Arquitetura e Administração de Bases de Dados
Oracle11g PL/SQL Programming

João Costa
jcosta@isec.pt

1

Agenda

- *Triggers*
  - *De linha (ou de registo)* – **`ROW TRIGGER`**
  - *De comando* – **`STATEMENT TRIGGER`**

2

Triggers

- Semelhante a procedimentos
- Executam automaticamente em resposta a um evento
  - Comandos **DML**
  - Comandos **DDL**
  - Eventos da base de dados
- Os *triggers* **DML** estão associados a uma tabela ou vista

Triggers

- Os *triggers* executam automaticamente com base num evento DML ou de sistema
- O tempo de execução pode ser:
  - Antes (**BEFORE**) ou depois (**AFTER**) do evento
  - Ao nível da linha (**ROW**) ou do comando (**STATEMENT**)
- A cláusula **WHEN** permite processamento condicional do *trigger*

Criar um *trigger* DML

```
CREATE [OR REPLACE] TRIGGER nome_trigger
[ BEFORE | AFTER ]
[ INSERT | DELETE | UPDATE
                  | UPDATE of colunas ]
ON nome_tabela
[ REFERENCING OLD as antigo| NEW as novo ]
[ FOR EACH ROW ]
[ WHEN condições  ]
DECLARE
 -- declaração de váriáveis
BEGIN
-- Código
END;
```


Trigger DML Ex. BEFORE ... FOR EACH ROW

```
CREATE OR REPLACE TRIGGER trig_ins_venda
  BEFORE INSERT ON vendas
  FOR EACH ROW
  WHEN (new.quantidade <= 0)
DECLARE
BEGIN
   RAISE_APPLICATION_ERROR(-20000,'Quantidade inválida');
END;
/
```

```
insert into vendas values (99,sysdate,1,1,10,-1,0);

Error report -
ORA-20000: Quantidade inválida
ORA-06512: na "LEIBD2C30.TRIG_INS_VENDA", linha 3
ORA-04088: erro durante a execução do trigger 'TRIG_INS_VENDA'
```

Tempo de execução

- Quando executar o código do trigger ?
  - antes (**`BEFORE`**) ou
  - depois (**`AFTER`**) de acontecer o evento
- Scope de execução do código ?
  - Para cada linha afetada (**`ROW`**) ou
  - Por cada comando executado(**`STATEMENT`**)
- A cláusula **`WHEN`** permite processamento condicional

Eventos

- **`INSERT, UPDATE, DELETE`**
  - Usar o operador OR para incluir mais do que um evento. *ex.:*
    - `INSERT OR DELETE`
    - `INSERT OR DELETE OR UPDATE`
- No evento de UPDATE, a opção **`OF`** *nome_da_coluna*
  - **`UPDATE OF`** *`coluna1, coluna2, …`*
  - *especifica quais as colunas que, se os valores forem alterados, permitem ativar o trigger*
- **`ON`** *nome_da_tabela*
  - *apenas o nome de uma única tabela.*
  - *Ex.* ***ON*** *autores*

Corpo do *trigger*

- É um bloco PL/SQL
  - Deve incluir uma secção DECLARE
    - Se for necessário declarações
  - Pode referenciar identificadores de correlação (se row trigger)
    - Precedidos por ":" (ex: **:NEW , :OLD**)
  - Pode chamar outros subprogramas

9

Identificadores de correlação

- Variáveis especiais associadas a DML
  - Por defeito: **OLD** e **NEW**

DML Event	OLD Identifier	NEW Identifier
INSERT	---	Contains insert values
UPDATE	Contains values of the original row	Contains new value for any columns updated and original values for any columns not updated
DELETE	Contains values of the original row	Not Available (*Note: "Not Available" indicates any references would retrieve a NULL value*)

10

Identificadores de correlação **OLD** e **NEW**

```
CREATE OR REPLACE TRIGGER trig_ins_venda
  BEFORE UPDATE ON vendas
  FOR EACH ROW
DECLARE
BEGIN
   IF :NEW.QUANTIDADE > :OLD.QUANTIDADE THEN
   RAISE_APPLICATION_ERROR(-20000,'Não pode ser superior');
   END IF;
END;
```

codigo_venda	codigo_livro	codigo_cliente	quantidade	preco_unitario
212	12	21	10	12

UPDATE VENDAS SET QUANTIDADE=50
WHERE CODIGO_VENDA=212;

codigo_venda	codigo_livro	codigo_cliente	quantidade	preco_unitario
212	12	21	50	12

```
:NEW.QUANTIDADE = ??
:OLD.QUANTIDADE = ??

:NEW.PRECO_UNITARIO = ??
:OLD.PRECO_UNITARIO = ??

:NEW.CODIGO_LIVRO = ??
:OLD.CODIGO_LIVRO = ??

:NEW.PRECO_TABELA = ??
:OLD.PRECO_TABELA = ??
```


Trigger DML Ex. **BEFORE ... FOR EACH ROW**

```
CREATE OR REPLACE TRIGGER trig_ins_venda
  BEFORE INSERT ON vendas
  FOR EACH ROW
  WHEN (new.quantidade <= 0)
DECLARE
BEGIN
   RAISE_APPLICATION_ERROR(-20000,'Quantidade inválida');
END;
/
```

```
insert into vendas values (99,sysdate,1,1,-1,10,0);

Error report -
ORA-20000: Quantidade inválida
ORA-06512: na "LEIBD2C30.TRIG_INS_VENDA", linha 3
ORA-04088: erro durante a execução do trigger 'TRIG_INS_VENDA'
```

Trigger DML Ex. BEFORE ... FOR EACH ROW

```
CREATE OR REPLACE TRIGGER trig_ins_venda2
  BEFORE INSERT ON vendas
  FOR EACH ROW
DECLARE
  CURSOR C1 IS
    SELECT QUANT_EM_STOCK as STOCK
    FROM LIVROS
    WHERE CODIGO_LIVRO = :NEW.CODIGO_LIVRO;
BEGIN
  FOR R IN C1
  LOOP
   IF R.STOCK < :NEW.QUANTIDADE THEN
     RAISE_APPLICATION_ERROR(-20001,'STOCK INSUFICIENTE.MAX='||R.STOCK);
   END IF;
  END LOOP;
END;
/

insert into vendas values (99,sysdate,1,1,999999,10,0);
```


Trigger DML Ex. AFTER ... FOR EACH ROW

```
CREATE OR REPLACE TRIGGER trig_ins_venda3
  AFTER INSERT ON vendas
  FOR EACH ROW
BEGIN
  UPDATE LIVROS
  SET QUANT_EM_STOCK = QUANT_EM_STOCK -:NEW.QUANTIDADE
  WHERE CODIGO_LIVRO = :NEW.CODIGO_LIVRO;
END;
```

```
SELECT QUANT_EM_STOCK FROM LIVROS WHERE CODIGO_LIVRO=1;
QUANT_EM_STOCK
--------------
            20

insert into vendas values (99,sysdate,1,1,3,10,0);

SELECT QUANT_EM_STOCK FROM LIVROS WHERE CODIGO_LIVRO=1;
QUANT_EM_STOCK
--------------
            17
```

Trigger DML Ex. AFTER ... (COMANDO)

```
CREATE OR REPLACE TRIGGER trig_ins_venda3
  AFTER INSERT ON vendas

BEGIN
  UPDATE LIVROS
  SET QUANT_EM_STOCK = QUANT_EM_STOCK -:NEW.QUANTIDADE
  WHERE CODIGO_LIVRO = :NEW.CODIGO_LIVRO;
END;
```

```
Error report -
ORA-04082: referências NEW ou OLD não são permitidas em triggers a nível de
tabela
04082. 00000 -  "NEW or OLD references not allowed in table level triggers"
*Cause:    The trigger is accessing "new" or "old" values in a table trigger.
*Action:   Remove any new or old references.
```


Trigger DML Ex. AFTER ... (COMANDO)

```
CREATE OR REPLACE TRIGGER trig_ins_venda4
  AFTER INSERT ON vendas

BEGIN
  UPDATE LIVROS L
  SET QUANT_EM_STOCK = 1000 - (SELECT SUM(QUANTIDADE)
                        FROM VENDAS V
                        WHERE V.CODIGO_LIVRO = L.CODIGO_LIVRO);
END;
```

Num Trigger de Comando,

- não é possível utilizar o `:NEW` e/ou `:OLD` para saber qual informação do registo que ativou a execução do trigger.
- não se sabe qual(is) o(s) registo(s) afetado(s)
- apenas é possivel saber qual o tipo de comando.

Predicados condicionais

- **`IF INSERTING`**
- **`IF DELETING`**
- **`IF UPDATING`**

- Permite diferenciar o processamento para cada tipo de comando DML
  - Múltiplos comandos DML podem disparar o mesmo *trigger*

- Pode especificar uma coluna
  - IF **`UPDATING`**(‘*lastname*’) THEN…

Aplicações dos *triggers*

- *Auditing*
  - Registo de atividade na base de dados
  - Ex: Alterações de dados sensíveis (ex. vencimentos)
    - Usar um *trigger*
      - Guardar os valores originais e os novos numa tabela de registo
    - Havendo problemas no futuro existe registo da alteração

Aplicações dos *triggers*

- Integridade dos dados
  - Validações complexas ou impossíveis de fazer com restrições **`CHECK`**
  - Ex:

    Garantir que o preço não pode ser alterado para um valor inferior ao atual
    - Os valores de **`NEW`** e **`OLD`** podem ser comparados num *trigger*

Aplicações dos *triggers*

- Integridade referencial
  - Em restrições de chave forasteira
    - Se tentarmos alterar o valor da chave na "tabela "mãe"
      - Ocorre um erro se existirem registos associados na tabela "filha"
    - Um *trigger* pode fazer o *update* em cascata dos registos da tabela "filha"

Aplicações dos *triggers*

- Dados calculados
  - Colunas cujo valor seja calculado a partir dos valores de outras colunas
    - Ex: Tabela-resumo com vendas totais por produto
      - Esta tabela tem que ser atualizada em tempo real
    - Por cada nova venda registada
      - É disparado um *trigger* que atualiza o total de vendas do produto respetivo na tabela-resumo
      - É calculado o total da venda

Restrições dos *triggers*

- Não podem incluir comandos de controlo de transações
- Não podem usar os tipos de dados `LONG` e `LONG RAW`
- Erro "*mutating table*"
  - Tentar, num *trigger* de linha, modificar uma tabela que está a ser alterada pelo evento que disparou o *trigger*

Comando ALTER TRIGGER

- Usado para compilar e ativar/desativar o *trigger*

```
ALTER TRIGGER trigger_name COMPILE;
```

```
ALTER TRIGGER trigger_name DISABLE | ENABLE;
```

```
ALTER TABLE table_name DISABLE|ENABLE ALL TRIGGERS;
```


Apagar um *trigger*

```
DROP TRIGGER trigger_name;
```

Nota:

Se uma tabela ou vista for apagada, todos os *triggers* DML a ela associados serão apagados

Dicionário de dados

- Igual aos outros subprogramas
  - Excepto para ver o código fonte
    - Vista **`USER_TRIGGERS`**
      - Coluna Description – Código do cabeçalho
      - Coluna Trigger_body – Código do corpo do *trigger*

Triggers - Resumo

- Os *triggers* executam automaticamente com base num evento DML ou de sistema
- O tempo de execução pode ser:
  - Antes (**`BEFORE`**) ou depois (**`AFTER`**) do evento
  - Ao nível da linha (**`ROW`**) ou do comando (**`STATEMENT`**)
- A cláusula **`WHEN`** permite processamento condicional do *trigger*

Triggers – Resumo (cont.)

- Identificadores de correlação permitem aceder aos valores envolvidos no comando DML
  - `NEW, OLD`
- Predicados condicionais permitem um processamento diferente para cada comando DML
  - `INSERTING, UPDATING, DELETING`

Exercícios

- Crie uma tabela ***caixa*** que para cada dia, registe a quantidade de livros vendidos, e o total recebido
- Implemente um **trigger** de comando que mantenha a tabela caixa atualizada
- Implemente um **trigger** de linha que mantenha a tabela caixa atualizada

Exercícios

- Implemente um **trigger** que ao remover um cliente da tabela ***clientes***, apague da tabela *vendas* todos os livros adquiridos pelo cliente removido.
- Á medida que os livros vão sendo vendidos, o atributo ***quant_em_stock*** da tabela livros necessita ser atualizado em conformidade. Implemente um **trigger** que mantenha atualizado o atributo ***quant_em_stock***.
- Implemente um **trigger** que garanta que o nome do autor é inserido em maiúsculas

Triggers INSTEAD OF

- Permite contornar o problema das vistas "não modificáveis"
  - Um comando DML a uma vista dispara um *trigger* INSTEAD OF
  - Os comandos DML do *trigger* serão executados nas tabelas base da vista
    - Usando valores do evento que disparou o *trigger*

INSTEAD OF - Exemplo

```
CREATE VIEW vendas_hoje AS
        SELECT  codigo_venda, codigo_livro,
                quantidade, preco_unitario
        FROM    VENDAS v
        WHERE   TO_CHAR(DATA_VENDA,'dd-mm-yyyy') =
                TO_CHAR(SYSDATE   ,'dd-mm-yyyy');
```

```
insert into vendas_hoje values (99,1, 10,3);
```


INSTEAD OF - Exemplo

```
CREATE VIEW vendas_hoje AS
        SELECT  codigo_venda, codigo_livro, nome,
                quantidade, preco_unitario
        FROM    VENDAS v, clientes c
        WHERE   TO_CHAR(DATA_VENDA,'dd-mm-yyyy') =
                TO_CHAR(SYSDATE   ,'dd-mm-yyyy')
        AND     c.codigo_cliente = v.codigo_cliente;
```

```
insert into vendas_hoje values (99,1,'Carlos Mota',10,3);

Error report -
SQL Error: ORA-01776: não é possível modificar mais de uma tabela base
através de uma vista de junção
01776. -  "cannot modify more than one base table through a join view"
*Cause:    Columns belonging to more than one underlying table were either
inserted into or updated.
*Action:   Phrase the statement as two or more separate statements.
```

INSTEAD OF - Exemplo

```
CREATE OR REPLACE TRIGGER trig_vendas_hoje
  INSTEAD OF INSERT ON vendas_hoje
  FOR EACH ROW
DECLARE
  codcliente CLIENTES.CODIGO_CLIENTE%TYPE;
BEGIN
  SELECT codigo_cliente INTO codcliente
  FROM  clientes
  WHERE UPPER(nome) = UPPER(:NEW.nome);

  INSERT INTO vendas VALUES ( :new.codigo_venda,
                SYSDATE, codcliente, :new.codigo_livro,
                :new.quantidade,  :new.preco_unitario,
                :new.quantidade * :new.preco_unitario );
END;
/
```


Triggers – Resumo (cont.)

- *Triggers* INSTEAD OF fornecem um mecanismo para alterar vistas "não modificáveis"
- O comando ALTER TRIGGER permite compilar e ativar/desativar um *trigger*
- A vista USER_TRIGGERS, do dicionário de dados, permite ver o código dos *triggers*