# FGA0137
# Sistemas de Banco de Dados 1

Prof. Maurício Serrano

Material original: Prof. Jose Fernando Rodrigues Junior

**2021/2**

# Triggers e Stored Procedures

Módulo 5

# Triggers em PostgreSQL

- Todos os bancos de dados comerciais possuem uma **linguagem procedural** auxiliar para a definição de **procedimentos armazenados**
  - Definição de regras de negócio
  - Especificação de restrições de integridade não possíveis no modelo relacional
  - Cálculo de atributos derivados
  - Auditoria
  - Adição de funcionalidades ao banco

# Triggers em PostgreSQL

- PostgreSQL não possui uma única linguagem procedural, este SGBD aceita várias linguagens e pode ser estendido para outras
  - PL/pgSQL
  - PL/Tcl
  - PL/Perl
  - PL/Python
  - Entre outras não distribuídas com o SGBD: PL/Java, PL/PHP, PL/Py, PL/R, PL/Ruby, PL/Scheme e PL/sh

# Triggers em PostgreSQL

- Um dos principais usos de linguagens procedurais em bancos de dados é a definição de **gatilhos**
- Gatilhos são execuções disparadas pelo banco em função de EVENTOS que ocorrem
- Um evento ocorre
  - Em uma tabela
  - De acordo com uma operação DML (INSERT, UPDATE ou DELETE)
  - Em um momento: antes ou depois (BEFORE ou AFTER)

# Triggers de DML

- **Tabela desencadeadora**
- **Instrução de disparo**
  - INSERT
  - UPDATE
  - DELETE
- ***Timing***
  - BEFORE
  - AFTER
- ***Nível***
  - Row (for each row)
  - Statement

# Triggers

- **Identificadores de correlação** – variáveis de vínculo PL/SQL
  - sempre vinculados à tabela desencadeadora do trigger

| **Before ou After** | | |
|---|---|---|
| **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 |

# Exemplo

Turma = {Sigla, Letra, NAlunos}

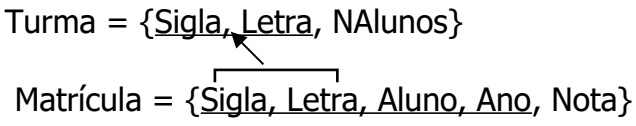
Matrícula = {Sigla, Letra, Aluno, Ano, Nota}

- Deseja-se manter a tabela Turma atualizada de acordo com as remoções ocorridas na tabela Matrícula

```
CREATE OR REPLACE FUNCTION update_turma() RETURNS trigger
   AS $update_turma$
BEGIN
    UPDATE Turma SET NAlunos = NAlunos - 1 WHERE Sigla =
   old.Sigla AND Letra = old.Letra;
   RETURN NULL;
END;
$update_turma$ LANGUAGE plpgsql;

CREATE TRIGGER NroDeAlunos
   AFTER DELETE ON Matricula
   FOR EACH ROW EXECUTE PROCEDURE update_turma();
```

**identificador `old` refere-se aos valores antes do update na tabela `Matricula`**

# Exemplo

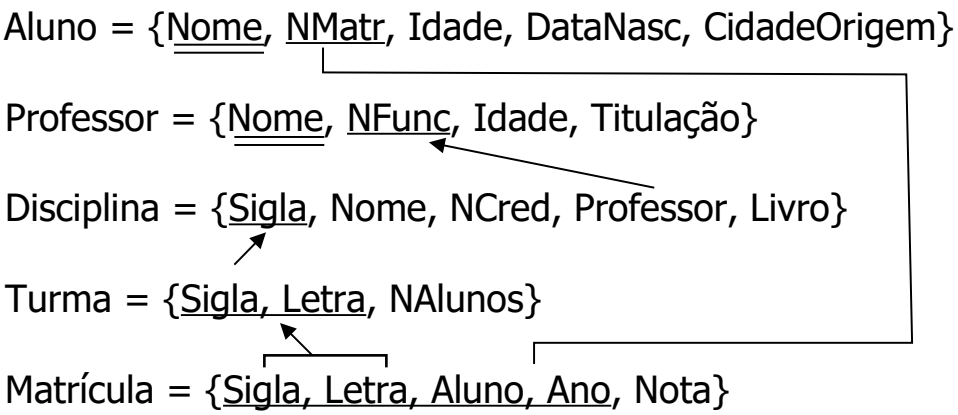



- Caso se deseje manter a tabela turma atualizada para remoções, inserções e atualizações, teríamos:

```
CREATE OR REPLACE FUNCTION update_turma() RETURNS trigger AS $update_turma$
BEGIN
    IF (TG_OP = 'DELETE') THEN
            UPDATE Turma SET NAlunos = NAlunos - 1 WHERE Sigla = old.Sigla AND
            Letra = old.Letra;
    ELSIF (TG_OP = 'INSERT') THEN
            UPDATE Turma SET NAlunos = NAlunos + 1 WHERE Sigla = new.Sigla AND
            Letra = new.Letra;
    ELSIF (TG_OP = 'UPDATE') THEN
            IF((NEW.SIGLA <> OLD.SIGLA) || (NEW.LETRA <> OLD.LETRA)) THEN
                    UPDATE Turma SET NAlunos = NAlunos - 1 WHERE Sigla = old.Sigla
                    AND Letra = old.Letra;
                    UPDATE Turma SET NAlunos = NAlunos + 1 WHERE Sigla = new.Sigla
                    AND Letra = new.Letra;
            END IF;
    END IF;
    RETURN NULL;
END;
$update_turma$ LANGUAGE plpgsql;

DROP TRIGGER NroDeAlunos ON Matricula;

CREATE TRIGGER NroDeAlunos
AFTER DELETE OR UPDATE OR INSERT ON Matricula
FOR EACH ROW EXECUTE PROCEDURE update_turma();
```

# Triggers em PostgreSQL

- Definição de regra de negócio
- Imagine que o esquema de exemplo de aula




fosse modificado para que professores pudessem se tornar alunos e se matricular.

# Triggers em PostgreSQL

- Definição de regra de negócio
- Imag ula

Há uma regra de negócio implícita neste caso:

*um professor não pode se matricular em disciplinas que ele mesmo ministra

fosse modificado para que professores pudessem se tornar alunos e se matricular.

# Triggers em PostgreSQL

```
CREATE OR REPLACE FUNCTION check_professor() RETURNS trigger AS $check_professor$
DECLARE
    discs_prof INTEGER;
BEGIN
    SELECT COUNT(*) INTO discs_prof
    FROM DISCIPLINA
    WHERE Professor = NEW.Aluno AND
              Sigla = NEW.Sigla;

    IF discs_prof > 0 THEN
              RAISE EXCEPTION 'Um professor não pode se matricular em disciplinas que ele mesmo
              ministra';
    END IF;
    RETURN NEW;          -- retorna a tupla para prosseguir com a operação
END;
$check_professor$ LANGUAGE plpgsql;

DROP TRIGGER check_matricula_de_professor ON Matricula;

CREATE TRIGGER check_matricula_de_professor
BEFORE INSERT ON Matricula
FOR EACH ROW EXECUTE PROCEDURE check_professor();
```

# Triggers em PostgreSQL

INSERT INTO ALUNO VALUES(20, 'Prof', 35, '05/05/1976','Lins'); ➔ o professor 20 se torna aluno

E tenta se matricular em sua própria disciplina

INSERT INTO MATRICULA VALUES('FGA0137', A, 20, 2021, 0);

```
ERRO:   Um professor não pode se matricular em disciplinas que ele mesmo
ministra


********** Error **********

ERRO: Um professor não pode se matricular em disciplinas que ele mesmo ministra
SQL state: P0001
```

```
BEFORE INSERT ON Matricula
FOR EACH ROW EXECUTE PROCEDURE check_professor();
```

Exercício 1: implemente uma regra de negócio que impede que professores tenha sua titulação atualizada para um status anterior da ordem <Bacharel, Mestre, Doutor, Livre-docente, Titular, Catedrático>

Exercício 2: implemente uma regra de negócio que impede que um aluno se matricule em mais do que 20 créditos por semestre

Exercício 3: acrescente um atributo nota_media na tabela disciplina e escreva um gatilho que mantém o valor deste atributo atualizado de acordo com as atualizações da tabela matricula

Exercício 4: usando trigger, implemente o fato de que o atributo Aluno.Idade é derivado de Aluno.DataNasc

```
CR
DE

BE

EN
$c

DR

CR
BEFORE INSERT ON Matricula
FOR EACH ROW EXECUTE PROCEDURE check_professor();
```

# Triggers em PostgreSQL

- Auditoria
  - Exemplo:
  - Uma informação particularmente importante é a nota que um aluno recebe quando faz uma disciplina
  - É interessante ter-se um controle de todas as vezes que informações importantes são alteradas, como e quando foram alteradas

# Triggers em PostgreSQL

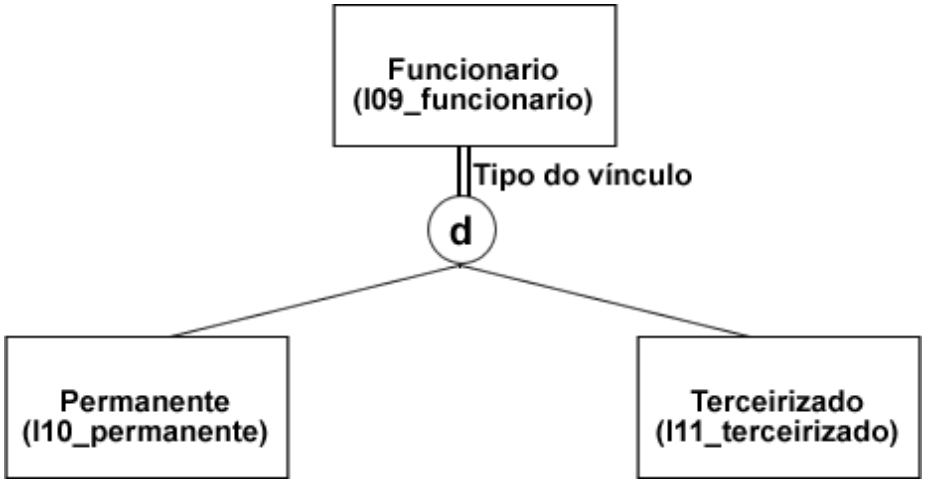
```
CREATE TABLE AUDIT_NOTA(
 id_audit SERIAL PRIMARY KEY,

 Sigla CHAR(7),
 Letra CHAR(1),
 Aluno INTEGER,
 Ano INTEGER,

 Nota_anterior DECIMAL(3,1),
 Nota_nova DECIMAL(3,1),

 Data DATE,
 Usuario VARCHAR(300)
 --pense nas constraints necessárias....
);
```

# Triggers em PostgreSQL

```
DROP TRIGGER NroDeAlunos ON Matricula;
DROP TRIGGER check_matricula_de_professor ON Matricula;
CREATE OR REPLACE FUNCTION audit_nota() RETURNS trigger AS $audit_nota$
BEGIN
    IF OLD.NOTA <> NEW.NOTA THEN

            INSERT INTO AUDIT_NOTA
            VALUES(DEFAULT, NEW.Sigla, NEW.Letra, NEW.Aluno, NEW.Ano,
        OLD.Nota, NEW.Nota, now(), USER );

    END IF;
    RETURN NULL;
END;
$audit_nota$ LANGUAGE plpgsql;

DROP TRIGGER auditoria_de_nota ON Matricula;
CREATE TRIGGER auditoria_de_nota
AFTER UPDATE ON Matricula
FOR EACH ROW EXECUTE PROCEDURE audit_nota();
```

# Triggers em PostgreSQL

- Especialização/generalização
  - Usando apenas o relacional, nem sempre é possível manter as regras de generalização/especialização
  - Exemplo:
    - Um funcionário não pode ser ao mesmo tempo permanente e terceirizado – pois definiu-se a especialização como disjunta



# Triggers em PostgreSQL

```
(select fucpf
from funcionario join permanente on fucpf = pecpf)

intersect

(select fucpf
from funcionario join terceirizado on fucpf = tcpf)
```

➔ **Deve retornar conjunto vazio;**

➔ **No entanto, sem trigger, nada impede que um funcionário seja inserido em ambas as tabelas.**

# Triggers em PostgreSQL

```
CREATE OR REPLACE FUNCTION check_permanente_fc() RETURNS trigger AS
   $check_permanente_fc$
BEGIN
   PERFORM * FROM permanente WHERE PECPF = NEW.TCPF;
   IF FOUND THEN
          RAISE EXCEPTION 'Este funcionário já se encontra na tabela de
   permanentes';
   END IF;
   RETURN NEW;       -- retorna a tupla para prosseguir com a operação
END;
$check_permanente_fc$ LANGUAGE plpgsql;

DROP TRIGGER check_permanente ON terceirizado;

CREATE TRIGGER check_permanente
BEFORE UPDATE OR INSERT ON terceirizado
FOR EACH ROW EXECUTE PROCEDURE check_permanente_fc();
```

# Triggers em PostgreSQL

Exercício 5: escrever o código que garante que uma tupla que esteja em terceirizado não possa ser inserida em permanente.

```
CREAT                                                    er   AS
    $c
BEGIN
    PE
    IF
                                                              de
    pe
    EN
    RE                                                        )
END;
$chec

DROP

CREATE TRIGGER check_permanente
BEFORE UPDATE OR INSERT ON terceirizado
FOR EACH ROW EXECUTE PROCEDURE check_permanente_fc();
```

# Triggers de DML

- **Tabela desencadeadora**
- **Instrução de disparo**
  - INSERT
  - UPDATE
  - DELETE
- ***Timing***
  - BEFORE
  - AFTER
- ***Nível***
  - Row (for each row)
  - Statement

# Triggers de DML

- **Tabela desencadeadora**
- **Instrução de disparo**
  - INSERT
  - UPDATE
  - DELETE
- ***Timing***
  - BEFORE
  - AFTER
- ***Nível***
  - Row (for each row)
  - **Statement ←**

# Triggers em PostgreSQL

**Exemplo, nível de statement:**

```
CREATE OR REPLACE FUNCTION st_alteracoes_aluno() RETURNS trigger AS
   $st_alteracoes_aluno$
BEGIN
   RAISE NOTICE 'Statement - Tentou-se remover dados da tabela Aluno';

   RETURN NULL;
END;
$st_alteracoes_aluno$ LANGUAGE plpgsql;

DROP TRIGGER st_alteracoes_aluno_aviso ON ALUNO;

CREATE TRIGGER st_alteracoes_aluno_aviso
AFTER DELETE ON ALUNO
EXECUTE PROCEDURE st_alteracoes_aluno();
```

# Triggers em PostgreSQL

Exercício 6: escreva dois gatilhos de nível de statement que avisa ao usuário qual o número de tuplas existentes na tabela matricula antes e após operações de insert, update, ou delete.

```
Exemp

CREAT                                                    AS
    $s
BEGIN
    RA

    RE
END;
$st_a

DROP

CREAT
AFTER DELETE ON ALUNO
EXECUTE PROCEDURE st_alteracoes_aluno();
```

# Triggers em PostgreSQL

**Exemplo, nível de rows:**

```
CREATE OR REPLACE FUNCTION rw_alteracoes_aluno() RETURNS trigger AS
   $rw_alteracoes_aluno$
BEGIN
   RAISE NOTICE 'Rows - Dados foram removidos da tabela Aluno';

   RETURN NULL;
END;
$rw_alteracoes_aluno$ LANGUAGE plpgsql;

DROP TRIGGER rw_alteracoes_aluno_aviso;

CREATE TRIGGER rw_alteracoes_aluno_aviso
AFTER DELETE ON ALUNO
FOR EACH ROW EXECUTE PROCEDURE rw_alteracoes_aluno();
```

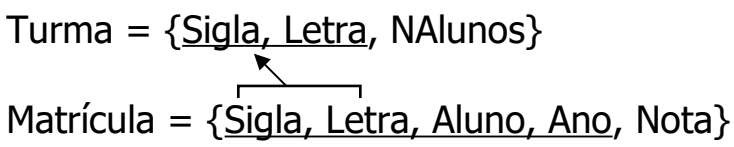
# Triggers em PostgreSQL

Outro exemplo, nível de rows:

```
CREATE OR REPLACE FUNCTION st_matricula() RETURNS trigger AS $st_matricula$
BEGIN
    RAISE NOTICE 'Não é permitido remover dados da tabela Matricula';
   /*Como o procedimento é chamado em BEFORE, pode-se evitar que os dados sejam removidos*/

    /*Impede que os dados sejam removidos*/
     RETURN NULL;
    /*Equivale ao RETURN NULL, pois new em uma operação de delete é NULL*/
    /*RETURN NEW*/;
    /*Efetiva a remoção dos dados, retornando a tupla a ser deletada*/
    /*RETURN OLD*/;
END;
$st_matricula$ LANGUAGE plpgsql;

--DROP TRIGGER st_matricula_aviso ON MATRICULA;

CREATE TRIGGER st_matricula_aviso
BEFORE DELETE ON MATRICULA
FOR EACH ROW EXECUTE PROCEDURE st_matricula();
```

# Triggers em PostgreSQL

```
CREATE OR REPLACE FUNCTION st_matricula_testando() RETURNS trigger AS $st_matricula_testando$
BEGIN
    /*Caso o atributo letra tenha valor C, interrompe a inserção/atualização com o RETURN NULL*/
    IF(NEW.LETRA = 'C') THEN
              RAISE NOTICE 'Não';
              RETURN NULL;
    END IF;
    /*Caso continue, corrige o atributo sigla com a função UPPER*/
    NEW.sigla := UPPER(NEW.sigla);
    /*O return new concretiza a operação*/
    RETURN NEW;
END;
$st_matricula_testando$ LANGUAGE plpgsql;

--DROP TRIGGER st_matricula_teste ON MATRICULA;

CREATE TRIGGER st_matricula_teste
BEFORE INSERT OR UPDATE ON MATRICULA
FOR EACH ROW EXECUTE PROCEDURE st_matricula_testando();
```



# Triggers em PostgreSQL

- Trigger after: não é possível escrever nem :old e nem em :new
- Trigger before: não é possível escrever em :old, apenas em :new

```
CREATE
BEGIN
    /*C                                                                    L*/
    IF(


    END
    /*C
    NEW
    /*C
    RET
END;
$st_mat

--DROP                                                                    a}

CREATE
BEFORE
FOR EACH ROW EXECUTE PROCEDURE st_matricula_testando();
```

# 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**
  - .....

# Funções em PostgreSQL

- Consolidação de dados
  - Às vezes os dados podem se encontrar em estados inconsistentes, o que irá requerer intervenção do DBA para corrigir os dados
  - Para tanto, podem-se usar funções não vinculadas a triggers
  - Exemplo: número de alunos da tabela de turmas

```
alter table turma drop constraint Turma_ck;

CREATE OR REPLACE FUNCTION corrige_n_alunos() RETURNS void AS $corrige_n_alunos$
DECLARE
  cursor_turma CURSOR FOR SELECT * FROM TURMA;
  turma_row turma%ROWTYPE;
  total_alunos INTEGER;
BEGIN
   OPEN cursor_turma;
   LOOP
            FETCH cursor_turma INTO turma_row;
            EXIT WHEN NOT FOUND;

            SELECT COUNT(*) INTO total_alunos
            FROM MATRICULA
            WHERE SIGLA = turma_row.SIGLA AND LETRA = turma_row.LETRA;

            UPDATE turma SET nalunos = total_alunos
            WHERE SIGLA = turma_row.SIGLA AND LETRA = turma_row.LETRA;
   END LOOP;
   CLOSE cursor_turma;
END;
$corrige_n_alunos$ LANGUAGE plpgsql;

SELECT corrige_n_alunos()
```

```
alter table turma drop constraint Turma_ck;
```

Exercício 7: acrescente um atributo booleano na tabela matricula denominado “aprovacao” com valor default “false”. Escreva um procedimento que para cada tupla de matricula atualize o valor do atributo aprovação, escrevendo “true” caso a nota seja maior ou igual a 5 ou “false” caso contrário.

Exercício 8: escreva um gatilho que usa a função do exercício 7, mantendo atualizado o atributo criado na tabela matricula toda vez que uma tupla for inserida ou atualizada

Exercício 9: escreva uma função que atualiza o atributo Idade da tabela Aluno

Exercício 10: escreva uma função que escreve (RAISE NOTICE) os dados de cada professor referente a total de disciplinas, total de créditos, e total de alunos.