Restrições de integridade

Parte II

Banco de Dados I

Carina F. Dorneles
dorneles@inf.ufsc.br

ine
CTC • UFSC
INFORMÁTICA E ESTATÍSTICA

Restrições de Integridade

- Mantém a consistência da Base de Dados

- Restrições
  - De domínio, de chave primária, de valores NULL,de integridade, de valores

  - Regras de negócio
    - São restrições aplicadas aos valores dos atributos e que normalmente estão relacionados ao domínio de problema

    - Podem ser escritas através de
      - *Stored Procedure*
      - *Functions*
      - *Triggers*

Regras de Negócio

- **SQL Pura**
  - Não suporta execução de procedimentos armazenados em condição lógica (uso de IF THEN ELSE)
  - Falha no suporte a operações de laço (WHILE, FOR)

Regras de Negócio

- Duas Soluções

1. Comandos SQL podem ser chamados de dentro de uma linguagem de programação

   - Lógica fica na linguagem de programação
   - Aplicativo cliente
     - Envia o comando para o servidor de BD e aguarda o comando ser processado,
     - Recebe os resultados
     - Realizar algum processamento
     - Dependendo do resultado, pode enviar outras consultas para o servidor novamente
     - Envolve comunicação entre processos e uso de rede se o cliente e o servidor estiverem em máquinas diferentes

2. SQL procedural armazenado dentro do banco de dados

   - Executado pelo SGBD, e apenas invocado pelo programa
   - Todo o processamento é feito no servidor.
     - *Stored procedures* (procedimentos)
     - *Functions* (funções)
     - *Triggers* (gatilhos)

SQL Procedural

- SQL Procedural e os SGBDs
  - Diferentes SGBDs oferecem diferentes características em suas linguagens SQL procedurais

- Denominada de diferentes formas

  - Oracle: PL/SQL
  - PostgreSQL: PL/pgSQL
  - SQL Server: Transact-SQL
  - Interbase/Firebird: DSQL e isql

Formato geral da SQL Procedural

- Composta de blocos
- Utilização de variáveis e tipos
- Uso de cursores para retorno de mais de uma linha
- Uso de funções específicas para manipulação de dados
- Uso de condições
  - IF THEN ELSE
- Uso de laços
  - WHILE, FOR

Algumas funções úteis no SQL procedural

Funções

- Manipulação de cadeias de caracteres e números no SQL procedural
- Diferentes SGBDs oferecem diferentes funções
- Podem ser usadas dentro de instruções SQL* ou SQL procedural*.

**nem todos os SGBDs possuem. Verificar antes de usar.*

Funções

- Manipulação de cadeias de caracteres e números no SQL procedural
- Diferentes SGBDs oferecem diferentes função
- Podem ser usadas dentro de instruções SQL* ou SQL procedural*.
  - Exemplos:

```
SELECT UPPER(nome)
FROM pessoa;
```

Nome
JULIANA PAULA
PAULINHA VERA
CLAUDIA REGINA
CLAUDIO ROGÉRIO
FLAVIO AUGUSTO

```
SELECT nome
FROM pessoa
WHERE UPPER(nome) =
      UPPER('juliana paula')
```

Nome
Juliana Paula

**nem todos os SGBDs possuem. Verificar antes de usar.*

Outros exemplos de Funções

Função	***Descrição***	***Exemplo***
substr(string,pos,len)	Retorna *len* caracteres, começando em *pos*	**nome** = ' Mariana' SUBSTR(nome, 5, 3) = 'ana'
lenght(string)	Retorna o tamanho de string em caracteres.	**nome** = 'Mariana' LENGTH (nome) = 8
lpad(string, length, pad)	Retorna a *string* preenchida com *pad* à esquerda, até completar tamanho *lenght*	**tipo** = sedan LPAD (tipo, 7, '*') = **sedan
rpad(string, length, pad)	Retorna a *string* preenchida com *pad* à direita, até completar tamanho *lenght*	**tipo** = sedan LPAD (tipo, 7, '*') = sedan**
ltrim(string,trimlist)	Retorna a string sem os caracteres trimlist à esquerda	**nome** = 'yyxxMariana' LTRIM (nome, 'xy') = Mariana
rtrim(string,trimlist)	Retorna a string sem os caracteres trimlist à direita	**nome** = 'Marianaxxyy' RTRIM (nome, 'xy') = Mariana
replace (string,target,replacement)	Substitui na string, target por replacement	**nome**= 'JACK and JUE' REPLACE(nome,'J','BL') = 'BLACK and BLUE'
translate(string,fromlist,tolist)	Substitui na string, fromlist por tolist	**nome**= 'JACK and JUE' TRANSLATE(nome,'J','BL') = 'BLACK and BLUE'

SQL procedural armazenado dentro do banco de dados

Stored procedures (procedimentos)
Functions (funções)
Triggers (gatilhos)

Procedimentos
e Funções

Procedimento/Funções

- É um programa que é armazenado dentro do banco de dados
  - É compilado uma vez
- São referenciados pelos aplicativos
- Podem receber parâmetros de entrada e retornar resultados
- Podem ser chamados a partir de
  - Programas escritos em linguagens padrão como Java, C#, Delphi
  - Linguagens de scripts, *JavaScript*, *VBScript*
  - Comandos SQL, *SQL Plus*, *Query Analyzer*

ATENÇÃO!!!!!!!

- Interbase/Firebird
  - CREATE PROCEDURE
- PostgreSQL
  - CREATE FUNCTION
- MySQL
  - CREATE PROCEDURE
  - CREATE FUNCTION
- Oracle
  - CREATE PROCEDURE
  - CREATE FUNCTION
- SQL Server
  - CREATE PROCEDURE
  - CREATE FUNCTION

Diferenças (**Teoricamente!!!!!!!!**)

- Procedimento
  - Não retorna valores
  - Não pode ser chamado nas cláusulas `SELECT/WHERE/HAVING` do SQL
- Funções
  - Retorna valores
  - Pode ser chamada nas cláusulas `SELECT/WHERE/HAVING` do SQL
- Sem o uso de cursores, conseguem tratar apenas uma linha de retorno

Diferenças (**Teoricamente!!!!!!!!**)

- **As diferenças são dependentes do SGBD.**

  Por exemplo:

  - Interbase
    - Procedimentos podem retornar um valor, mas não podem ser executados cláusulas `SELECT/WHERE/HAVING`
  - PostgreSQL
    - Funções e procedimentos são considerados a mesma coisa
    - As funções sempre retornam alguma coisa, mesmo que seja "nada"

Procedimento/Funções

no PostgreSQL

Procedimento/Função

- Possui um nome
- Deve ser chamado pelo nome
- Criado como uma unidade

```
CREATE [OR REPLACE] FUNCTION nome(parametros...)
  RETURNS tipo_retorno
AS $$
DECLARE variáveis...
BEGIN
 corpo
END;
$$ LANGUAGE 'plpgsql';
```

Tipos de variáveis

- Exemplos:

- `codigo` **`integer;`**
- `quantidade` **`numeric(5);`**
- `url` **`varchar;`**
- `linha cliente`**`%ROWTYPE;`**
- `campo cliente.nome`**`%TYPE;`**
- `registro` **`RECORD;`**

Procedimento/Função
sem SQL embutida **com** parâmetro de entrada

- Exemplo I (sintaxe Postgresql):

```
CREATE OR REPLACE FUNCTION media(num1 int, num int)
RETURNS numeric
AS $$
DECLARE total numeric;
    BEGIN
        total = (num1+num)/2;
        RETURN total; /* poderia retornar (num1+num)/2 */
    END;
$$ LANGUAGE plpgsql;
```

Exemplo de chamada pelo pdAdmin III:

```
SELECT media(2,4)
```

Procedimento/Função **com SQL** embutida **sem** parâmetro de entrada

- Exemplo 2 (sintaxe Postgresql):

```
CREATE OR REPLACE FUNCTION nomeMedia()
RETURNS TABLE (nome text, total numeric)
AS $$
      SELECT nome, avg(preco)
      FROM produto
      GROUP BY nome
$$ LANGUAGE SQL;
```

Exemplo de chamada:

```
SELECT media(2,4)
```

Procedimento/Função **com SQL** embutida **com** parâmetro de entrada

- Exemplo 3 (sintaxe Postgresql):

```
CREATE OR REPLACE FUNCTION salariosUF(uf char(2))
RETURNS
   TABLE (soma numeric, med numeric, mini numeric, maxi numeric)
AS $$

   SELECT sum(salario), avg(salario), min(salario), max(salario)
   FROM funcionario f join cidade c on c.codigo =f.codcid
   WHERE c.uf = $1

$$ LANGUAGE SQL;
```

Exemplo de chamada:

```
SELECT salariosUF('RS
```

Procedimento/Função **com PLpgSQL**, **com** par. de entrada e **IF.THEN.ELSE**

- Exemplo 4 (sintaxe Postgresql):

```
CREATE OR REPLACE FUNCTION alteraCat(cod integer)
RETURNS BOOLEAN
AS $$
DECLARE idad integer;
   BEGIN
      select idade into idad
      from pessoa
      where codigo = cod;
      if (idad >=18) then
         update contato
         set categoria = 'adulto'
         where codigo = cod;
      else
         update contato
         set categoria = 'adolescente'
         where codigo = cod;
      end if;
   return 1;
   END;
$$ LANGUAGE plpgsql;
```

Exemplo de chamada:

```
SELECT alteraCat(1)
```

Triggers

Triggers (gatilho)

- É um recurso de programação, desenvolvido dentro do SGBD

- Associadas a ***uma*** tabela do banco de dados
  - Uma tabela pode possuir múltiplos *triggers*

- Podem alterar valores, inserir linhas e chamar procedimentos

Aplicações de *triggers*

- Checar a validade de um dado
- Manter a consistência dos dados
- Propagar alterações de um dado de uma tabela para outras
- Não faz parte do padrão SQL
- Cada SGBD implementa sua própria sintaxe

Triggers

- Código SQL procedural chamado ***antes*** ou ***depois*** do dado de uma linha ser
  - **deletado**, **inserido** ou **alterado**
- Tipos de triggers:

  BEFORE, AFTER
  - Cada um pode ser declarado para INSERT, UPDATE e DELETE

Trigger (gatilho)

- Possui um nome
- Deve ser chamado pelo nome
- Criado como uma unidade

Sintaxe Geral no PostgreSQL:

```
CREATE TRIGGER nome
{ BEFORE | AFTER } { INSERT [OR UPDATE OR DELETE] }
ON tabela
FOR EACH ROW
EXECUTE PROCEDURE funcNome (argumentos)
```

No Postgresql, toda *trigger* deve chamar uma função

Exemplo 1

```
CREATE TRIGGER consAgenda BEFORE INSERT ON consulta
    FOR EACH ROW EXECUTE PROCEDURE consAgenda();
```

Exemplo 1

```
CREATE FUNCTION consAgenda ()
RETURNS trigger AS $consAgenda$
DECLARE codigo integer;
   BEGIN
      SELECT codPac INTO codigo
      FROM agenda
      WHERE codPac = NEW.codPac and data = NEW.data and hora = NEW.hora;

      if (codigo is null) then
        RAISE EXCEPTION '% paciente nao agendou consulta', NEW.codpac;
      end if;

   return NEW; /*se a trigger for AFTER, RETURN NULL */
   END;
$consAgenda$ LANGUAGE plpgsql;

CREATE TRIGGER consAgenda BEFORE INSERT ON consulta
    FOR EACH ROW EXECUTE PROCEDURE consAgenda();
```

Exemplo 2

```
CREATE OR REPLACE FUNCTION valConvenio ()
RETURNS trigger AS $valConvenio$
   BEGIN
      if (NEW.valor > (OLD.valor*1.10) and NEW.codigo = OLD.codigo) then
        RAISE EXCEPTION '% valor excedeu os 10 por cento', NEW.codigo;
      end if;
   return NEW;
   END;
$valConvenio$ LANGUAGE plpgsql;

CREATE TRIGGER valConvenio BEFORE UPDATE ON convenio
    FOR EACH ROW EXECUTE PROCEDURE valConvenio();
```