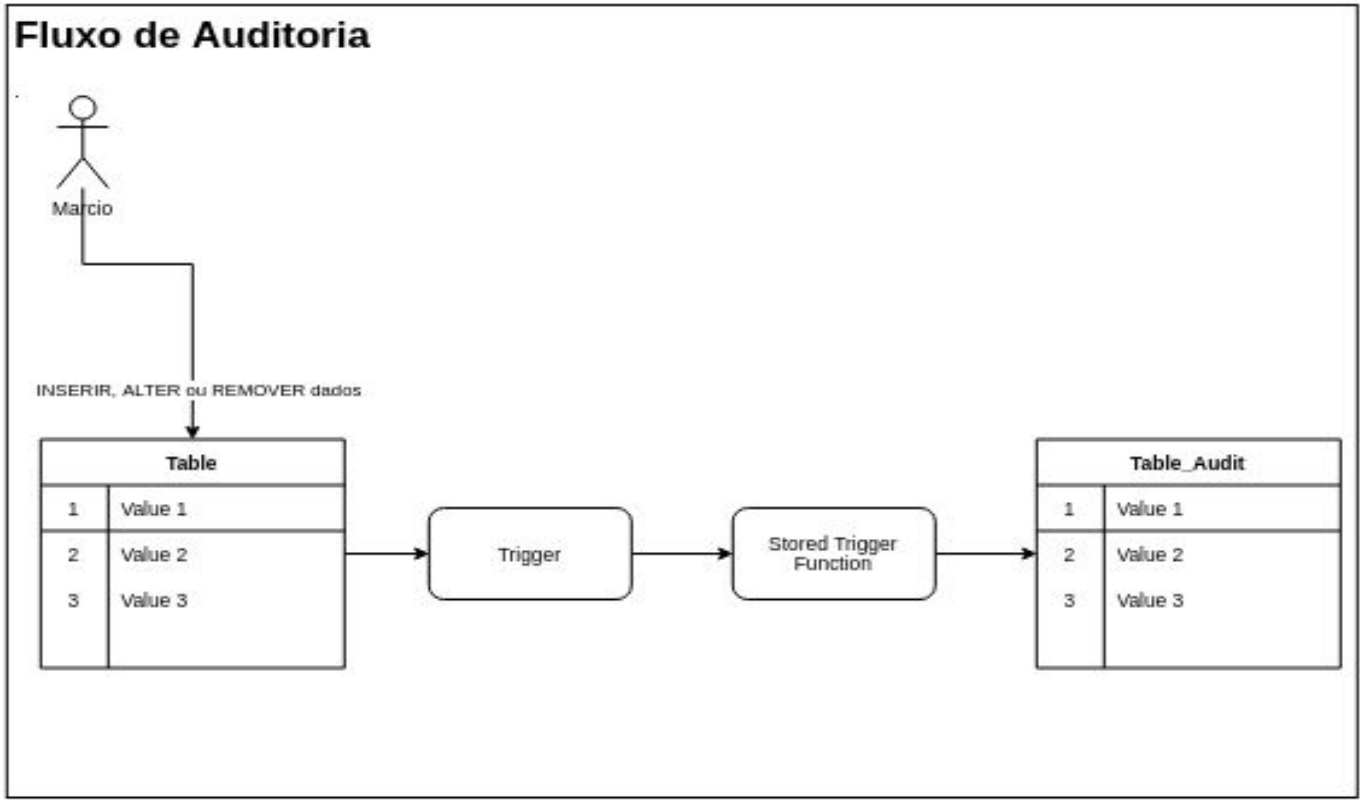
Vamos criar um mecanismo de auditoria das nossas tabelas no banco de dados.

Auditoria em banco de dados

A auditoria em banco de dados ajuda a manter a conformidade regulatória, entender a atividade do banco de dados e obter informações sobre discrepâncias e anomalias. Isso é essencial para identificar possíveis preocupações para o negócio ou suspeitas de violações de segurança nos sistemas da companhia.

(Fonte: https://www.mxm.com.br/blog/importancia-de-realizar-auditoria-em-banco-de-dados/)

Auditoria em banco de dados



Auditoria em banco de dados (DER)