FIAP ON

DATABASE PROGRAMMING

# GATILHOS *MÁGICOS*

MILTON GOYA

10

## LISTA DE QUADROS

## LISTA DE CÓDIGOS-FONTE

**SUMÁRIO**

# 1 GATILHOS MÁGICOS

Nos últimos capítulos, aproveitamos para otimizar todas as ações e blocos de código na programação PL/SQL. As procedures, por exemplo, são acionadas pelo usuário ou pela aplicação. As triggers (assunto deste capítulo) também possuem os procedimentos de otimização armazenados, são acionadas pelo próprio Oracle quando ocorre um determinado evento, como um gatilho mágico!

Para a Oracle (2016), gatilhos ou TRIGGERS são procedimentos armazenados PL/SQL associados a tabelas, visões, esquemas ou bancos de dados. São executados ou disparados automaticamente quando ocorrem determinados eventos. Os gatilhos são desenvolvidos para serem executados em resposta a qualquer um dos seguintes eventos:

Um comando DML ou DATABASE MANIPULATION LANGUAGE, normalmente é acionado antes ou depois de um comando INSERT, UPDATE ou DELETE.

Um comando DDL ou DATABASE DEFINITION LANGUAGE geralmente é acionado antes ou depois de um comando CREATE, ALTER ou DROP.

Uma operação de banco de dados (LOGON, LOGOFF, SERVERERROR, STARTUP ou SHUTDOWN).

Para Dillon *et al.* (2013), os gatilhos ou TRIGGERS podem ser usados com os seguintes propósitos:

- Geração automática dos valores para colunas derivadas.
- Manutenção da integridade referencial.
- Registrar o histórico de acesso a uma tabela.
- Registrar o histórico das alterações em uma tabela.
- Auditoria.
- Replicação síncrona das tabelas.
- Impor restrições de segurança.
- Prevenir transações inválidas.

- Implantação das regras dos negócios.

Estudaremos mais profundamente gatilhos afetados pelos comandos DML, visto que são os mais utilizados pelos desenvolvedores. Os demais tipos de gatilhos costumam ser de responsabilidade dos administradores de banco de dados.

O quadro abaixo mostra as diferenças entre os gatilhos e os procedimentos armazenados e funções.

| Gatilhos | Procedimentos ou Funções |
|---|---|
| Ativados Implicitamente | Ativados Explicitamente |
| Proibido usar COMMIT, ROLLBACK e SAVEPOINT | COMMIT, ROLLBACK e SAVEPOINT são permitidos |
| Quem ativa não precisa ter privilégios de execução | Quem ativa precisa ter privilégios de execução |

Quadro 1 – Comparação entre gatilhos e procedimentos armazenados e funções
Fonte: Puga, França e Goya (2015)

## 1.1 Sintaxe

Para a Oracle (2016), um gatilho pode ser acionado quando ocorre uma operação DML para inserção, alteração ou exclusão de dados.

```
CREATE [OR REPLACE] TRIGGER [esquema.]nome_trigger
{BEFORE ou AFTER}
[evento] ON [esquema.]tabela_nome
[referencing Old as valor_anterior ou NEW as valor_novo)
{nível de linha ou nível de instrução} [WHEN (condição)]] DECLARE
BEGIN
   corpo_trigger
END;
```

Código-fonte 1 – Sintaxe da criação de TRIGGER
Fonte: Oracle (2016)

**Onde**, **Create or Replace Trigger** é a instrução para a criação ou a substituição do gatilho ou TRIGGER.

**Esquema** é o nome do esquema ao qual pertence o objeto.

**Nome_trigger** é o nome ou identificador do gatilho ou TRIGGER.

**Before** indica que a execução do gatilho ou TRIGGER será antes da realização do evento.

**After** indica que a execução do gatilho ou TRIGGER será depois da realização do evento.

**Evento** define qual o tipo de instrução que provocará o disparo do gatilho ou TRIGGER. As instruções possíveis são INSERT, UPDATE ou DELETE.

**On tabela_nome** indica a tabela que está associada à instrução.

**Referencing Old as valor_anterior ou NEW as valor_novo** especifica os identificadores das variáveis que armazenarão os valores antigos e novos. OLD e NEW são pseudocolunas que auxiliam na manipulação das linhas afetadas.

**Nível de linha** é acionado uma vez para cada linha afetada pela instrução e identificado por FOR EACH ROW.

**Nível de instrução** é acionado antes ou depois da instrução.

**When condição** pode ser utilizada em gatilho de linha. Indica que o gatilho será acionado somente quando a condição for verdadeira.

### 1.1.1 TRIGGER DML

Para a Oracle (2016), um gatilho ou TRIGGER tem três características: ação, escopo e tempo.

A AÇÃO representa o evento ou comando que aciona a execução do gatilho ou TRIGGER.

O ESCOPO define se o gatilho ou TRIGGER será executado para cada linha processada ou para a instrução.

O TEMPO indica se a execução ocorrerá antes ou depois da ação.

Ibidem, um gatilho ou TRIGGER DML é acionado por uma instrução DML. As instruções DML que podem acionar um gatilho são:

INSERT

UPDATE

UPDATE FOR nome_da_coluna [, nome_da_coluna ...]

DELETE

Ibidem, as palavras-chaves obrigatórias AFTER ou BEFORE e a cláusula opcional FOR EACH ROW definem o momento em que o gatilho será acionado.

Vejamos um exemplo simples:

```
SET SERVEROUTPUT ON

CREATE OR REPLACE TRIGGER mudancas_salariais
BEFORE UPDATE ON emp
FOR EACH ROW
DECLARE
   saldo number;
BEGIN
   saldo := :NEW.sal  - :OLD.sal;
   DBMS_OUTPUT.PUT_LINE('Salario Anterior: ' || :OLD.sal);
   DBMS_OUTPUT.PUT_LINE('Salario Novo: ' || :NEW.sal);
   DBMS_OUTPUT.PUT_LINE('Diferenca Salarial: ' || saldo);
END;
/
```

Código-fonte 2 – Exemplo de criação de gatilho
Fonte: Adaptado de Oracle (2016)

No exemplo, estamos criando um gatilho denominado MUDANCAS_SALARIAIS, que é acionado antes da atualização dos dados da tabela EMP (BEFORE UPDATE ON EMP). Esse exemplo foi criado com a opção FOR EACH ROW, indicando que será executado uma vez para cada linha que a operação modifica. Na seção executável, estamos declarando uma variável denominada SALDO, que é usada para receber a diferença entre o salário atual do funcionário e seu novo salário.

Perceba que estamos referenciando o salário novo com o pseudorregistro :NEW e o salário anterior com o pseudorregistro :OLD. Isso quer dizer que, ao atualizar uma linha na tabela EMP, estamos subtraindo o novo salário, :NEW.SAL, do salário anterior, :OLD.SAL, e atribuindo o resultado à variável SALDO. Nosso gatilho, então, exibe o salário anterior, :OLD.SAL, o novo salário, :NEW.SAL, e o saldo calculado.

Vamos acionar o nosso gatilho para vê-lo em ação:

```
SET SERVEROUTPUT ON

UPDATE EMP
    SET sal = sal * 2
  WHERE empno = 7900
/

Salario Anterior: 950
Salario Novo: 1900
Diferenca Salarial: 950
```

Código-fonte 3 – Exemplo de acionamento de gatilho
Fonte: Adaptado de Oracle (2016)

O nosso gatilho foi acionado ao atualizarmos o salário do funcionário de código 7900. No caso, estamos multiplicando seu salário por dois. Como resultado, o gatilho exibiu o salário anterior, 950, o salário novo, 1900, e a diferença salarial entre os dois valores, 950. Perceba que o gatilho foi acionado em operações de atualização, pois foi criado com BEFORE UPDATE ON EMP. Vamos alterar nosso gatilho para ser acionado em operações de inclusão e exclusão de dados.

```
SET SERVEROUTPUT ON

CREATE OR REPLACE TRIGGER mudancas_salariais
BEFORE INSERT OR UPDATE OR DELETE ON emp
FOR EACH ROW
DECLARE
   saldo number;
BEGIN
   saldo := :NEW.sal  - :OLD.sal;
   DBMS_OUTPUT.PUT_LINE('Salario Anterior: ' || :OLD.sal);
   DBMS_OUTPUT.PUT_LINE('Salario Novo: ' || :NEW.sal);
   DBMS_OUTPUT.PUT_LINE('Diferenca Salarial: ' || saldo);
END;
/
```

Código-fonte 4 – Exemplo de criação de gatilho DML
Fonte: Adaptado de Oracle (2016)

Fizemos apenas uma alteração no nosso gatilho. Agora é acionado a cada inclusão, alteração e exclusão, graças à linha BEFORE INSERT OU UPDATE OR DELETE ON EMP. Essa alteração tem efeitos importantes nos pseudorregistros:registros: NEW e:OLD:

- Em operações de INSERT, apenas: NEW pode conter dados. Isso ocorre porque a linha está sendo incluída nessa operação e não existem dados anteriores a ela.

- Em operações de UPDATE, tanto :NEW quanto :OLD podem ser referenciados. :NEW referencia o novo valor usado na declaração DML que acionou o gatilho, enquanto :OLD representa o valor preexistente na coluna antes da alteração.
- Em operações de DELETE, apenas :OLD pode conter dados. Isso ocorre porque não existem novos dados a serem incluídos, existem os dados que estão sendo excluídos.

Vejamos como nosso gatilho se comporta nessas três situações:

```
SET SERVEROUTPUT ON

--
-- Acionando o gatilho com uma instrução INSERT
--

INSERT INTO emp (empno, sal)
VALUES (1000, 2780);

Salario Anterior:
Salario Novo: 2780
Diferenca Salarial:
```

Código-fonte 5 – Exemplo de acionamento de gatilho com INSERT
Fonte: Adaptado de Oracle (2016)

No exemplo, estamos testando o acionamento do gatilho com o comando INSERT. Note que, ao acionarmos o gatilho com o comando INSERT, não existe o valor do salário anterior, ou seja, não há valor para: OLD e, sendo assim, o salário anterior é exibido sem nenhum valor. O salário novo mostra o valor inserido e a diferença salarial também é exibida sem nenhum valor.

```
SET SERVEROUTPUT ON

--
-- Acionando o gatilho com uma instrução UPDATE
--

UPDATE EMP
   SET sal = sal * 2
 WHERE empno = 1000;

Salario Anterior: 2780
Salario Novo: 5560
Diferenca Salarial: 2780
```

Código-fonte 6 – Exemplo de acionamento de gatilho com UPDATE
Fonte: Adaptado de Oracle (2016)

No exemplo, estamos testando o acionamento do gatilho com o comando UPDATE. Ao acionarmos o gatilho com o comando UPDATE, mostra o salário anterior, o novo salário e a diferença salarial. O resultado foi diferente do acionamento anterior, porque, desta vez, havia valores anteriores a serem exibidos.

```
SET SERVEROUTPUT ON

--
-- Acionando o gatilho com uma instrução DELETE
--

DELETE emp
 WHERE empno = 1000;

Salario Anterior: 5560
Salario Novo:
Diferenca Salarial:
```

Código-fonte 7 – Exemplo de acionamento de gatilho com DELETE
Fonte: Adaptado de Oracle (2016)

Ao acionarmos o gatilho com o comando DELETE, exibe o salário anterior, mas não há salário novo a ser exibido nem diferença salarial. Novamente, o comportamento desse acionamento foi diferente dos anteriores. Isso porque, dessa vez, não tínhamos novos dados a serem exibidos.

Perceba que nem sempre queremos que nosso gatilho atue da mesma forma para os comandos DML. Para tratar esses casos, podemos testar quais operações estão sendo executadas por meio dos predicados INSERTING, UPDATING e UPDATING.

Segundo Puga, França e Goya (2015), esses predicados retornam o valor TRUE nas seguintes condições:

- INSERTING – Resulta TRUE, se a instrução de acionamento do gatilho for INSERT.
- UPDATING – Resulta TRUE, se a instrução de acionamento do gatilho for UPDATE.
- DELETING – Resulta TRUE, se a instrução de acionamento do gatilho for DELETE.

Vamos aperfeiçoar nosso gatilho com o uso desses predicados:

```
SET SERVEROUTPUT ON

CREATE OR REPLACE TRIGGER mudancas_salariais
BEFORE INSERT OR UPDATE OR DELETE ON emp
FOR EACH ROW
DECLARE
   saldo number;
BEGIN

   CASE
     WHEN INSERTING THEN
          DBMS_OUTPUT.PUT_LINE('    Novo : ' || :NEW.sal);
     WHEN UPDATING THEN
          saldo := :NEW.sal  - :OLD.sal;
          DBMS_OUTPUT.PUT_LINE('Anterior : ' || :OLD.sal);
          DBMS_OUTPUT.PUT_LINE('    Novo : ' || :NEW.sal);
          DBMS_OUTPUT.PUT_LINE('Diferenca: ' || saldo);
     WHEN DELETING THEN
          DBMS_OUTPUT.PUT_LINE('Anterior : ' || :OLD.sal);
  END CASE;

END;
/
```

Código-fonte 8 – Exemplo de criação de gatilho DML
com os predicados INSERTING, UPDATING e DELETING
Fonte: Adaptado de Oracle (2016)

No exemplo, agora estamos testando se estamos incluindo, atualizando ou eliminando um registro na tabela EMP. Caso a operação seja um comando de INSERT, o predicado INSERTING estará com o valor TRUE e o gatilho exibirá apenas o valor que está sendo incluído.

Se a operação for um comando de UPDATE, o predicado UPDATING estará com o valor TRUE e o gatilho exibirá o valor do salário, o novo salário e a diferença entre o salário atual e o novo. Caso a operação seja um comando de DELETE, o predicado DELETING estará com o valor TRUE e o gatilho exibirá apenas o valor do salário que está sendo removido da tabela.

Vejamos como o nosso gatilho passa a se comportar:

```
SET SERVEROUTPUT ON

--
-- Acionando o gatilho com um instrução INSERT
--
```

```
INSERT INTO emp (empno, sal)
VALUES (1000, 2780);

Novo: 2780
```

Código-fonte 9 – Exemplo de acionamento de gatilho com teste de INSERTING
Fonte: Adaptado de Oracle (2016)

Perceba que, dessa vez, testamos o acionamento do gatilho com a instrução INSERT, somente a mensagem do novo salário foi exibida. Isso acontece porque, em nosso gatilho, testamos se ocorreu uma operação de INSERT por meio do uso do predicado INSERTING.

```
SET SERVEROUTPUT ON

--
-- Acionando o gatilho com uma instrução UPDATE
--

UPDATE EMP
   SET sal = sal * 2
 WHERE empno = 1000;

Anterior: 2780
Novo: 5560
Diferenca: 2780
```

Código-fonte 10 – Exemplo de acionamento de gatilho
com teste de INSERTING, UPDATING e DELETING
Fonte: Adaptado de Oracle (2016)

A mensagem mostrada foi diferente do teste anterior, porque estamos testando a operação de UPDATE com o predicado UPDATING. Perceba que, em uma operação de UPDATE, temos tanto o valor para NEW quanto para OLD e, assim, podemos mostrar uma mensagem mais completa.

```
SET SERVEROUTPUT ON

--
-- Acionando o gatilho com uma instrução DELETE
--

DELETE emp
 WHERE empno = 1000;

Anterior: 5560
```

Código-fonte 11 – Exemplo de acionamento de gatilho
com teste de INSERTING, UPDATING e DELETING
Fonte: Adaptado de Oracle (2016)

Note que o comportamento do gatilho mudou conforme a operação que estamos realizando e as mensagens são específicas para cada tipo de operação realizada.

### 1.1.2 Transações autônomas com TRIGGER

Para Dillon et al (2013), os gatilhos não devem afetar a transação atual, e por esse motivo, não devem conter instruções COMMIT ou ROLLBACK. Caso seja necessário executar uma operação de COMMIT ou ROLLBACK no gatilho, o indicado é que seja colocada em um procedimento autônomo. O procedimento autônomo pode ser executado a partir do gatilho.

As transações autônomas permitem que deixe o contexto da transação de chamada, execute uma transação independente e retorne à transação de chamada sem afetar seu estado. A transação autônoma não possui nenhum *link* para a transação de chamada, portanto, apenas as informações podem ser compartilhadas por ambas as transações.

Vejamos um exemplo simples:

```
SET SERVEROUTPUT ON

CREATE TABLE auditoria
(codigo   NUMBER(5),
 hora     DATE,
 operacao VARCHAR2(6),
 antigo   NUMBER (7,2),
 novo     NUMBER (7,2));

CREATE OR REPLACE PROCEDURE registra
 (p_codigo   IN  VARCHAR2,
  P_operacao IN  VARCHAR2,
  P_antigo   IN  NUMBER,
  P_novo     IN  NUMBER) AS
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  INSERT INTO auditoria (codigo, hora, operacao, antigo, novo)
  VALUES (p_codigo, SYSDATE, p_operacao, p_antigo, p_novo);
  COMMIT;
END;
/

CREATE OR REPLACE TRIGGER mudancas_salariais
BEFORE INSERT OR UPDATE OR DELETE ON emp
FOR EACH ROW
BEGIN
```

```
   CASE
     WHEN INSERTING THEN
          registra(:NEW.empno, 'INSERT', :OLD.sal, :NEW.sal);
     WHEN UPDATING THEN
          registra(:OLD.empno, 'UPDATE', :OLD.sal, :NEW.sal);
     WHEN DELETING THEN
          registra(:OLD.empno, 'DELETE', :OLD.sal, :NEW.sal);
  END CASE;

END;
/
```

Código-fonte 12 – Exemplo de criação de gatilho DML
executando um PROCEDIMENTO AUTÔNOMO
Fonte: Adaptado de Oracle (2016)

No exemplo, criamos uma tabela denominada AUDITORIA. Vamos inserir o código do empregado, a data de execução da operação DML, o salário antigo e o novo. Em seguida, criaremos um procedimento armazenado denominado REGISTRA, que é necessário porque queremos executar o comando COMMIT por meio de um gatilho. O procedimento armazenado é autônomo, que insere os dados recebidos e grava definitivamente os dados na tabela. O nosso gatilho foi alterado para executar o procedimento armazenado, informando diferentes dados, conforme a operação DML executada.

Vejamos como nosso gatilho passa a se comportar:

```
SET SERVEROUTPUT ON

ALTER SESSION SET nls_date_format='DD/MM/YY HH24:MI:SS';

--
-- Acionando o gatilho com uma instrução INSERT
--

INSERT INTO emp (empno, sal)
VALUES (1000, 2780);

SELECT * FROM auditoria;

CODIGO HORA              OPERAC ANTIGO  NOVO
------ ----------------- ------ ------ -----
  1000 10/12/17 01:01:53 INSERT         2780
```

Código-fonte 13 – Exemplo de acionamento de gatilho com uma operação de INSERT
Fonte: Adaptado de Oracle (2016)

Em nosso exemplo, estamos alterando o formato de exibição da data para a seção corrente. O formato-padrão exibe a data em formato dia, mês e ano. Após a alteração, o novo formato exibirá a data no formato dia, mês, ano, hora com 24 horas, minuto e segundo. O formato da data voltará ao formato original na próxima vez que entrarmos com esse usuário.

Ao executar o comando INSERT na tabela EMP, o gatilho é acionado e executa o procedimento armazenado que inclui os dados na tabela AUDITORIA. No exemplo, exibimos o conteúdo da tabela. Perceba que ao consultarmos a tabela AUDITORIA, é exibido o código inserido, data e hora da operação, qual foi a operação, o valor anterior do campo salário e o valor atual do mesmo campo. No caso, como foi uma operação de INSERT, não há valor anterior.

```
--
-- Acionando o gatilho com uma instrução UPDATE
--

UPDATE EMP
   SET sal = sal * 2
 WHERE empno = 1000;

SELECT * FROM auditoria;

CODIGO HORA              OPERAC ANTIGO       NOVO
------ ----------------- ------ ------ ----------
  1000 10/12/17 01:01:53 INSERT              2780
  1000 10/12/17 01:04:08 UPDATE   2780       5560
```

Código-fonte 14 – Exemplo de acionamento de gatilho com uma operação de UPDATE
Fonte: Adaptado de Oracle (2016)

Ao executar o comando UPDATE na tabela EMP, o gatilho é acionado e executa o procedimento armazenado, incluindo os dados na tabela AUDITORIA. Note que a tabela foi consultada e mantém informações sobre a operação anterior, INSERT, e a operação atual, UPDATE. Dessa vez, temos o valor anterior e o novo valor para exibir.

```
--
-- Acionando o gatilho com uma instrução DELETE
--

DELETE emp
 WHERE empno = 1000;

SELECT * FROM auditoria;

CODIGO HORA              OPERAC ANTIGO       NOVO
------ ----------------- ------ ------ ----------
```

```
  1000 10/12/17 01:01:53 INSERT              2780
  1000 10/12/17 01:04:08 UPDATE   2780       5560
  1000 10/12/17 01:06:53 DELETE   5560
```

Código-fonte 15 – Exemplo de acionamento de gatilho
com execução de um procedimento armazenado
Fonte: Adaptado de Oracle (2016)

Dessa vez o comando UPDATE acionou o gatilho e registrou os dados do registro que está sendo eliminado na tabela AUDITORIA. A consulta à tabela mostra que não temos dados novos a serem registrados. Isso acontece porque a operação de DELETE não possui dados novos.

### 1.1.3 Testando condições com a CLÁUSULA WHEN

Para Feuerstein e Pribyl (2014), a cláusula WHEN pode ser usada em um gatilho DML para restringir seu acionamento. O gatilho será acionado para as linhas que satisfaçam a condição definida. Para deixar o acionamento do gatilho mais preciso, podemos usar a cláusula OF, indicando qual coluna deve ser afetada para o gatilho ser acionado.

Vejamos um exemplo simples:

```
CREATE OR REPLACE TRIGGER mudancas_salariais
BEFORE INSERT OR DELETE OR UPDATE OF sal ON emp
FOR EACH ROW
WHEN (NEW.SAL > 1000)
BEGIN

   CASE
     WHEN INSERTING THEN
          registra(:NEW.empno, 'INSERT', :OLD.sal, :NEW.sal);
     WHEN UPDATING THEN
          registra(:OLD.empno, 'UPDATE', :OLD.sal, :NEW.sal);
     WHEN DELETING THEN
          registra(:OLD.empno, 'DELETE', :OLD.sal, :NEW.sal);
  END CASE;

END;
/
```

Código-fonte 16 – Exemplo de criação de gatilho DML
com o condicional WHEN e cláusula OF
Fonte: Adaptado de Oracle (2016)

Alteramos o exemplo anterior acrescentando UPDATE OF SAL ON EMP e WHEN (NEW.SAL > 1000). Com essas alterações, o gatilho será acionado nos casos em que o novo salário for superior a 1000 e a coluna afetada for a coluna SAL.

Vamos testar nossas alterações:

```
TRUNCATE TABLE auditoria;

--
-- Tentativa de acionar o gatilho com uma instrução INSERT e
-- salário inferior a 1000
--

INSERT INTO emp (empno, sal)
VALUES (1000, 780);

SELECT * FROM auditoria;

no rows selected

--
-- Acionando o gatilho com uma instrução INSERT e
-- salário superior a 1000
--

INSERT INTO emp (empno, sal)
VALUES (1001, 2780);

SELECT * FROM auditoria;

CODIGO HORA              OPERAC ANTIGO NOVO
------ ----------------- ------ ------ ----
  1001 10/12/17 15:29:56 INSERT        2780
```

Código-fonte 17 – Exemplo de acionamento de gatilho com INSERT e as cláusulas WHEN e OF
Fonte: Adaptado de Oracle (2016)

Iniciamos o nosso teste executando o comando TRUNCATE para limpar os dados da tabela AUDITORIA. O comando TRUNCATE apaga todos os registros de uma tabela e não permite que sejam recuperados usando o comando ROLLBACK.

No exemplo, fizemos duas operações de INSERT. A primeira operação incluiu o funcionário de número 1000 na tabela EMP, mas não acionou o gatilho, porque o valor do salário era inferior a 1000. A segunda operação incluiu o funcionário de número 1001, acionou o gatilho e atualizou a tabela AUDITORIA, porque o valor do salário era superior a 1000.

```
--
-- Tentativa de acionar o gatilho com uma instrução UPDATE e
-- salário inferior a 1000
--

UPDATE EMP
   SET sal = 780
 WHERE empno = 1000;

SELECT * FROM auditoria;

CODIGO HORA              OPERAC ANTIGO       NOVO
------ ----------------- ------ ------ ----------
  1000 10/12/17 01:01:53 INSERT              2780

--
-- Acionando o gatilho com uma instrução UPDATE e
-- salário superior a 1000
--

UPDATE EMP
   SET sal = 3000
 WHERE empno = 1000;

SELECT * FROM auditoria;

CODIGO HORA              OPERAC ANTIGO       NOVO
------ ----------------- ------ ------ ----------
  1000 10/12/17 01:01:53 INSERT              2780
  1000 10/12/17 00:41:41 UPDATE    780       3000
```

Código-fonte 18 – Exemplo de acionamento de gatilho
com UPDATE e as cláusulas WHEN e OF
Fonte: Adaptado de Oracle (2016)

No exemplo, fizemos duas operações de UPDATE. A primeira alterou o salário do funcionário de número 1000 na tabela EMP, mas não acionou o gatilho, porque o valor do novo salário era inferior a 1000. A segunda alterou o salário do funcionário de número 1000, acionou o gatilho e atualizou a tabela AUDITORIA, porque o valor do novo salário era superior a 1000.

```
--
-- Tentativa de acionar o gatilho com uma instrução DELETE e
-- salário superior a 1000
--

DELETE emp
 WHERE empno = 1000;

SELECT * FROM auditoria;
```

```
CODIGO HORA              OPERAC ANTIGO       NOVO
------ ----------------- ------ ------ ----------
  1000 10/12/17 01:01:53 INSERT              2780
  1000 10/12/17 00:41:41 UPDATE    780       3000
```

Código-fonte 19 – Exemplo de acionamento de gatilho
com DELETE e as cláusulas WHEN e OF
Fonte: Adaptado de Oracle (2016)

No exemplo, fizemos uma operação de DELETE. O comando eliminou o funcionário de número 1000 da tabela EMP, mas não acionou o gatilho apesar de seu salário ser superior a 1000. O gatilho não foi acionado, porque usamos o termo WHEN (NEW.SAL > 1000) em sua construção, e em uma operação de DELETE, não existe um valor novo a ser testado. Nesse caso, o termo deveria ser WHEN (NEW.SAL > 1000 OR OLD.SAL > 1000), nosso código alterado ficaria assim:

```
CREATE OR REPLACE TRIGGER mudancas_salariais
BEFORE INSERT OR DELETE OR UPDATE OF sal ON emp
FOR EACH ROW
WHEN (NEW.SAL > 1000 OR
      OLD.SAL > 1000)
BEGIN

   CASE
     WHEN INSERTING THEN
          registra(:NEW.empno, 'INSERT', :OLD.sal, :NEW.sal);
     WHEN UPDATING THEN
          registra(:OLD.empno, 'UPDATE', :OLD.sal, :NEW.sal);
     WHEN DELETING THEN
          registra(:OLD.empno, 'DELETE', :OLD.sal, :NEW.sal);
  END CASE;

END;
/
```

Código-fonte 20 – Atualização do exemplo de criação
de gatilho DML com o condicional WHEN e cláusula OF
Fonte: Adaptado de Oracle (2016)

Com a alteração, o nosso gatilho passa ser acionado quando NEW.SAL ou OLD.SAL forem maiores que 1000. Relembrando que NEW.SAL possui valores nas operações de INSERT e UPDATE, enquanto OLD.SAL tem valores nas operações de UPDATE e DELETE.

### 1.1.4 Cláusula Referencing

Segundo Puga, França e Goya (2015), o padrão SQL ANSI permite definir outros nomes ou ALIAS para os pseudorregistros "OLD" e "NEW", que são utilizados na definição da ação do gatilho. Esse recurso pode ser útil quando trabalhamos com códigos extensos.

Vamos alterar nosso exemplo anterior para demonstrar o uso desse recurso:

```
CREATE OR REPLACE TRIGGER mudancas_salariais
BEFORE INSERT OR DELETE OR UPDATE OF sal ON emp
REFERENCING NEW AS novo_emp
            OLD AS antigo_emp
FOR EACH ROW
WHEN (novo_emp.SAL > 1000 OR
      antigo_emp.SAL > 1000)
BEGIN

   CASE
     WHEN INSERTING THEN
          registra(:novo_emp.empno, 'INSERT', :antigo_emp.sal,
:novo_emp.sal);
     WHEN UPDATING THEN
          registra(:antigo_emp.empno, 'UPDATE', :antigo_emp.sal,
:novo_emp.sal);
     WHEN DELETING THEN
          registra(:antigo_emp.empno, 'DELETE', :antigo_emp.sal,
:novo_emp.sal);
  END CASE;

END;
/
```

Código-fonte 21 – Atualização do exemplo de criação de gatilho DML com REFERENCING
Fonte: Adaptado de Oracle (2016)

No exemplo, o nosso gatilho agora referencia o pseudorregistro NEW com o nome de NOVO_EMP e o pseudorregistro OLD com o nome de ANTIGO_EMP. A funcionalidade do gatilho é a mesma, registrar as alterações do salário dos funcionários ao longo do tempo.

### 1.1.5 Momento de ativação

Para a Oracle (2016), basicamente, existem quatro momentos ou tempo de ativação de um gatilho ou TRIGGER DML:

- BEFORE STATEMENT – O gatilho é definido usando a palavra-chave BEFORE, mas a cláusula FOR EACH ROW é omitida.
- BEFORE EACH ROW – O gatilho é estipulado usando a palavra-chave BEFORE e a cláusula FOR EACH ROW.
- AFTER EACH ROW – O gatilho é estabelecido usando a palavra-chave AFTER e a cláusula FOR EACH ROW.
- AFTER STATEMENT – O gatilho é determinado usando a palavra-chave APÓS, mas a cláusula FOR EACH ROW é omitida.

É possível criar vários gatilhos usando os mesmos momentos de ativação, mas não garante a ordem de execução, a menos que você use a cláusula FOLLOWS (ORACLE, 2016).

Vejamos alguns exemplos simples para entender melhor o momento de ativação.

```
SET SERVEROUTPUT ON

CREATE OR REPLACE TRIGGER testa_momento
BEFORE INSERT OR UPDATE OR DELETE ON emp
BEGIN

   CASE
     WHEN INSERTING THEN
          DBMS_OUTPUT.PUT_LINE('Insert – BEFORE STATEMENT');
     WHEN UPDATING THEN
          DBMS_OUTPUT.PUT_LINE('Update – BEFORE STATEMENT');
     WHEN DELETING THEN
          DBMS_OUTPUT.PUT_LINE('Delete – BEFORE STATEMENT');
  END CASE;

END;
/
```

Código-fonte 22 – Exemplo de criação de gatilho DML com BEFORE STATEMENT
Fonte: Adaptado de Oracle (2016)

O exemplo mostra a criação de um gatilho com a palavra-chave BEFORE, mas sem a cláusula FOR EACH ROW.

Vejamos o comportamento do gatilho criado dessa forma:

```
--
-- Acionando o gatilho BEFORE STATEMENT com o comando DELETE
--

DELETE emp;

-- Delete - BEFORE STATEMENT

ROLLBACK;
```

Código-fonte 23 – Exemplo de acionamento de gatilho
com DELETE e momento BEFORE STATEMENT
Fonte: Adaptado de Oracle (2016)

No exemplo, estamos apagando todas as 14 linhas da tabela EMP. Nosso gatilho BEFORE STATEMENT é executado uma única vez antes de o comando DELETE ser executado, como mostra a única mensagem de DELETE – BEFORE STATEMENT do nosso teste. Podemos usar esse tipo de gatilho, por exemplo, para verificar se um determinado usuário pode executar uma operação específica em certo momento do dia e impedir a operação, caso o usuário não esteja autorizado a isso.

Não se esqueça de executar o comando ROLLBACK para desfazer as ações de nosso teste e não comprometer o resultado dos demais exemplos.

Vejamos outro exemplo:

```
SET SERVEROUTPUT ON

CREATE OR REPLACE TRIGGER testa_momento
BEFORE INSERT OR UPDATE OR DELETE ON emp
FOR EACH ROW
BEGIN

   CASE
     WHEN INSERTING THEN
          DBMS_OUTPUT.PUT_LINE('Insert – BEFORE EACH ROW');
     WHEN UPDATING THEN
          DBMS_OUTPUT.PUT_LINE('Update – BEFORE EACH ROW');
     WHEN DELETING THEN
          DBMS_OUTPUT.PUT_LINE('Delete – BEFORE EACH ROW');
  END CASE;

END;
/
```

Código-fonte 24 – Exemplo de criação de gatilho DML com BEFORE EACH ROW
Fonte: Adaptado de Oracle (2016)

O exemplo exibe a criação de um gatilho com a palavra-chave BEFORE e a cláusula FOR EACH ROW. Vejamos o comportamento do gatilho criado dessa forma:

```
--
-- Acionando o gatilho BEFORE EACH ROW com o comando DELETE
--

DELETE emp;

-- Delete - BEFORE EACH ROW
-- Delete - BEFORE EACH ROW
-- Delete - BEFORE EACH ROW
-- Delete - BEFORE EACH ROW
-- Delete - BEFORE EACH ROW
-- Delete - BEFORE EACH ROW
-- Delete - BEFORE EACH ROW
-- Delete - BEFORE EACH ROW
-- Delete - BEFORE EACH ROW
-- Delete - BEFORE EACH ROW
-- Delete - BEFORE EACH ROW
-- Delete - BEFORE EACH ROW
-- Delete - BEFORE EACH ROW
-- Delete - BEFORE EACH ROW

ROLLBACK;
```

Código-fonte 25 – Exemplo de acionamento de gatilho com DELETE e momento BEFORE EACH ROW
Fonte: Adaptado de Oracle (2016)

No exemplo, estamos apagando todas as 14 linhas da tabela EMP. Nosso gatilho BEFORE EACH ROW é executado 14 vezes, como mostram as 14 mensagens de DELETE – BEFORE EACH ROW do nosso teste. O gatilho é acionado uma vez antes de cada registro ser apagado. Esse recurso permite desenvolver gatilhos que, por exemplo, impeçam que registros sejam apagados indevidamente pelo usuário.

Não se esqueça de executar o comando ROLLBACK para desfazer as ações de nosso teste e não comprometer o resultado dos demais exemplos.

Vejamos mais um exemplo:

```
SET SERVEROUTPUT ON

CREATE OR REPLACE TRIGGER testa_momento
AFTER INSERT OR UPDATE OR DELETE ON emp
FOR EACH ROW
BEGIN

   CASE
     WHEN INSERTING THEN
```

```
          DBMS_OUTPUT.PUT_LINE('Insert – AFTER EACH ROW');
     WHEN UPDATING THEN
          DBMS_OUTPUT.PUT_LINE('Update – AFTER EACH ROW');
     WHEN DELETING THEN
          DBMS_OUTPUT.PUT_LINE('Delete – AFTER EACH ROW');
  END CASE;

END;
/
```

Código-fonte 26 – Exemplo de criação de gatilho DML com AFTER EACH ROW
Fonte: Adaptado de Oracle (2016)

O exemplo mostra a criação de um gatilho com a palavra-chave AFTER, mas sem a cláusula FOR EACH ROW.

Vejamos o comportamento do gatilho criado dessa forma:

```
--
-- Acionando o gatilho BEFORE EACH ROW com o comando DELETE
--

DELETE emp;

-- Delete - AFTER EACH ROW
-- Delete - AFTER EACH ROW
-- Delete - AFTER EACH ROW
-- Delete - AFTER EACH ROW
-- Delete - AFTER EACH ROW
-- Delete - AFTER EACH ROW
-- Delete - AFTER EACH ROW
-- Delete - AFTER EACH ROW
-- Delete - AFTER EACH ROW
-- Delete - AFTER EACH ROW
-- Delete - AFTER EACH ROW
-- Delete - AFTER EACH ROW
-- Delete - AFTER EACH ROW
-- Delete - AFTER EACH ROW

ROLLBACK;
```

Código-fonte 27 – Exemplo de acionamento de gatilho
com DELETE e momento AFTER EACH ROW
Fonte: Adaptado de Oracle (2016)

No exemplo, estamos apagando todas as 14 linhas da tabela EMP. Nosso gatilho AFTER EACH ROW é executado 14 vezes, como mostram as 14 mensagens de DELETE – AFTER EACH ROW do nosso teste. O gatilho é acionado uma vez após cada registro ser apagado. Esse recurso permite desenvolver gatilhos que, por exemplo, atualizem os dados de uma tabela de estoque após um produto ser removido do cadastro.

Não se esqueça de executar o comando ROLLBACK para desfazer as ações do nosso teste e não comprometer o resultado dos demais exemplos.

Vejamos mais um exemplo:

```
SET SERVEROUTPUT ON

CREATE OR REPLACE TRIGGER testa_momento
AFTER INSERT OR UPDATE OR DELETE ON emp
BEGIN

   CASE
     WHEN INSERTING THEN
          DBMS_OUTPUT.PUT_LINE('Insert - AFTER STATEMENT');
     WHEN UPDATING THEN
          DBMS_OUTPUT.PUT_LINE('Update - AFTER STATEMENT');
     WHEN DELETING THEN
          DBMS_OUTPUT.PUT_LINE('Delete - AFTER STATEMENT');
  END CASE;

END;
/
```

Código-fonte 28 – Exemplo de criação de gatilho DML com AFTER STATEMENT
Fonte: Adaptado de Oracle (2016)

O exemplo exibe a criação de um gatilho com a palavra-chave AFTER e a cláusula FOR EACH ROW.

Vejamos o comportamento do gatilho criado dessa forma:

```
--
-- Acionando o gatilho BEFORE EACH ROW com o comando DELETE
--

DELETE emp;

-- Delete - AFTER STATEMENT

ROLLBACK;
```

Código-fonte 29 – Exemplo de acionamento de gatilho
com DELETE e momento AFTER STATEMENT
Fonte: Adaptado de Oracle (2016)

No exemplo, estamos apagando todas as 14 linhas da tabela EMP. Nosso gatilho AFTER STATEMENT é executado uma vez, como mostra a mensagem de DELETE – AFTER STATEMENT do nosso teste. O gatilho é acionado uma vez após a execução do comando.

Não se esqueça de executar o comando ROLLBACK para desfazer as ações de nosso teste e não comprometer o resultado dos demais exemplos.

## CONCLUSÃO

*Triggers* ou gatilhos são estruturas fantásticas que permitem uma automação grande de vários processos de armazenamento e tratamento de dados que, sem o uso dessa estrutura, teriam que ser acionados manualmente. Essa estrutura aumenta muito as possibilidades do que pode ser feito com dados.

# REFERÊNCIAS

DILLON, S.; BECK, C.; KYTE, T.; KALLMAN, J.; ROGERS, H. **Beginning Oracle Programming**. Apress, 2013.

FEUERSTEIN, S.; PRIBYL, B. **Oracle Pl/SqlProgramming**. 6. ed. California, USA: O'Reilly Media, 2014.

ORACLE, **Oracle Database:** PL/SQL Language Reference 12c Release 2 (12.2) B28370-05. USA: Oracle Press, 2016.

PUGA, S.; FRANÇA, E.; GOYA, M. **Banco de dados**. São Paulo: Pearson, 2015.