Laboratório de Bases de Dados

Prof. José Fernando Rodrigues Jr.

Aula 5 – Triggers

Material original editado: Profa. Elaine Parros Machado de Sousa

Triggers em Oracle

- Tipos
  - Para tabelas
    - Triggers de DML (INSERT, UPDATE, DELETE)
    - Triggers de Sistema (DDL, e logs)

Triggers de DML

➔ Programação orientada a eventos com 4 dimensões:

1. **Tabela desencadeadora**
2. **Instrução de disparo**

   INSERT

   UPDATE

   DELETE
3. ***Timing***

   BEFORE

   AFTER
4. **Nível**

   linha

   instrução

Triggers

- Para que usar?
  - criar conteúdo de uma coluna derivado de outras
  - atualizar tabelas em função da atualização de uma determinada tabela
  - criar *logs* – segurança ➔ **auditoria**
  - restrições de consistência e validade que não possam ser implementadas com *constraints* – por exemplo, envolvendo ciclos com múltiplas tabelas

Atributo derivado

➔ **Computação de atributo derivado:**

```
CREATE OR REPLACE TRIGGER NroDeAlunos
AFTER INSERT ON Matricula
FOR EACH ROW /* nível de linha */
DECLARE
   NroAlunos NUMBER;
BEGIN
   SELECT Nalunos INTO NroAlunos
   FROM Turma
   WHERE Sigla = :new.Sigla and Numero = :new.Numero;

   UPDATE Turma set NAlunos = NroAlunos + 1
   WHERE Sigla = :new.Sigla and Numero = :new.Numero;
EXCEPTION
 WHEN OTHERS
       THEN dbms_output.put_line('Erro nro:  ' || SQLCODE
                          || '. Mensagem: ' || SQLERRM );
END NroDeAlunos;
```

identificador `:new` refere-se à tabela `Matricula`

Atributo derivado

➔ **Computação de atributo derivado:**

```
CREATE OR REPLACE TRIGGER UpdateTotalAlunos
AFTER INSERT OR DELETE ON Matricula
FOR EACH ROW
BEGIN
   IF INSERTING THEN
    UPDATE Professor
    SET TotalAlunos = TotalAlunos + 1
    WHERE NFunc = (SELECT Professor FROM Disciplina WHERE Sigla = :NEW.Sigla);

   ELSIF DELETING THEN
    UPDATE Professor
    SET TotalAlunos = TotalAlunos - 1
    WHERE NFunc = (SELECT Professor FROM Disciplina WHERE Sigla = :OLD.Sigla);
  END IF;
END;

ALTER TABLE Professor ADD TotalAlunos NUMBER DEFAULT 0;
```

Auditoria

➔ **Auditoria:**

```
CREATE OR REPLACE TRIGGER AuditMatricula
AFTER INSERT OR UPDATE OR DELETE ON Matricula
FOR EACH ROW
DECLARE
  operacao VARCHAR2(30);
BEGIN
   IF INSERTING THEN operacao := 'INSERT';
        ELSIF UPDATING THEN operacao := 'UPDATE';
        ELSIF DELETING THEN operacao := 'DELETE';
   END IF;

 insert into output values(op_seq.nextval, operacao,:old.Aluno||'-'||:old.nota);
 insert into output values(op_seq.nextval, operacao,:new.Aluno||'-'||:new.nota);

 insert into output values(0, '------------', '-----------');
END;
```

```
*Output
DROP TABLE output;
CREATE TABLE output(iNr NUMBER, operacao VARCHAR2(30), msg varchar2(200));
```

```
Drop sequence op_seq;
CREATE SEQUENCE op_seq START WITH    1  INCREMENT BY   1;
```

Triggers de DML

- **Identificadores de tuplas** – variáveis de vínculo PL/SQL (p/ triggers com nível de linha)
  - sempre vinculados à tabela desencadeadora do trigger
  - pseudoregistros do tipo ***tabela_desencadeadora*%ROWTYPE**

instrução / identificador	`:old`	`:new`
`INSERT`	NULL	valores que serão inseridos
`UPDATE`	valores antes da atualização	novos valores para a atualização
`DELETE`	valores antes da remoção	NULL

BEFORE é usado, entre outras coisas, para validar, editar, e até mesmo impedir uma operação.

AFTER é usado para desencadear operações em decorrência de outras, por exemplo, para computar atributos derivados.
Ou para auditar operações que foram executadas.

O que um trigger não executa?

- **DML sobre tabelas mutantes**
- **Uma tabela mutante é um tabela que está sendo alterada por INSERT, UPDATE ou DELETE em tempo BEFORE**
- Erro ORA-04091: table is mutating, trigger/function may not see it

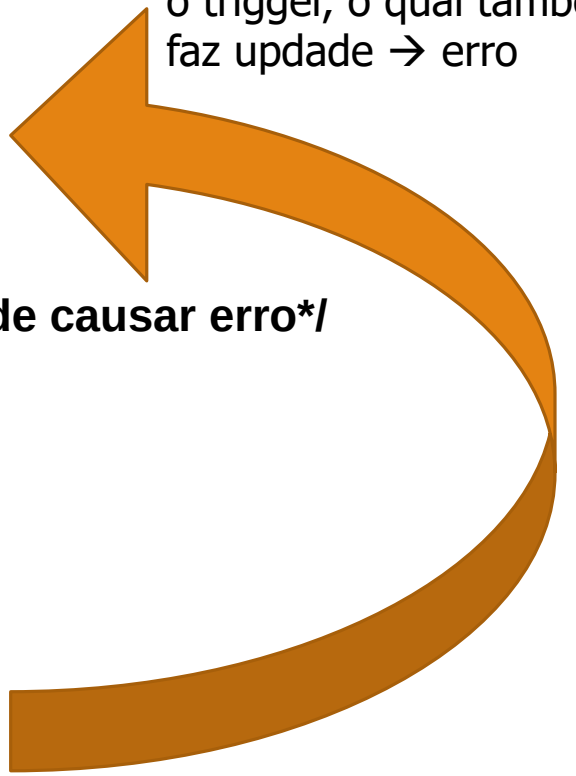
O que um trigger não executa?

```
CREATE OR REPLACE TRIGGER tabela_mutante
BEFORE UPDATE ON Aluno
FOR EACH ROW /* nível de linha */
BEGIN
   UPDATE Aluno
   SET NOME = UPPER(NOME)
      WHERE NUSP = :old.NUSP;
   /*Mesmo um simples SELECT INTO a partir de Aluno pode causar erro*/
EXCEPTION
   WHEN OTHERS THEN
     raise_application_error(-20001, SQLERRM);
END tabela_mutante;

INSERT INTO Aluno VALUES(111,'Carla', 20, null, 'Ibate');
UPDATE Aluno SET Nome = 'Carlos' WHERE NUSP = 111;
```

O que um trigger não executa?

```
CREATE OR REPLACE TRIGGER tabela_mutante
BEFORE UPDATE ON Aluno
FOR EACH ROW /* nível de linha */
BEGIN
   UPDATE Aluno
   SET NOME = UPPER(NOME)
      WHERE NUSP = :old.NUSP;
   /*Mesmo um simples SELECT INTO a partir de Aluno pode causar erro*/
EXCEPTION
   WHEN OTHERS THEN
     raise_application_error(-20001, SQLERRM);
END tabela_mutante;

INSERT INTO Aluno VALUES(111,'Carla', 20, null, 'Ibate');
UPDATE Aluno SET Nome = 'Carlos' WHERE NUSP = 111;
```

Um update desencadeo
o trigger, o qual também
faz updade → erro



O que um trigger não executa?

ORA-20001: ORA-04091: a tabela D2373891.ALUNO é mutante; talvez o gatilho/função não possa localizá-la

ORA-06512: em "D2373891.TABELA_MUTANTE", line 8
ORA-04088: erro durante a execução do gatilho 'D2373891.TABELA_MUTANTE'

Como proceder então?

O que um trigger não executa?

Este caso possui uma solução; ao invés de se disparar outra operação DML – deve-se editar a variável :new (possível apenas em timing BEFORE).

```
CREATE OR REPLACE TRIGGER tabela_mutante
BEFORE UPDATE ON Aluno
FOR EACH ROW /* nível de linha */
BEGIN
      :new.NOME := UPPER(:new.NOME);
EXCEPTION
   WHEN OTHERS THEN
    raise_application_error(-20001, SQLERRM);
END tabela_mutante;

INSERT INTO Aluno VALUES(111,'Carla', 20, null, 'Ibate');
```

UPDATE Aluno SET Nome = 'Carlos' WHERE NUSP = 111; → ok

O que um trigger não executa?

Obviamente, este recurso (edição de :new) não faz sentido em AFTER, o que causa erro de compilação.

Já :old jamais pode ser editado, por razões óbvias.

Nomes para :new e :old

```
CREATE OR REPLACE TRIGGER AcertaNota
BEFORE INSERT OR UPDATE ON Matricula
/*especificando nomes para NEW e OLD ...*/
   REFERENCING new AS nova_matricula
   FOR EACH ROW
   WHEN (:nova_matricula.nota < 0)

   BEGIN
      :nova_matricula.nota := 0;
   END AcertaNota;
```

O que um trigger não executa?

- **Não são permitidos comandos transacionais (SET TRANSACTION, COMMIT, SAVEPOINT, e ROLLBACK) dentro de um trigger**

- **Por consequência, não são permitidos comandos DDL (CREATE, ALTER, e DROP), pois eles disparam COMMIT automaticamente**

- **Assim, um trigger fica sujeito à transação (commit/rollback) definida na sessão em que o trigger foi disparado**

Consistência

➔ **Consistência**

Exemplo: máximo de 10 alunos por turma

```
CREATE OR REPLACE TRIGGER CheckMaxAlunos
BEFORE INSERT ON Matricula
FOR EACH ROW
DECLARE
  v_TotalAlunos NUMBER;
BEGIN
  -- Obtendo o número de alunos já matriculados na turma da
   disciplina
  SELECT NAlunos INTO v_TotalAlunos
  FROM Turma
  WHERE Sigla = :NEW.Sigla AND Numero = :NEW.Numero;

  -- Verificando se o número de alunos é 10 ou mais
  IF v_TotalAlunos >= 10 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Não é possível realizar a
   matrícula, pois a turma já possui 10 alunos.');
  END IF;
END;
```

Consistência

➔ **Consistência**

Exemplo: um professor pode lecionar no máximo 3 disciplinas

```
CREATE OR REPLACE TRIGGER CheckMaxDisciplinasProfessor
BEFORE INSERT OR UPDATE ON Disciplina
FOR EACH ROW
DECLARE
  v_TotalDisciplinas NUMBER;
BEGIN
  -- Contando o número de disciplinas que o professor já leciona
  SELECT COUNT(*) INTO v_TotalDisciplinas
  FROM Disciplina
  WHERE Professor = :NEW.Professor;

  -- Verificando se o professor já leciona 3 disciplinas ou mais
  IF v_TotalDisciplinas >= 3 THEN
    RAISE_APPLICATION_ERROR(-20002, 'O professor já ministra 3
   disciplinas e não pode assumir outra.');
  END IF;
END;
/
```

Consistência

➔ **Consistência**

Exemplo: um professor pode lecionar no máximo 3 disciplinas

```
CREATE OR REPLACE TRIGGER CheckMaxDisciplinasProfessor
BEFORE INSERT OR UPDATE ON Disciplina
FOR EACH ROW
DECLARE
  v_TotalDisciplinas NUMBER;
BEGIN
  -- Contando o número de disciplinas que o professor já leciona
  SELECT COUNT(*) INTO v_TotalDisciplinas
  FROM Disciplina
  WHERE Professor = :NEW.Professor;

  -- Verificando se o professor já leciona 3 disciplinas ou mais
  IF v_TotalDisciplinas >= 3 THEN
    RAISE_APPLICATION_ERROR(-20002, 'O professor já ministra 3
   disciplinas e não pode assumir outra.');
  END IF;
END;
/
```

Auditoria

➔ **Auditoria:**

```
CREATE OR REPLACE TRIGGER LogProfessor
AFTER INSERT OR UPDATE OR DELETE ON Professor
FOR EACH ROW

DECLARE
  v_operacao CHAR;
BEGIN
  IF INSERTING THEN v_operacao := 'I';
  ELSIF UPDATING THEN v_operacao := 'U';
  ELSIF DELETING THEN v_operacao := 'D';
  END IF;

  INSERT INTO logTabelaProfessor VALUES (USER, SYSDATE, v_operacao);
END LogProfessor;


CREATE TABLE logTabelaProfessor (usuario VARCHAR(100),data DATE,op CHAR);
INSERT INTO Professor VALUES (1, 'Joao Silva', 40, 'Doutorado');
ROLLBACK;
SELECT * FROM logTabelaProfessor;
```

Auditoria

No entanto, esta operação de log **não é imune** a uma operação de **ROLLBACK**.

➔ **O log não é eficiente** em termos de **segurança** sobre o que é executado no banco de dados

Auditoria

Um trigger, no entanto, pode executar comandos transacionais ou comandos DDL desde que isto seja "avisado" por meio do comando PRAGMA AUTONOMOUS_TRANSACTION

Trata-se de uma **alternativa importante** para se garantir a autonomia das triggers com relação à auditoria do Banco de Dados

Auditoria

➔ **Auditoria:**

```
CREATE OR REPLACE TRIGGER LogProfessor
AFTER INSERT OR UPDATE OR DELETE ON Professor
FOR EACH ROW

DECLARE
 PRAGMA AUTONOMOUS_TRANSACTION;
  v_operacao CHAR;
BEGIN
  IF INSERTING THEN v_operacao := 'I';
  ELSIF UPDATING THEN v_operacao := 'U';
  ELSIF DELETING THEN v_operacao := 'D';
  END IF;

  INSERT INTO logTabelaProfessor VALUES (USER, SYSDATE, v_operacao);
  COMMIT;
END LogProfessor;


CREATE TABLE logTabelaProfessor (usuario VARCHAR(100),data DATE,op CHAR);
INSERT INTO Professor VALUES (1, 'Joao Silva', 40, 'Doutorado');
ROLLBACK;
SELECT * FROM logTabelaProfessor;
```

Auditoria

Agora, o trigger escreveu dados que **não serão apagados, mesmo com o ROLLBACK**.

Os dados da tabela **Professor serão apagados**, pois a transação em andamento na seção **não é influenciada pela transação autônoma do trigger**.

Auditoria

Ok, mas e se o usuário fizer uma operação **DELETE ou UPDATE na tabela de log**?

Neste caso podemos definir **uma trigger BEFORE que impede a operação**.

Auditoria

➔ **Auditoria:**

```
CREATE OR REPLACE TRIGGER logTabelaProfessorImpedeEdicao
BEFORE UPDATE OR DELETE ON logTabelaProfessor
FOR EACH ROW
BEGIN
 raise_application_error(-20001, 'Não, não, você não pode mexer no log!');
END logTabelaProfessorImpedeEdicao;

delete from logTabelaProfessor;

Relatório de erro:
Erro de SQL: ORA-20001: Não, não, você não pode mexer no log!
```

Triggers de Sistema

- *Triggers* disparados por:
  - instruções DDL
    - CREATE – before/after
    - ALTER - before/after
    - DROP - before/after
    - DDL - before/after
    - ...
  - eventos do banco de dados
    - STARTUP - after
    - SHUTDOWN - before
    - LOGON - after
    - LOGOFF - before
    - SERVERERROR – after
    - ...:
- Níveis
  - **`DATABASE`**
  - **`SCHEMA`**
    - do usuário que criou o *trigger* ou de outro usuário

Triggers de Sistema

Dados disponíveis:

- sysdate
- sys_context('USERENV','OS_USER')
- sys_context('USERENV','CURRENT_USER')
- sys_context('USERENV','HOST')
- sys_context('USERENV','TERMINAL')
- ora_dict_obj_owner
- ora_dict_obj_type
- ora_dict_obj_name
- ora_sysevent

Sistema

```
-- conectado com role DBA
CREATE OR REPLACE TRIGGER TodosUsuarios
AFTER LOGON ON DATABASE
BEGIN
 INSERT INTO logUser VALUES (USER, 'Trigger TodosUsuarios');
END;
```

Apenas em tempo AFTER

```
-- conectado com role USUÁRIO NÃO DBA
CREATE OR REPLACE TRIGGER UsuarioLogado
AFTER LOGON ON SCHEMA
BEGIN
    INSERT INTO logUser VALUES (USER, 'Trigger UsuarioLogado');
END;
```

Esquema

```
CREATE OR REPLACE TRIGGER DropBloqueado
BEFORE DROP ON SCHEMA
BEGIN
  -- Bloqueia qualquer tentativa de exclusão de tabelas
  IF DICTIONARY_OBJ_TYPE = 'TABLE' THEN
    RAISE_APPLICATION_ERROR(-20004, 'A exclusão de tabelas não é permitida neste
   esquema.');
  END IF;
END;
-- ----------------------------------------------
CREATE OR REPLACE TRIGGER AlterBloqueado
BEFORE ALTER ON SCHEMA
BEGIN
   IF DICTIONARY_OBJ_TYPE = 'TABLE' AND DICTIONARY_OBJ_NAME = 'ALUNO' THEN
    RAISE_APPLICATION_ERROR(-20007, 'Alterações na tabela Aluno não são
   permitidas.');
  END IF;
END;
```

Tempo BEFORE ou AFTER

Procedures X Triggers

Procedure/Function	Trigger
bloco identificado PL/SQL	bloco identificado PL/SQL
pode ser usado em pacotes ou mesmo em triggers	objeto independente
recebe parâmetros	não recebe parâmetros, usa apenas new e old
executado explicitamente	executado (disparado) implicitamente – execução orientada a eventos

Triggers

- Para que usar?
  - restrições de consistência e validade que não possam ser implementadas com *constraints* – por exemplo, envolvendo ciclos com múltiplas tabelas
  - criar conteúdo de uma coluna derivado de outras
  - atualizar tabelas em função da atualização de uma determinada tabela
  - criar *logs* – segurança ➔ **auditoria**

Recursos

- SQL *Reference*
- *Database Concepts*
- *Application Developer's Guide – Fundamentals*
  - usando triggers: informações, exemplos, eventos, atributos,...

PRÁTICA 5