



Laboratório de Bases de Dados

Prof. José Fernando Rodrigues Jr.

Aula 7 – 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)

- Para visões

- Triggers de DML Instead-OF (INSERT, UPDATE, DELETE)



Triggers de DML

- **Tabela desencadeadora**
- **Instrução de disparo**
 - INSERT
 - UPDATE
 - DELETE
- ***Timing***
 - BEFORE
 - AFTER
- **Nível**
 - linha
 - instrução

Exemplo

Turma = {Sigla, Numero, NAlunos}

Matrícula = {Sigla, Numero, Aluno, Ano, Nota}

```
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;
EXCEPTION
    ....
END NroDeAlunos;
```

identificador :new
refere-se à tabela
Matricula

```
DELETE FROM matricula WHERE aluno = 222;
```

Quais dados são disponíveis?

```
CREATE OR REPLACE TRIGGER AuditPartido
AFTER INSERT OR UPDATE OR DELETE ON LE08_Partido
FOR EACH ROW
DECLARE
    v_op VARCHAR2(30);
BEGIN
    IF INSERTING THEN v_op := 'INSERT';
    ELSIF UPDATING THEN v_op := 'UPDATE';
    ELSIF DELETING THEN v_op := 'DELETE';
    END IF;

    insert into output values(op_seq.nextval, v_op, :old.sigla||'-'||:old.nome);
    insert into output values(op_seq.nextval, v_op, :new.sigla||'-'||:new.nome);

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

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



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

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



O que um trigger executa?

- **Suponha os seguintes comandos SQL**

- **INSERT**

- INSERT INTO LBD01_VINCULO_USP(NROUSP, TIPOVINC, NOME, DATAINGRESSO) VALUES(21, 2, 'João', '05/09/2002')
 - INSERT INTO LBD01_VINCULO_USP(NROUSP, TIPOVINC, NOME, DATAINGRESSO) VALUES(22, 1, 'Gilberto', '09/05/2000')
 - INSERT INTO LBD01_VINCULO_USP(NROUSP, TIPOVINC, NOME, DATAINGRESSO) VALUES(23, 3, 'Alfredo', '03/04/2002')

- **UPDATE**

- UPDATE LBD01_VINCULO_USP SET NOME = UPPER(NOME) WHERE NROUSP > 20;

- **DELETE**

- DELETE LBD01_VINCULO_USP WHERE NROUSP > 20;

- **Quais são os valores de :old e :new para cada uma destas as operações?**



O que um trigger executa?

- Primeiramente, o corpo do trigger será executado 3 vezes para cada tipo de operação
- Os valores para cada execução considerando cada operação serão:

INSERT	:old						:new					
	NroUSP	TipoVinc	Nome	DataIngresso	DataNascimento	Ativo	NroUSP	TipoVinc	Nome	DataIngresso	DataNascimento	Ativo
	NULL	NULL	NULL	NULL	NULL	NULL	21	2	João	05/09/2002	NULL	y
	NULL	NULL	NULL	NULL	NULL	NULL	22	1	Gilberto	09/05/2000	NULL	y
	NULL	NULL	NULL	NULL	NULL	NULL	23	3	Alfredo	03/04/2002	NULL	y



O que um trigger executa?

- Primeiramente, o corpo do trigger será executado 3 vezes para cada tipo de operação
- Os valores para cada execução considerando cada operação serão:

:old

:new

INSERT

NroUSP	TipoVinc	Nome	DataIngresso	DataNascimento	Ativo
NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL

NroUSP	TipoVinc	Nome	DataIngresso	DataNascimento	Ativo
21	2	João	05/09/2002	NULL	y
22	1	Gilberto	09/05/2000	NULL	y
23	3	Alfredo	03/04/2002	NULL	y

UPDATE

NroUSP	TipoVinc	Nome	DataIngresso	DataNascimento	Ativo
21	2	João	05/09/2002	NULL	y
22	1	Gilberto	09/05/2000	NULL	y
23	3	Alfredo	03/04/2002	NULL	y

NroUSP	TipoVinc	Nome	DataIngresso	DataNascimento	Ativo
21	2	JOÃO	05/09/2002	NULL	y
22	1	GILBERTO	09/05/2000	NULL	y
23	3	ALFREDO	03/04/2002	NULL	y

O que um trigger executa?

- Primeiramente, o corpo do trigger será executado 3 vezes para cada tipo de operação
- Os valores para cada execução considerando cada operação serão:

:old

:new

INSERT

NroUSP	TipoVinc	Nome	DataIngresso	DataNascimento	Ativo
NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL

NroUSP	TipoVinc	Nome	DataIngresso	DataNascimento	Ativo
21	2	João	05/09/2002	NULL	y
22	1	Gilberto	09/05/2000	NULL	y
23	3	Alfredo	03/04/2002	NULL	y

UPDATE

NroUSP	TipoVinc	Nome	DataIngresso	DataNascimento	Ativo
21	2	João	05/09/2002	NULL	y
22	1	Gilberto	09/05/2000	NULL	y
23	3	Alfredo	03/04/2002	NULL	y

NroUSP	TipoVinc	Nome	DataIngresso	DataNascimento	Ativo
21	2	JOÃO	05/09/2002	NULL	y
22	1	GILBERTO	09/05/2000	NULL	y
23	3	ALFREDO	03/04/2002	NULL	y

DELETE

NroUSP	TipoVinc	Nome	DataIngresso	DataNascimento	Ativo
21	2	JOÃO	05/09/2002	NULL	y
22	1	GILBERTO	09/05/2000	NULL	y
23	3	ALFREDO	03/04/2002	NULL	y

NroUSP	TipoVinc	Nome	DataIngresso	DataNascimento	Ativo
NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL

Esta análise é válida tanto para **BEFORE** quanto para **AFTER**.

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

AFTER é usado para, entre outras coisas, **desencadear operações** em decorrência de outras, e para **auditar** operações que foram executadas.

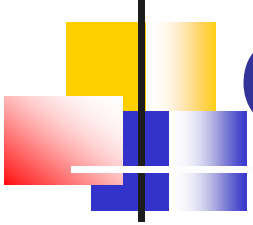
INSERT

UPDATE

NroUSP	TipoVinc	Nome	DataIngresso	DataNascimento	Ativo	NroUSP	TipoVinc	Nome	DataIngresso	DataNascimento	Ativo
21	2	João	05/09/2002	NULL	y	21	2	JOÃO	05/09/2002	NULL	y
22	1	Gilberto	09/05/2000	NULL	y	22	1	GILBERTO	09/05/2000	NULL	y
23	3	Alfredo	03/04/2002	NULL	y	23	3	ALFREDO	03/04/2002	NULL	y

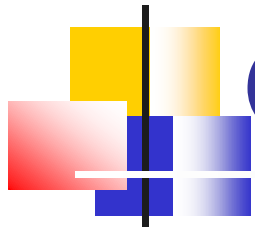
DELETE

NroUSP	TipoVinc	Nome	DataIngresso	DataNascimento	Ativo	NroUSP	TipoVinc	Nome	DataIngresso	DataNascimento	Ativo
21	2	João	05/09/2002	NULL	y	NULL	NULL	NULL	NULL	NULL	NULL
22	1	Gilberto	09/05/2000	NULL	y	NULL	NULL	NULL	NULL	NULL	NULL
23	3	Alfredo	03/04/2002	NULL	y	NULL	NULL	NULL	NULL	NULL	NULL



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
- 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 LE01_ESTADO
FOR EACH ROW /* nível de linha */
BEGIN
    UPDATE LE01_ESTADO
    SET NOME = UPPER(NOME)
    WHERE Sigla = :old.Sigla;
    /*Mesmo um simples SELECT INTO a partir de LE01_ESTADO pode causar erro*/
EXCEPTION
    WHEN OTHERS THEN
        raise_application_error(-20001, SQLERRM);
END tabela_mutante;
```



```
INSERT INTO LE01 VALUES('PA','Para');
UPDATE LE01 SET SIGLA = 'PR' WHERE SIGLA = 'PA';
```

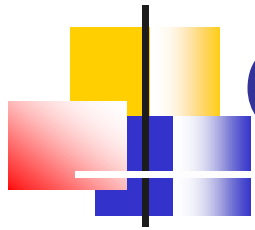


O que um trigger não executa?

Erro de SQL: ORA-20001: ORA-04091: a tabela
SCC0241_2012_2.L01_ESTADO é **mutante**; talvez o
gatilho/função não possa localizá-la

ORA-06512: em "SCC0241_2012_2.TABELA_MUTANTE", line 7
ORA-04088: erro durante a execução do gatilho
'SCC0241_2012_2.**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 (apenas em timing BEFORE).

```
CREATE OR REPLACE TRIGGER tabela_mutante
BEFORE UPDATE ON LE01_ESTADO
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 LE01 VALUES('PA','Para');
UPDATE LE01 SET SIGLA = 'PR' WHERE SIGLA = 'PA'; → ok
```



O que um trigger não executa?

Este caso
editar

deve-se

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.

CREATE
BEFORE
FOR EACH
BEGIN
:new
EXCEPT
WHEN
raise_
END tabela_mutante;

INSERT INTO LE01 VALUES('PA' 'Para');

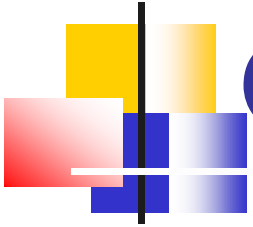
Nomes para :new e :old

```
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;
```

Transações

- Operações **DDL não estão sujeitas a transações**, isto é, um CREATE TABLE não é desfeito com **ROLLBACK** e não requer COMMIT para ser efetivado
- De fato, **uma operação DDL dispara automaticamente um COMMIT**, para que todos os dados pendentes sejam validados antes que o esquema sofra alterações



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 definida na sessão em que o trigger foi disparado

Exemplo – criando log

```
CREATE OR REPLACE TRIGGER LogEstado  
AFTER INSERT OR UPDATE OR DELETE ON LE01_ESTADO  
FOR EACH ROW
```

```
DECLARE
```

```
    v_operacao CHAR;
```

```
BEGIN
```

```
    IF INSERTING THEN v_operacao := 'I';
```

```
        ELIF UPDATING THEN v_operacao := 'U';
```

```
        ELIF DELETING THEN v_operacao := 'D';
```

```
    END IF;
```

```
    INSERT INTO logTabelaEstado
```

```
        VALUES (USER, SYSDATE, v_operacao);
```

```
END LogEstado;
```

```
CREATE TABLE logTabelaEstado(usuario VARCHAR(100),data DATE, op CHAR);
```

```
INSERT INTO LE01 VALUES('MT','Mato Grosso')
```

```
ROLLBACK
```

```
select * from logTabelaEstado
```

Exemplo – criando log

```
CREATE OR REPLACE TRIGGER LogEstado
```

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

```
VALUES (USER, SYSDATE, v_operacao);  
END LogEstado;
```

```
* CREATE TABLE logTabelaEstado(usuario VARCHAR(100),data DATE, op CHAR);
```



O que um trigger não executa?

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

Exemplo – criando log

```
CREATE OR REPLACE TRIGGER LogEstado
AFTER INSERT OR UPDATE OR DELETE ON LE01_ESTADO
FOR EACH ROW
DECLARE
    v_operacao CHAR;
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    IF INSERTING THEN v_operacao := 'I';
        ELIF UPDATING THEN v_operacao := 'U';
        ELIF DELETING THEN v_operacao := 'D';
    END IF;

    INSERT INTO logTabelaEstado
        VALUES (USER, SYSDATE, v_operacao);

    COMMIT;
END LogEstado;

* CREATE TABLE logTabelaEstado(usuario VARCHAR(100),data DATE, op CHAR);
  INSERT INTO LE01 VALUES('MT','Mato Grosso')
  ROLLBACK
  select * from logTabelaEstado
```

Exemplo – criando log

```
CREATE OR REPLACE TRIGGER LogEstado  
AFTER INSERT OR UPDATE OR DELETE ON LE01_ESTADO  
FOR EACH ROW  
DECLARE
```

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

Os dados da tabela **LE01** serão **apagados**, pois a transação em andamento na seção **não é influenciada pela transação autônoma da trigger.**

```
END LogEstado;
```

```
* CREATE TABLE logTabelaEstado(usuario VARCHAR(100),data DATE, op CHAR);  
  INSERT INTO LE01 VALUES('MT','Mato Grosso')  
  ROLLBACK  
  select * from logTabelaEstado
```


Exemplo – criando log

```
CREATE OR REPLACE TRIGGER LogEstado  
AFTER INSERT OR UPDATE OR DELETE ON LE01_ESTADO  
FOR EACH ROW  
DECLARE
```

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**.

```
END LogEstado;
```

```
* CREATE TABLE logTabelaEstado(usuario VARCHAR(100),data DATE, op CHAR);  
  INSERT INTO LE01 VALUES('MT','Mato Grosso')  
  ROLLBACK  
  select * from logTabelaEstado
```

Exemplo – criando log

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

```
delete from logTabelaEstado
```

Relatório de erro:

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

ORA-06512: em "JUNIO.LOGESTADOIMPEDEEDICAO", line 2

ORA-04088: erro durante a execução do gatilho 'JUNIO.LOGESTADOIMPEDEEDICAO'



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

■ *Triggers* disparados por:

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

■ **DATABASE**

■ **SCHEMA**

- do usuário que criou o *trigger* ou de outro usuário

Exemplo

-- conectado com role DBA

CREATE OR REPLACE TRIGGER TodosUsuarios

AFTER LOGON ON DATABASE

BEGIN

INSERT INTO logUser VALUES (USER, 'Trigger TodosUsuarios');

END;

-- conectado com role USUÁRIO NÃO DBA

CREATE OR REPLACE TRIGGER UsuarioLogado

AFTER LOGON ON SCHEMA

BEGIN

INSERT INTO logUser VALUES (USER, 'Trigger UsuarioLogado');

END;



Triggers Instead-of

- Usados para alterar **visões não atualizáveis** e **visões de junção atualizáveis**
 - permitem fazer as atualizações de maneira adequada para a **semântica da aplicação**
- Executa o corpo do *trigger* **AO INVÉS** da instrução que o acionou

Exemplo

Professor = {Nome, NFunc, Idade, Titulação}

Disciplina = {Sigla, Nome, NCred, Professor, Livro}



```
CREATE VIEW prof_disciplina AS
  SELECT d.sigla, d.nome, p.nfunc, p.nome as professor
 FROM Disciplina d, Professor p
 WHERE d.professor = p.nfunc;
```

-- qual o efeito do comando abaixo na base de dados???

```
DELETE FROM prof_disciplina WHERE nfunc = 111;
```

Exemplo

Professor = {Nome, NFunc, Idade, Titulação}

Disciplina = {Sigla, Nome, NCred, Professor, Livro}

Nesta view, apenas a tabela Disciplina terá preservação de chave, portanto a operação de delete sobre um atributo da tabela Professor causará um erro: tentativa de atualização de tabela sem preservação de chave.

fessor


-- qual o efeito do comando abaixo na base de dados???

DELETE FROM prof_disciplina WHERE nfunc = 111;

Exemplo

Professor = {Nome, NFunc, Idade, Titulação}

Disciplina = {Sigla, Nome, NCred, Professor, Livro}



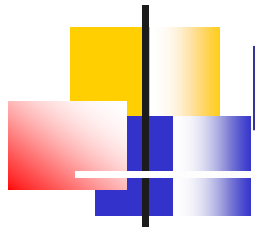
```
CREATE OR REPLACE TRIGGER RemoveProfDisciplina  
INSTEAD OF DELETE ON prof_disciplina  
FOR EACH ROW /* opcional - sempre é nível de linha */
```

```
BEGIN
```

```
UPDATE Disciplina SET professor = null  
WHERE professor = :old.nfunc;
```

```
DELETE FROM Professor WHERE nfunc = :old.nfunc;
```

```
END RemoveProfDisciplina;
```



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 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 7
