

TRABALHANDO COM TRIGGERS NO POSTGRESQL

PROFESSOR: ADRIANO GOMES

DISCIPLINA: FUNDAMENTOS DE BANCO DE DADOS



UMA VISÃO GERAL SOBRE AS TRIGGERS

- TRIGGERS, EM TERMOS DE BANCO DE DADOS, SÃO AS OPERAÇÕES REALIZADAS DE FORMA ESPONTÂNEA PARA EVENTOS ESPECÍFICOS. QUANDO TRATAMOS DOS EVENTOS, ESTES PODEM SER TANTO UM INSERT QUANTO UM UPDATE, OU MESMO UM DELETE. ASSIM, PODEMOS DEFINIR DETERMINADAS OPERAÇÕES QUE SERÃO REALIZADAS SEMPRE QUE O EVENTO OCORRER.
- QUANDO NOS REFERIRMOS A UMA OPERAÇÃO COM UMA TRIGGER, ESTA É CONHECIDA POR TRIGGER DE FUNÇÃO OU TRIGGER FUNCTION.
- LEMBRE-SE QUE TRIGGER E FUNÇÃO DE TRIGGER SÃO DUAS COISAS DIFERENTES, ONDE A PRIMEIRA PODE SER CRIADA UTILIZANDO A INSTRUÇÃO CREATE TRIGGER, ENQUANTO QUE A ÚLTIMA É DEFINIDA PELO COMANDO CREATE FUNCTION. EM LINHAS GERAIS, COM AS TRIGGERS DEFINIMOS QUAL TAREFA EXECUTAR, E COM AS TRIGGERS DE FUNÇÃO DEFINIMOS COMO ESSA TAREFA SERÁ REALIZADA.

CRIAR DE FORMA ABSTRATA UMA TRIGGER FUNCTION E UMA TRIGGER

- AO TEMOS UMA GRANDE QUANTIDADE DE ACESSOS AO BANCO DE DADOS POR MÚLTIPAS APLICAÇÕES, A UTILIZAÇÃO DAS TRIGGERS É DE GRANDE UTILIDADE, E COM ISSO, PODEMOS MANTER A INTEGRIDADE DE DADOS COMPLEXOS, ALÉM DE PODERMOS ACOMPANHAR AS MUDANÇAS OU O LOG A CADA MODIFICAÇÃO OCORRIDA NOS DADOS PRESENTES NUMA TABELA.
- AS TRIGGERS FUNCTIONS PODEM SER DEFINIDAS EM LINGUAGENS COMPATÍVEIS AO POSTGRESQL, COMO PL/PGSQL, PL/PYTHON, PL/JAVA DENTRE OUTROS.

CRIAÇÃO DA ESTRUTURA DE UMA TRIGGER FUNCTION

```
332  /* Criação da estrutura de uma trigger function*/
333
334  CREATE OR REPLACE FUNCTION nome_funcao_trigger
335  RETURNS trigger AS $ExemploFuncaoFBD$
336  BEGIN
337  /* Aqui definimos nossos códigos.*/
338  RETURN NEW;
339  END;
340 $ExemploFuncaoFBD
341
```

SINTAXE DE UMA TRIGGER SIMPLES

```
343 /* Sintaxe de uma trigger simples*/
344
345 CREATE [ CONSTRAINT ] TRIGGER NAME { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
346 ON nome_da_tabela
347 [ FROM referencia_nome_da_tabela ]
348 [ NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY DEFERRED } ]
349 [ FOR [ EACH ] { ROW | STATEMENT } ]
350 [ WHEN ( condition ) ]
351 EXECUTE PROCEDURE nome_funcao ( arguments )
352 -- Quando um evento for declarado com:
353 INSERT
354 UPDATE [ OF column_NAME [, ... ] ]
355 DELETE
356 TRUNCATE
```

OBSERVAÇÕES

- REPARE QUE UMA TRIGGER FUNCTION É, NA REALIDADE, UMA FUNÇÃO NO POSTGRESQL, MAS COM A DIFERENÇA DE QUE ELA NÃO RECEBE ARGUMENTOS, E SIM UMA ESTRUTURA DE DADOS ESPECIAL CHAMADA DE TRIGGERDATA. REPARE TAMBÉM QUE O SEU TIPO DE RETORNO É A TRIGGER, ONDE ELA É CHAMADA AUTOMATICAMENTE NO MOMENTO DA OCORRÊNCIA DOS EVENTOS (QUE PODEM SER INSERT, UPDATE, DELETE OU TRUNCATE).
- COM O POSTGRESQL TEMOS DOIS TIPOS DE TRIGGER DISPONÍVEIS: TRIGGER DE NÍVEL DE LINHA (ROW-LEVEL TRIGGER) E A TRIGGER A NÍVEL DE INSTRUÇÃO (STATEMENT LEVEL TRIGGER). AMBOS SÃO ESPECIFICADOS COM A UTILIZAÇÃO DAS CLÁUSULAS FOR EACH ROW (NÍVEL GATILHO DE LINHA) E FOR EACH STATEMENT, RESPECTIVAMENTE.

QUANDO UTILIZAR?

- A UTILIZAÇÃO DELAS PODE SER DEFINIDA DE ACORDO COM A QUANTIDADE DE VEZES QUE A TRIGGER DEVERÁ SER EXECUTADA. POR EXEMPLO, SE UMA INSTRUÇÃO UPDATE FOR EXECUTADA, E ESTA AFETAR SEIS LINHAS, TEMOS QUE A TRIGGER DE NÍVEL DE LINHA SERÁ EXECUTADA SEIS VEZES, ENQUANTO QUE A TRIGGER A NÍVEL DE INSTRUÇÃO SERÁ CHAMADA APENAS UMA VEZ POR INSTRUÇÃO SQL.

QUANDO UTILIZAR?

- QUANDO UTILIZAMOS TRIGGERS PODEMOS CONECTÁ-LAS TANTO A TABELAS QUANTO A VIEWS, DE FORMA QUE AS TRIGGERS SÃO EXECUTADAS PARA AS TABELAS EM DUAS SITUAÇÕES: BEFORE E AFTER, PARA QUALQUER UMA DAS INSTRUÇÕES DML (INSERT, UPDATE, DELETE), ALÉM DE TAMBÉM POSSIBILITAR A SUA EXECUÇÃO UTILIZANDO A DECLARAÇÃO TRUNCATE.

QUANDO UTILIZAR?

- QUANDO TEMOS A TRIGGER DEFINIDA COM A INSTRUÇÃO INSTEAD OF PODEMOS UTILIZAR AS DML'S PARA AS VIEWS.
- AS TRIGGERS SERÃO DISPARADAS ANTES OU DEPOIS DAS INSTRUÇÕES DML, MAS PODEM SER DEFINIDAS APENAS A NÍVEL DE INSTRUÇÃO.
- JÁ QUANDO UTILIZAMOS O INSTEAD OF NAS INSTRUÇÕES DML, PODEMOS EXECUTÁ-LAS APENAS A NÍVEL DE LINHA.

PARÂMETROS

- COM RELAÇÃO AOS DEMAIS PARÂMETROS, TEMOS O NAME, QUE É UTILIZADO PARA ATRIBUIRMOS UM NOME PARA A TRIGGER, O QUAL DEVE SER DISTINTO DAS DEMAIS TRIGGERS CRIADAS PARA A MESMA TABELA. A INSTRUÇÃO TABLE_NAME APRESENTA O NOME DA TABELA EM USO.
- QUANTO AOS EVENTOS (EVENTS), ESTES PODEM SER INSERT, UPDATE, DELETE OU TRUNCATE, OS QUAIS ESPECIFICAM O EVENTO QUE IRÁ DISPARAR A TRIGGER.

EXPRESSÃO

- A EXPRESSÃO CONDITION É UMA EXPRESSÃO BOOLEANA QUE DETERMINA SE A TRIGGER FUNCTION SERÁ EXECUTADA. SE A CONDIÇÃO WHEN FOR ESPECIFICADA, A FUNÇÃO SERÁ CHAMADA SE A CONDIÇÃO RETORNAR TRUE. ALÉM DISSO, ELA PODE SER REFERIDA A COLUNAS QUE CONTENHAM OS VALORES ANTIGOS E SE QUER PASSAR OS NOVOS VALORES. PARA ISSO SÃO UTILIZADAS AS INSTRUÇÕES OLD.COLUMN_NAME OU NEW.COLUMN_NAME, RESPECTIVAMENTE. LEMBRE-SE QUE AS FUNCTION_NAMES SÃO FUNÇÕES FORNECIDAS PELOS USUÁRIOS.

ARGUMENTOS

- POR ÚLTIMO, TEMOS OS ARGUMENTS, QUE SÃO LISTAS OPCIONAIS DE ARGUMENTOS SEPARADOS POR VÍRGULAS QUE PODEM SER FORNECIDOS PARA A FUNÇÃO QUANDO A TRIGGER FOR EXECUTADA.

DEMONSTRAÇÃO

- PARA DEMONSTRARMOS O PROCEDIMENTO DE CRIAÇÃO DAS TRIGGERS E SUA UTILIZAÇÃO CRIAREMOS ALGUNS EXEMPLOS SIMPLES.
- PARA ISSO, CRIAREMOS UMA NOVA BASE DE DADOS, A QUAL CHAMAREMOS DE **FBDUF**C E EM SEGUIDA CRIAREMOS UMA TABELA FUNCIONARIOS QUE CONTERÁ OS CAMPOS DA LISTAGEM 1. NESTE NOSSO EXEMPLO QUEREMOS MANTER ATUALIZADOS TODOS OS REGISTROS ADICIONADOS PARA UMA POSSÍVEL FUNCIONARIOS_AUDITORIA.

LISTAGEM 1

```
357  
358 /*Criando a tabela de testes Funcionarios*/  
359 CREATE TABLE funcionarios (  
360     nome character varying(100) NOT NULL,  
361     email character varying(200) NOT NULL,  
362     telefone character(14) NOT NULL,  
363     profissao character varying(150) NOT NULL,  
364     endereco character varying(100) NOT NULL,  
365     salario real  
366 );|
```

DEMONSTRAÇÃO

- APÓS A CRIAÇÃO DA NOSSA TABELA PRINCIPAL CRIAREMOS UMA NOVA TABELA COM O NOME DE FUNCIONARIOS_FUNCIONARIOS_AUDITORIA, QUE SERÁ RESPONSÁVEL POR MANTER O HISTÓRICO DAS ALTERAÇÕES REALIZADAS NOS REGISTROS, COMO PODEMOS VER NA LISTAGEM 2.

LISTAGEM 2

```
368 /*Criação da tabela funcionarios_funcionarios_auditoria*/
369 CREATE TABLE funcionarios_funcionarios_auditoria (
370     codigo_func INT NOT NULL,
371     data_alteracao TEXT NOT NULL
372 );
373 |
```

- OBSERVE QUE HÁ APENAS DOIS CAMPOS: O CÓDIGO DO FUNCIONÁRIO E A DATA DA ALTERAÇÃO/CRIAÇÃO DO REGISTRO, QUE RECEBERÁ UMA DATA NO FORMATO TIMESTAMP NO MOMENTO EM QUE O REGISTRO FOR CRIADO NA TABELA DE FUNCIONÁRIOS.

DEMONSTRAÇÃO

- PARA DARMOS SEGUIMENTO AOS NOSSOS TESTES IREMOS INSERIR ALGUNS DADOS, COMO PODEMOS VER NA LISTAGEM 3.

LISTAGEM 3

```
/*Inserindo dados na tabela de funcionarios*/
INSERT INTO FUNCIONARIOS (codigo, nome, email, telefone, profissao, endereco, salario)
VALUES (1, 'Edson Dionisio', 'edson.dionisio@gmail.com', '(88)997402800', 'Desenvolvedor Web',
2000.00);
INSERT INTO FUNCIONARIOS (codigo, nome, email, telefone, profissao, endereco, salario)
VALUES (2, 'Marilia Késsia', 'mkessia.dionisio@gmail.com', '(88)997402844', 'Analista de
desenvolvimento', 'rua testes', 6000.00);
INSERT INTO FUNCIONARIOS (codigo, nome, email, telefone, profissao, endereco, salario)
VALUES (3, 'Caroline França', 'carol@gmail.com', '(88)997402800', 'Analista de testes', 'rua
testes', 2500.00);
INSERT INTO FUNCIONARIOS (codigo, nome, email, telefone, profissao, endereco, salario)
VALUES (4, 'João da Silva', 'joao@gmail.com', '(88)997401654', 'Analista de finanças',
'rua testes', 8000.00);
INSERT INTO FUNCIONARIOS (codigo, nome, email, telefone, profissao, endereco, salario)
VALUES (5, 'Maria das Dores', 'maria@gmail.com', '(88)997407845', 'Secretaria',
'rua testes', 1800.00);
```

DEMONSTRAÇÃO

- AS TRIGGERS FUNCTIONS RECEBEM, ATRAVÉS DE UMA ENTRADA ESPECIAL, UMA ESTRUTURA TRIGGERDATA, QUE POSSUI UM CONJUNTO DE VARIÁVEIS LOCAIS QUE PODEMOS USAR NAS NOSSAS TRIGGERS FUNCTIONS. DENTRE AS VARIÁVEIS PRESENTES NESTA ESTRUTURA TEMOS AS VARIÁVEIS OLD E NEW, ALÉM DE OUTRAS QUE COMEÇAM COM O PREFIXO TG_, COMO TG_TABLE_NAME.

DEMONSTRAÇÃO

- A VARIÁVEL NEW É DO TIPO RECORD E CONTÉM UMA NOVA LINHA A SER ARMAZENADA COM BASE NOS COMANDOS INSERT/UPDATE DAS TRIGGERS A NÍVEL DE LINHA.
- JÁ A VARIÁVEL OLD TAMBÉM É DO TIPO RECORD E ARMAZENA A LINHA ANTIGA QUANDO UTILIZADA COM OS COMANDOS UPDATE/DELETE NAS TRIGGERS DE LINHA.

DEMONSTRAÇÃO

- APÓS A CRIAÇÃO DAS NOSSAS TABELAS DEFINIREMOS UMA TRIGGER FUNCTION, A QUAL CHAMAREMOS DE FUNCIONARIO_LOG_FUNC, E SERÁ RESPONSÁVEL POR REGISTRAR AS ALTERAÇÕES FEITAS NA TABELA DE FUNCIONARIOS_AUDITORIA DEPOIS DE UMA OPERAÇÃO DE INSERT NA TABELA FUNCIONÁRIOS, COMO APRESENTADA PELA LISTAGEM 4.

LISTAGEM 4

```
392 /*Criação da trigger function funcionario_log_func*/
393 CREATE OR REPLACE FUNCTION funcionario_log_func()
394 RETURNS trigger AS $teste_trigger$
395 BEGIN
396 INSERT INTO funcionario_funcionarios_auditoria
397 (log_id, data_criacao)
398 VALUES
399 (new.codigo_func, current_timestamp);
400 RETURN NEW;
401 END;
402 $teste_trigger|
```

OBSERVAÇÃO

- AO INSERIRMOS UM NOVO REGISTRO NA NOSSA TABELA DE FUNCIONÁRIOS, PODEMOS VER QUE UM NOVO REGISTRO FOI CRIADO TAMBÉM NA TABELA DE FUNCIONARIOS_AUDITORIA.
- PARA UM EXEMPLO UM POUCO MAIS COMPLEXO CRIAREMOS UMA TRIGGER CONTENDO AS TRÊS OPERAÇÕES DML'S CONTIDAS NUMA MESMA TRIGGER FUNCTION.
- PARA ISSO, REALIZAREMOS INICIALMENTE UMA ALTERAÇÃO NA NOSSA TABELA FUNCIONARIOS_AUDITORIA, ONDE ADICIONAREMOS UMA NOVA COLUNA OPERAÇÃO_REALIZADA PARA ARMAZENAR O NOME DA OPERAÇÃO REALIZADA.

LISTAGEM 5

ANTES DE PROSSEGUIRMOS, EXCLuiremos a ABELA FUNCIONARIOS_AUDITORIA UTILIZANDO O COMANDO DROP, COMO PODEMOS VER A SEGUIR:

```
404 /*Excluir a tabela funcionarios_auditoria*/
405
406 Drop table funcionarios_auditoria cascade;
407
```

APÓS A EXCLUSÃO DA TABELA, A CRIAREMOS NOVAMENTE, MAS CONTENDO AS SEGUINTEs COLUNAS DA LISTAGEM 6.

```
408 /*Criando a tabela funcionarios_auditoria*/
409
410 CREATE TABLE funcionarios_auditoria
411 (
412 log_id INT NOT NULL,
413 data_criacao TEXT NOT NULL,
414 operacao_realizada CHARACTER VARYING
415 );
```

LISTAGEM 7

NESTE MOMENTO PODEMOS UTILIZAR O COMANDO APRESENTADO NA LISTAGEM 7 PARA CRIAR OU RECRIAR A NOSSA TRIGGER FUNCTION.

```
418 /*Recriando a trigger function*/
419
420 CREATE OR REPLACE FUNCTION funcionario_log_function()
421 RETURNS trigger AS $BODY$
422 BEGIN
423 -- Aqui temos um bloco IF que confirmará o tipo de operação.
424 IF (TG_OP = 'INSERT') THEN
425 INSERT INTO funcionarios_auditoria (log_id, data_criacao, operacao_realizada)
426 VALUES (new.codigo_func, current_timestamp, 'Operação de inserção.
427 A linha de código ' || NEW.codigo_func || 'foi inserido');
428 RETURN NEW;
429 -- Aqui temos um bloco IF que confirmará o tipo de operação UPDATE.
430 ELSIF (TG_OP = 'UPDATE') THEN
431 INSERT INTO funcionarios_auditoria (log_id, data_criacao, operacao_realizada)
432 VALUES (NEW.codigo_func, current_timestamp, 'Operação de UPDATE.
433 A linha de código ' || NEW.codigo_func || ' teve os valores atualizados '
434 || OLD || ' com ' || NEW.* || '.');
435 RETURN NEW;
436 -- Aqui temos um bloco IF que confirmará o tipo de operação DELETE
437 ELSIF (TG_OP = 'DELETE') THEN
438 INSERT INTO funcionarios_auditoria (log_id, data_criacao, operacao_realizada)
439 VALUES (OLD.codigo_func, current_timestamp, 'Operação DELETE. A linha de código '
440 || OLD.codigo_func || ' foi excluída ');
441 RETURN OLD;
442 END IF;
443 RETURN NULL;
444 END;
445 $BODY$
```

LISTAGEM 8

AGORA CRIAREMOS A TRIGGER QUE SERÁ VINCULADA A TABELA DE FUNCIONÁRIOS,
COMO PODEMOS VER NA LISTAGEM 8

```
447 /*Criação da tabela de funcionários*/  
448 CREATE TRIGGER trigger_log_todas_as_operacoes  
449 AFTER INSERT OR UPDATE OR DELETE ON funcionarios  
450 FOR EACH ROW  
451 EXECUTE PROCEDURE funcionario_log_function();
```

LISTAGEM 9

PARA TERMOS OS RESULTADOS ARMAZENADOS NA NOSSA TABELA UTILIZANDO AS OPERAÇÕES DML, UTILIZAREMOS AS INSTRUÇÕES DE INSERT, UPDATE E DELETE, DE ACORDO COM A LISTAGEM 9

```
453 /*Utilizando as operações DML*/
454 INSERT INTO funcionarios (codigo, nome, email, telefone, profissao, endereco, salario)
455 VALUES (6, 'João da Silva 2', 'joaozinho@gmail.com', '(81)997445854', 'Analista de testes',
456 'rua desespero', 4500.00);
457 UPDATE funcionarios set nome = 'Caroline Dionisio' WHERE codigo_func = '3';
458 DELETE FROM funcionarios WHERE codigo_func= 5;
```

LISTAGEM 10

VEJA QUE A INSERÇÃO DOS REGISTROS NA TABELA DE FUNCIONARIOS_AUDITORIA COM AS INFORMAÇÕES DO CÓDIGO DOS REGISTROS, DATA DE ATUALIZAÇÃO E O TIPO DE OPERAÇÃO, FOI REALIZADA. PARA QUE POSSAMOS VER OS RESULTADOS BASTA UTILIZARMOS A INSTRUÇÃO SELECT, COMO APRESENTADA A SEGUIR:

```
e  
o  
c  
n  
it  
| 460 /*visualizando resultados*/  
| 461  
| 462 SELECT log_id, data_criacao FROM funcionarios_auditoria;  
| 463 |
```

TRABALHANDO COM AS VIEWS E AS TRIGGERS

NESTE MOMENTO VEREMOS UM POUCO SOBRE A UTILIZAÇÃO DAS VIEWS EM CONJUNTO COM AS TRIGGERS. PARA ISSO CRIAREMOS UM NOVO EXEMPLO COM A TABELA CHAMADA FUNCIONARIO_VIEW E UMA VIEW CHAMADA VIEW_FUNCIONARIOS. A TABELA FUNCIONARIO_VIEW ESTÁ NA LISTAGEM 11:

```
465 /*Criação da tabela funcionario_view*/
466 CREATE TABLE funcionario_view
467 (
468 codigo_func INT NOT NULL,
469 nome VARCHAR(100),
470 email VARCHAR(100)
471 );|
```

TRABALHANDO COM AS VIEWS E AS TRIGGERS

COM A TABELA CRIADA, VAMOS ADICIONAR ALGUNS DADOS DE TESTES, COMO PODEMOS VER NA LISTAGEM 12.

```
473 /* Inserção de registros de teste*/
474 INSERT INTO funcionario_view VALUES (1, 'Edson', 'edson.dionisio@gmail.com ');
475 INSERT INTO funcionario_view VALUES (2, 'Marília', 'mkessia@teste.com');
476 INSERT INTO funcionario_view VALUES (3, 'Caroline', 'carol@teste.com');
477 INSERT INTO funcionario_view VALUES (4, 'Gustavo', 'gustavo@teste.com');
478 INSERT INTO funcionario_view VALUES (5, 'Maria', 'maria@teste.com');
```

TRABALHANDO COM AS VIEWS E AS TRIGGERS

APÓS A INSERÇÃO DOS DADOS DE TESTES CRIAMOS A VIEW COM BASE NOS DADOS INSERIDOS DA SEGUINTE FORMA:

```
173  
480 /*criando view*/  
481  
482 CREATE VIEW view_funcionarios AS SELECT * FROM funcionario_view;  
483
```

TRABALHANDO COM AS VIEWS E AS TRIGGERS

ESSA FOI A FORMA MAIS SIMPLES DE APRESENTARMOS UMA VIEW, MAS PODEMOS FAZER DE FORMA DIFERENTE. VAMOS CRIAR UMA VIEW QUE SERÁ ATUALIZADA SEMPRE QUE UMA TRIGGER FOR DISPARADA, COMO PODEMOS VER NA LISTAGEM 13.

```
485 /*Criando uma trigger de atualização de View*/
486 CREATE FUNCTION atualiza_view_trigger()
487 RETURNS trigger AS $
488 BEGIN
489 IF (TG_OP = 'INSERT') THEN
490 INSERT INTO funcionario_view VALUES (NEW.codigo_func, NEW.nome, NEW.email);
491 RETURN NEW;
492 END IF;
493 RETURN NULL;
494 END;
```

TRABALHANDO COM AS VIEWS E AS TRIGGERS

EM SEGUIDA, CRIAMOS UMA TRIGGER E VINCULAMOS A VIEW VIEW_FUNCIONARIOS COM O CÓDIGO DA LISTAGEM 14

```
e 496 /*Criação da trigger dispara_trigger_func*/
e 497 CREATE TRIGGER dispara_trigger_func
e 498 INSTEAD OF INSERT ON view_funcionarios
e 499 FOR EACH ROW
e 500 EXECUTE PROCEDURE atualiza_view_trigger();
e 501 |
```

TRABALHANDO COM AS VIEWS E AS TRIGGERS

PARA QUE POSSAMOS VER OS RESULTADOS CONTIDOS NA VIEW VIEW_FUNCIONARIOS UTILIZAMOS A INSTRUÇÃO SELECT DA SEGUINTE FORMA:

```
503 /*visualizar resultado*/  
504  
505 SELECT * FROM view_funcionarios;  
506
```

TRABALHANDO COM AS VIEWS E AS TRIGGERS

NA VIEW_FUNCIONARIOS VAMOS INSERIR UM NOVO REGISTRO, COMO SEGUE O EXEMPLO:

```
508 /*inserindo novo registro*/  
509  
510 INSERT INTO view_funcionarios VALUES (6, 'Joao inserido através da view', 'view@teste.com');  
511
```

TRABALHANDO COM AS VIEWS E AS TRIGGERS

MAS O REGISTRO FOI INSERIDO NA TABELA FUNCIONÁRIO_VIEW, CERTO? PARA VERMOS O RESULTADO, UTILIZAREMOS O COMANDO SELECT> E VEREMOS QUE NOSSO REGISTRO FOI INSERIDO CORRETAMENTE, COMO SEGUE:

```
511  
512 /*visualizar resultado*/  
513  
514 SELECT * FROM funcionario_view;
```

TRABALHANDO COM AS VIEWS E AS TRIGGERS

PARA QUE POSSAMOS VER TODAS AS TRIGGERS QUE TEMOS ADICIONADAS NO NOSSO BANCO DE DADOS PODEMOS UTILIZAR O SEGUINTE COMANDO:

```
516 /*visualizar todas as triggers*/  
517  
518 SELECT * FROM pg_trigger;  
519
```

TRABALHANDO COM AS VIEWS E AS TRIGGERS

CASO A INTENÇÃO SEJA VER TODAS AS TRIGGERS PARA UMA TABELA ESPECÍFICA, ENTÃO A QUERY DEVERÁ SER DE ACORDO COM A APRESENTADA A SEGUIR:

```
520 /*visualizar todas as triggers da tabela funcionários*/  
521  
522 SELECT tgname FROM pg_trigger, pg_class WHERE tgrelid = pg_class.oid  
523 AND relname = 'funcionarios';|
```

TRABALHANDO COM AS VIEWS E AS TRIGGERS

PARA FINALIZARMOS, CASO TENHAMOS INTERESSE EM EXCLUIR AS TRIGGERS, PODEMOS USAR O SEGUINTE COMANDO:

```
525 /*excluir todas as triggers da tabela funcionários*/  
526  
527 DROP TRIGGER trigger_log_todas_as_operacoes ON funcionários;  
528 |
```