PCS3413

Engenharia de Software e Banco de Dados

Aula 20

Escola Politécnica da Universidade de São Paulo

PROCEDIMENTOS – TRIGGER STORED PROCEDURE

Regras de Negócio controladas pelo SGBD

- SGDB Relacional dispoõe de recursos que podem ser utilizados para controle de regras da aplicação:
  - regras declarativas
    - verificações: checks
    - integridade referêncial – chave estrangeira
    - regras de domínio, obrigatoriedade de atributos
  - regras procedimentais
    - stored procedure
    - trigger

Integridade Referencial

```
create table depto
(cod_depto numeric(4),
nome_depto varchar(25),
Primary Key (cod_depto));

create table emp (
nemp numeric(6),
cod_depto numeric(4),
Primary Key (nemp),
Foreign Key (cod_depto) references depto on delete set null);
```

Estado das relações depto e emp

cod_depto	nome_depto
D1	Marketing
D2	Qualidade

nemp	cod_depto
1	D2
2	D2
3	D1

delete from depto
where cod_depto = D1;

set null

nemp	cod_depto
1	D2
2	D2
3	

```
create table emp (
nemp numeric(6),
cod_depto numeric(4),
Primary Key (nemp),
Foreign Key (cod_depto) references depto on delete cascade);
```

cod_depto	nome_depto
D1	Marketing
D2	Qualidade

nemp	cod_depto
1	D2
2	D2
3	D1

```
delete from depto
where cod_depto = D1;
```

nemp	cod_depto
1	D2
2	D2

Conclusão

- Dependendo da forma como a restrição de chave estrangeira for declarada, uma remoção na tabela referenciada poderá:

1. não ser executada, ou
2. implicar em atualizações na(s) tabela(s) que mantém a referência, ou
3. implicar em remoções na(s) tabela(s) que mantém a referência.

Investigue o que acontece se a declaração fosse:

```
create table emp (
nemp numeric(6),
cod_depto numeric(4),
Primary Key (nemp),
Foreign Key (cod_depto) references depto on update cascade);
```

on update set null

REGRAS PROCEDIMENTAIS

Regras Procedimentais

- Trigger (gatilho)
  - procedimentos disparados a partir da ocorrência de um evento e que provocará a modificação de uma tabela (insert, delete, update) do BD.

armazenados no banco de dados

documentados no dicionário de dados

- Stored Procedure
  - procedimentos armazenados no banco de dados que armazenam regras associadas ao negócio da aplicação.

Stored Procedure - continuação

sintaxe – ambiente Oracle

create [**or replace**] **procedure** *procedure_name*
(argumento modo tipo_de_dado, **...**)
is
begin
....
end *procedure_name*;

argumento: variável passada para a procedure
modo: tipo do argumento
(in, out)

- quando necessário executar a procedure:

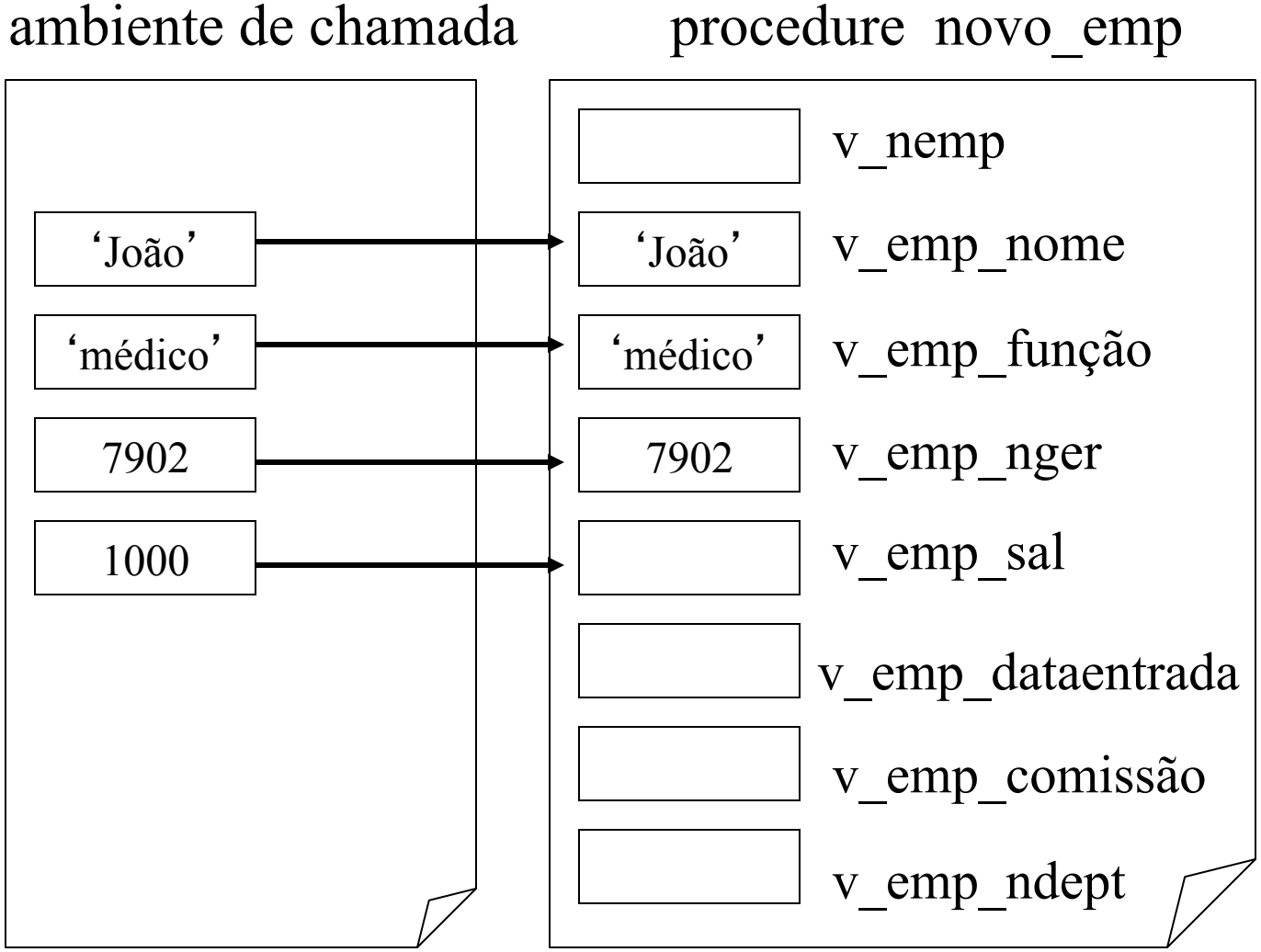
execute *procedure_name*;

Stored Procedure - exemplo

- introduz somente informação necessária para um novo empregado
  - Elimina entrada desnecessária da rotina de manipulação de dados

emp (nemp, nome, função, nger, sal, dataentrada, comissão, ndept)

exemplo


ambiente de chamada
procedure novo_emp
'João'
'médico'
7902
1000
v_nemp
'João'
v_emp_nome
'médico'
v_emp_função
7902
v_emp_nger
v_emp_sal
v_emp_dataentrada
v_emp_comissão
v_emp_ndept

continuação

```
CREATE OR REPLACE PROCEDURE  novo_emp
(v_emp_nome  IN   emp.nome%type,
v_emp_função  IN   emp.função%type,
v_gerente     IN emp.nger%type,
v_emp_sal     IN emp.sal%type)
IS
     v_emp_dataentrada       emp.dataentrada%type;
     v_emp_comissão          emp.comissão%type;
     v_ndept                 emp.ndept%type;
Begin
```

continuação

```
Begin
     v_emp_dataentrada := SYSDATE;
     IF  v_emp_função =  'VENDEDOR'
      THEN v_emp_comissão:= 0
      ELSE       v_emp_comissão : = null;
     end IF;
     SELECT  ndept    /*mesmo depto que o gerente */
                INTO v_ndept
     FROM    emp
     WHERE        nemp = v_gerente;
    INSERT  INTO   emp    VALUES (s_nemp.NEXTVAL, v_emp_nome,
    v_emp_função, v_gerente, v_emp_sal,  v_emp_dataentrada,
    v_emp_comissão, v_ndept);
end novo_emp;
```

Stored Procedure - benefícios

- assegura que ações relacionadas sejam todas efetuadas em conjunto, ou parcialmente

segurança / integridade

- armazenar uma única cópia do código no banco, evitando múltiplas cópias do mesmo código nas várias aplicações (reutilização).

memória

- modificação uma única vez de uma rotina, que fica acessível a várias aplicações (reutilização).
- elimina testes duplicados quando uma rotina é modificada.

manutenção

Trigger

- para se projetar um trigger é preciso:

1. especificar as condições sob as quais o trigger será disparado.
2. especificar as ações que serão executadas com o disparo do trigger.

trigger - continuação

- Ex. aplicação bancária

conta (número_conta, nome_ag, saldo)
depositante (número_conta, nome_cli)
empréstimo (número_empr, nome_ag, total)
devedor (número_empr, nome_cli)

- toda vez que uma retirada em conta acarretar em saldo negativo, um empréstimo é aberto com o valor do saldo negativo e o saldo da conta é zerado.
- o empréstimo é aberto com o mesmo número da conta e é associado ao mesmo cliente (devedor).
- a condição do trigger é atualização em conta que resulte em saldo negativo
- as ações que serão efetivadas com o disparo do trigger serão:
  - ➲ inserção de um empréstimo
  - ➲ inserção de um devedor
  - ➲ atualização de saldo = 0 em conta.

```
define trigger saldo_negativo
    after update of conta
   (if new saldo < 0 then
    insert into empréstimo values
       (conta.número_conta, conta.nome_ag,
        - new conta.saldo);
    insert into devedor values
       (select número_conta, nome_cli
        from  depositante
        where     número_conta =
              conta.número_conta);
    update conta
    set  saldo = 0
    where  número_conta = empréstimo. número_empr);
```

before
after

update
delete
insert

tabelas

- inserção de um empréstimo
- inserção de um devedor
- atualização de saldo = 0 em conta.

Triggers no DB2

- Sintaxe:
  create trigger <nome_trigger>
  <ação> on <nome_tabela>
  <operação>
  <ação_triggered>

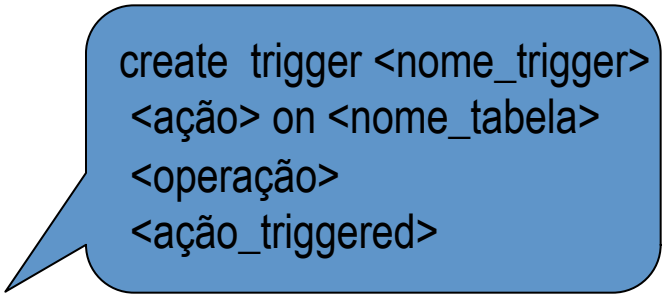
Usado para:

- Validar dados de entrada;
- Gerar um valor para uma nova linha inserida;
- Ler de outra tabela para fazer referência cruzada;
- Escrever em outra tabela para controle de auditoria.

Triggers no DB2 - exemplo

- Exemplo:
  - Criar um trigger que aumenta o id (EMPID) do empregado cada vez que uma nova pessoa é incluída na tabela EMPLOYEES.
  - EMPLOYEES (EMPID, NAME, DEPT)
  - Trigger:

```
create trigger new_emp
after insert on employees
for each row
update employees set empid = empid + 1
```



Triggers no DB2 – exemplo – continuação

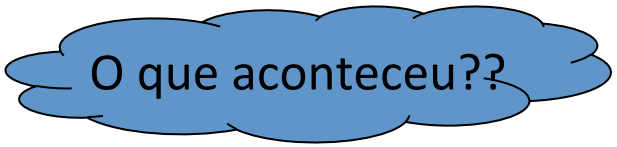
- Estado da tabela EMPLOYEES antes da execução da trigger:

EMPID	NAME	DEPT
1	Adam	A01
2	John	B01
3	Pedro	A01
4	Daniel	B01
5	Stan	B01
9	Bill	B01

insert into EMPLOYYES values
(10, 'Steve', 'B01')

- Estado da tabela EMPLOYEES depois da execução da trigger:

EMPID	NAME	DEPT
2	Adam	A01
3	John	B01
4	Pedro	A01
5	Daniel	B01
6	Stan	B01
10	Bill	B01
11	Steve	B01



Triggers no DB2 – exemplo – continuação

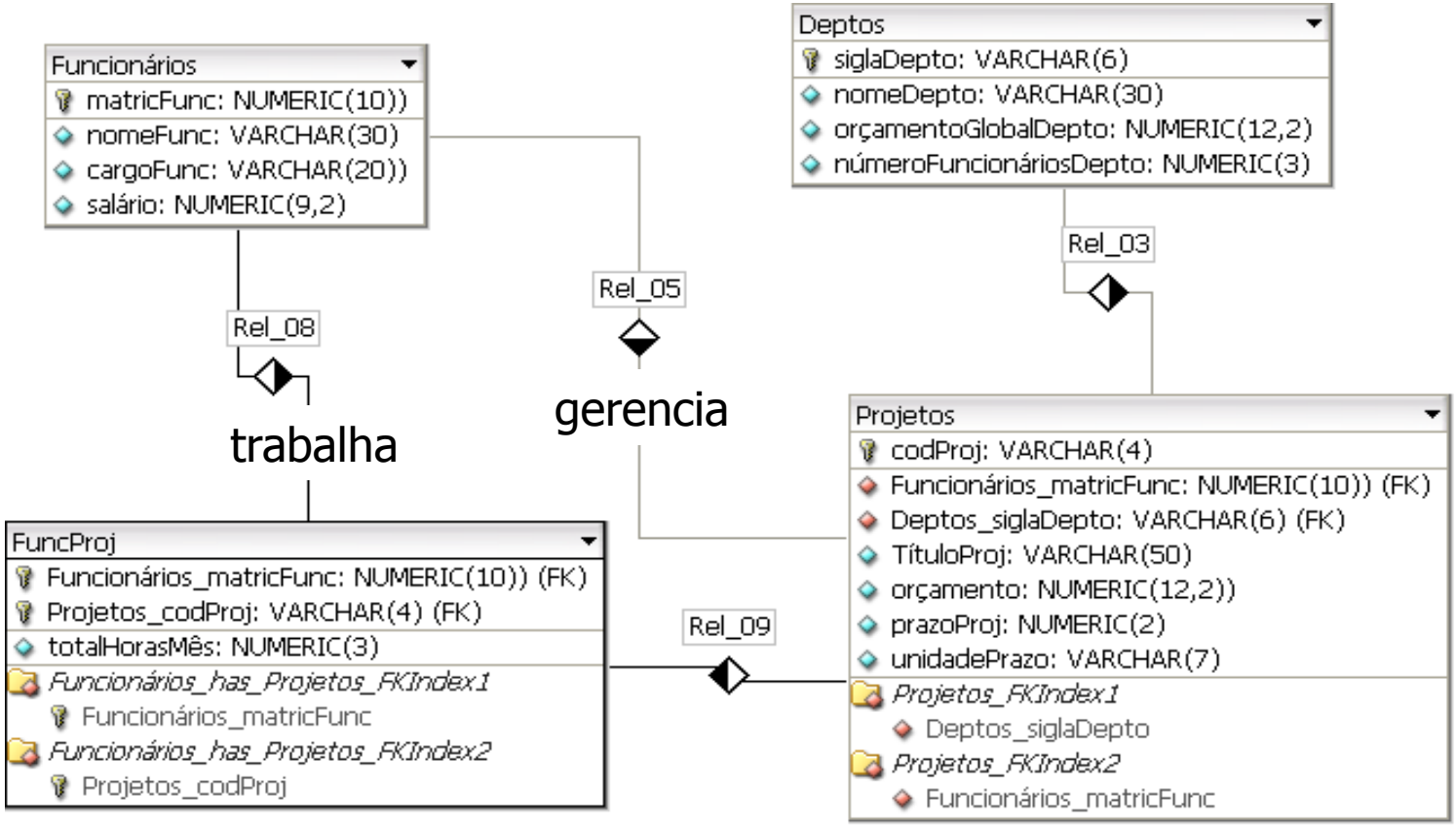
- Os EMPID de todos os empregados foram aumentados de 1, incluindo a linha que foi incluída!

- AFTER trigger: após a inclusão da linha (operação INSERT), a ação definida no trigger é executada de acordo com a operação (“for each row”) que foi definida no trigger.

- Verifique também o que acontece para os eventos: DELETE E UPDATE! E para os tempos de ativação: BEFORE, AFTER, INSTEAD OF!

Triggers no PostgreSql (8.0)

Modelo Relacional para o esquemaFuncProj



Representação lógica das tabelas para o esquema FuncProj

Funcionarios (**matricFunc**, nomeFunc, cargoFunc, salarioFunc);

Deptos (**siglaDepto**, nomeDepto, orcamentoGlobalDepto, numeroFuncionariosDepto)

Projetos (**codProj**, tituloProj, orcamento, prazoProj, unidadePrazo, CoordProj, siglaDepto)

FuncProj (**codProj, matricFunc**, totalHorasMes)

Criação da Trigger

- Regra:
  - não permite a inclusão ou a modificação de uma linha na tabela funcionários que apresente nome nulo e nem salário nulo ou menor que zero.

Função de Trigger

- no PostGreSQL um trigger sempre está associado a uma função.
- A função de trigger apresenta retorno do tipo trigger:

```
create or replace function VerificaEmp () returns trigger as $VerificaSalEmp$
```

```
-- cria a função de trigger
 create or replace function VerificaEmp () returns trigger as $VerificaSalEmp$
begin
     if new.nomeFunc is null or new. nomeFunc = ‘  ’
       then raise exception   'NomeFunc não pode ser nulo';
     end if;
     if new.salarioFunc is null or new.salarioFunc =< 0
       then raise exception      'Salario não pode ser nulo e deve ser maior que zero';
     end if;
     return new;
end;
$VerificaSalEmp$ language plpgsql;

-- cria a trigger
create trigger VerificaSalEmp before insert or update on funcionarios for each row
execute procedure VerificaEmp();
```

Variável NEW contém a nova linha criada por um insert ou update.

OLD: variável que contém os valores antigos de uma linha apagada ou atualizada por um delete ou update

Sintaxe de criação de triggers no PostgreSQL

```
CREATE TRIGGER nomeDoTrigger { BEFORE | AFTER }
    { event [ OR ... ] }
    ON nomeDaTabela [ FOR [ EACH ] { ROW | STATEMENT } ]
EXECUTE PROCEDURE nomeDaFunção ( argumentos )
```

NomeDoTrigger: define o nome do trigger

Before ou After: define se o trigger será executado antes ou depois da ação que o disparou (tempo de ativação do trigger)

Evento: indica qual ação (insert, delete ou update) dispara o trigger. O mesmo trigger pode ser disparado por diferentes ações, exemplo: insert or update

nomeDaTabela: indica a tabela a qual o trigger está associado.

For each row: indica que o trigger será executado para cada linha alterada pela ação que o disparou (row-level)

For each statement: indica que o trigger será executado uma única vez para a ação que o disparou.

nomeDaFunção: função que será executada quando o trigger for disparado.

Argumentos: argumentos a serem passados para a função de trigger.