Técnicas de Programação em BD

Funções, Procedimentos e Triggers

Fernanda Baião

Leonardo Guerreiro Azevedo

Asterio Tanaka

Programação... em Bancos de Dados??

- Objetivo
  - Acessar uma base de dados a partir de um programa de aplicação
    - E não a partir de interfaces interativas
- Por quê?
  - interfaces interativas são convenientes...
    - Especialmente para comandos DDL/DML ou para consultas ad hoc eventuais
  - ...mas não são suficientes!
    - A maioria das operações sobre bases de dados são feitas através de programas de aplicação
      - Cada vez mais a partir de aplicações web, serviços web, serviços em plataformas de nuvem

Passos gerais na interação entre uma aplicação e um SGBD

- Programa cliente abre uma conexão ao servidor de banco de dados
- Programa cliente submete comandos SQL (consultas e/ou atualizações) ao servidor de banco de dados
- Programa cliente fecha a conexão

Abordagens para programação com o banco de dados

- Comandos embutidos (SQL Embutida)
  - comandos de banco de dados são embutidos em uma linguagem de programação de propósito geral
- API (*Application Programming Interface*)
  - Biblioteca de funções para o banco de dados
  - Disponível na linguagem hospedeira para realizar as chamadas ao SGBD
- Procedimentos e funções armazenados

Abordagem SQL embutida

- Muitos dos comandos SQL podem ser embutidos em uma linguagem de programação de propósito geral, que é chamada linguagem hospedeira
  - C, ADA, COBOL ,PASCAL, Java
- Supre certas deficiências da SQL
  - consultas recursivas ou por similaridade
  - ações não-declarativas (apresentar dados em interfaces gráficas ou relatórios)
  - Implementação de algoritmos complexos
- Sintaxe
  - Palavra chave distingue entre comandos SQL e os demais comandos
    - sintaxe varia com a linguagem

      EXEC SQL
      ...
      END-EXEC

      EXEC SQL BEGIN
      ...
      EXEC SQL END (ou ;)
  - Variáveis compartilhadas entre as 2 linguagens são geralmente prefixadas com “:” em SQL

```
prompt  ("Entre com o Nome do Departamento:   ",
      dnome)   ;
    EXEC SQL
     select DNUMERO into :dnumero
     from DEPARTAMENTO where DNOME = :dnome ;
 EXEC SQL DECLARE EMP CURSOR FOR
     select SSN, PNOME, MINICIAL, UNOME, SALÁRIO
     from EMPREGADO where DNO = :dnumero
     FOR UPDATE 0F SALÁRIO ;
 EXEC SQL OPEN EMP ;
 EXEC SQL FETCH from EMP into :ssn, :pnome,
      :minicial, :unome, :salario ;
while (SQLCODE « 0) {
    printf("Nome do empregado e:", pnome, minit, unome)
    prompt("Entre com o aumento de salário: ", aumento) ;
    EXEC SQL
        update EMPREGADO
        set SALÁRIO = SALÁRIO + :raise
        where CURRENT OF EMP ;
    EXEC SQL FETCH from EMP into :ssn, :pnome,
      :minicial, :unome,
:salario ;
       }
 EXEC SQL CLOSE EMP   ;
```

```
import java.sql.*;
import sqlj.runtime.ref.*;
import sqlj.runtime.ConnectionContext;

public class Pessoa {

  private String nome;
  private Integer idade;
  private Integer id;

  private TesteSQLJContext ctx;

  public void select() {
    #sql [ctx] {
      SELECT ID, NOME, IDADE
         INTO :id, :nome, :idade
         FROM PESSOA
       WHERE ID = :id
    };
  }

  public void insert() {
    #sql [ctx] {
      INSERT INTO PESSOA (ID, NOME, IDADE)
        VALUES(:id, :nome, :idade)
    };
  }

   public void setConnection( Connection conn ) throws SQLException {
      ctx = new TesteSQLJContext( conn );
   }

   public void releaseConnection() {
      try { if(ctx!=null) ctx.close(ConnectionContext.KEEP_CONNECTION); }
      catch(Throwable e) {}
   }
}


//no arquivo TesteSQLJContext.sqlj

import sqlj.runtime.ref.*;
import java.sql.*;
#sql public context TesteSQLJContext;
```

Fonte: http://www.guj.com.br/java/61-usando-sqlj-ao-inves-de-jdbc

Abordagem API

- Abordagem para acesso a bancos de dados através de API
  - SQL/CLI (Call Level Interface)
  - JDBC
- Vantagem
  - A API para programação do sistema é a mesma para qualquer SGBD, não há necessidade de se desenvolver aplicações voltadas para um SGBD específico
    - Independência de plataforma
    - Flexibilidade
- Desvantagem
  - Verificação da sintaxe SQL em tempo de execução

Abordagem API JDBC

- muitos fabricantes de SGBDR fornecem drivers JDBC para que programas JAVA possam acessar seus produtos
  - Oracle, PostgreSQL, MySQL, ...
- driver JDBC
  - implementação das chamadas de função especificadas na API JDBC para o SGBDR de um fabricante em particular
- Um programa em Java pode se conectar a vários bancos de dados diferentes via JDBC
  - Data sources

Abordagem API JDBC - passos

- Importar a biblioteca de classes JDBC
- Carregar o driver JDBC
- Criar as variáveis apropriadas
- Criar conexão
- Preparar e executar comandos SQL, e manipular seus resultados
  - –Statement, PreparedStatement, CallableStatement, ResultSet

Procedimentos Armazenados (stored procedures)

- Módulos de programa armazenados pelo SGBD no servidor de banco de dados
  - diretamente implementados dentro do SBD
  - diretamente executados pelo SGBD
  - Estende a linguagem SQL com alguns construtores de programação (declarações condicionais e laços)
  - linguagem de programação do SGBD (PLSQL, PL/PGSQL, ...)

Procedimentos Armazenados (stored procedures)

- Vantagens
  - Manutenibilidade
    - Se o procedimento é necessário a várias aplicações cliente
      - Implementação única
      - Reutilização
  - Desempenho
    - Redução dos custos de comunicação
    - Acesso direto e de forma otimizada a base de dados
    - Pré-compilados
  - Segurança
    - Tudo que ocorre dentro delas acontece num contexto transacional
    - Usuários podem acessar apenas funções e não as estruturas em si (camada de acesso a dados)
  - Portabilidade
    - Implementação única (para cada SGBD) invocada por aplicações em vários ambientes

Implementação

Criação

```
CREATE PROCEDURE <procedure-name> [(<params>)]

<local-declarations>

<procedure-body;>

CREATE FUNCTION <fun-name> [(<params>)]
RETURNS <return-type>

<local-declarations>

<function-body;>
```

Chamada

```
CALL <procedure-name|fun-name> ([<arguments>]);
```

Codificação

- PL\SQL, PL/PGSQL, SDL, …

PL/pgSQL

- Linguagem procedural para o PostgreSQL
- Objetivos
  - Implementar funções, procedimentos e triggers
  - Adicionar estruturas de controle à linguagem SQL
  - Executar operações complexas
  - Confiável pelo SGBD
  - Fácil de usar
- Comando básico: CREATE [OR REPLACE] FUNCTION

CREATE FUNCTION

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
    | TRANSFORM { FOR TYPE type_name } [, ... ]
    | WINDOW
    | IMMUTABLE | STABLE | VOLATILE | [ NOT ] LEAKPROOF
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
  } ...
    [ WITH ( attribute [, ...] ) ]
```

https://www.postgresql.org/docs/9.5/static/sql-createfunction.html

Execução de código PL/pgSQL

- A chamada PL/pgSQL transforma o código fonte de uma função ou procedimento e produz uma árvore binária de instruções
  - Apenas na primeira vez em que a função é chamada (dentro de cada sessão)
  - A árvore de instruções traduz completamente a estrutura de comandos PL/pgSQL
    - expressões SQL embutidas não são traduzidas imediatamente, só após o primeiro uso da expressão SQL.
  - Chamadas subsequentes à expressão SQL reaproveitam o plano de execução.

OBS: Funções e procedimentos são criados por CREATE FUNCTION.
Qual a diferença entre função e procedimento?

Vantagens ao usar PL/pgSQL

- Cada comando SQL é executado individualmente pelo servidor de banco de dados.
  - Em uma aplicação:
    - Uma consulta é enviada ao servidor de banco de dados;
    - A aplicação fica esperando que a consulta seja processada;
    - Aplicação realização algum processamento, e envia outras consultas ao servidor de banco de dados;
  - Consequências: comunicação interprocesso e pode ter alto custo de uso da rede.

Vantagens ao usar PL/pgSQL

- Usando PL/pgSQL
  - Um bloco de comandos e uma série de consultas podem ser agrupadas em um procedimento.
  - Não é mais necessário o custo de comunicação cliente/servidor.
  - Consequência: Ganho de desempenho.

Estrutura do PL/pgSQL

- PL/pgSQL é uma linguagem estruturada em blocos, ou seja, o texto completo da definição da função deve ser um bloco.

```
-- comentario de linha

/* comentario

de bloco */
[<<label>>]
[ DECLARE
            declarations ]
BEGIN
            statements
END [ label ];
```

Exemplos

```
CREATE OR REPLACE FUNCTION soma(integer, integer)
RETURNS integer
AS 'select $1 + $2;'
LANGUAGE SQL;

CREATE OR REPLACE FUNCTION incrementa(i integer)
RETURNS integer AS $$
BEGIN
RETURN i + 1;
END;
$$ LANGUAGE plpgsql;

select soma(3,4);
select incrementa(8);
```

Exemplos

```
-- totaliza os salarios dos empregados de um departamento
create or replace function F_TOTSAL (char)
returns float
as
'declare
      nomedept alias for $1;
      total float;
begin
      select into total sum(salario)
      from empregado E,departamento D
      where E.num_departamento=D.num_departamento and
      D.nome=nomedept;
      return total;
end'
language 'plpgsql';
```

Cursores em PL/pgSQL

- A) Declarando o Cursor:
  - *name* CURSOR [(*arguments*)] FOR *select_query* ;
- B) Abrindo o Cursor:
  - OPEN *name*;
- C) Buscando os dados armazenados no Cursor:
  - FETCH *name* INTO *target*;
- D) Fechando o Cursor:
  - CLOSE *name* ;

```
-- totaliza o numero de empregados que trabalham em um local de projeto
create or replace function F_TOTTRAB_LOCAL (char)
returns integer as
'declare
        local_proj alias for $1;
        total integer:=0;
        c_numproj cursor is select num_projeto from projeto where local=local_proj;
        parcial integer;
        np integer;
begin
        open c_numproj;
        loop
           fetch c_numproj into np;
           exit when not found;
           select count(*) into parcial from TRABALHO where num_projeto=np;
           total:=total+parcial;
        end loop;
        return total;
end'
language 'plpgsql';
```

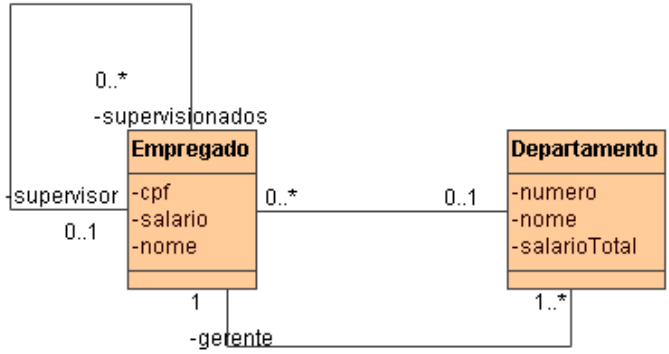
Triggers

- Conceito: procedimento armazenado executado automaticamente.
- Objetivo
  - monitorar uma base de dados e executar ações quando uma condição acontece
  - especificar certos tipos de regras ativas.
- Regras ativas
  - Regras que especificam ações que deve ser executadas automaticamente quando certos eventos ocorrem.
- Regras podem ser:
  - Para cada linha ou tupla
  - Para em nível de instrução

Regras ECA

- ECA (Event-Condition-Action)
  - Um modelo genérico para especificar regras de banco de dados ativos
- Conceitos
  - Evento
    - Qualquer evento temporal (ex: todo dia às 5h30)
    - geralmente atualizações no estado do banco de dados
  - Condição
    - Qualquer teste que seja aceitável em uma cláusula WHERE
    - determina se a ação da regra deve ser executada
      - A ação só é executada se a condição avaliar verdadeiro.
      - Se não houver condição definida, a ação é executada sempre.
  - Ação
    - uma transação de banco de dados ou um programa externo que será automatticamante executado.
    - usualmente uma sequência de comandos SQL

Exemplo de regra ativa




```
Empregado(cpf, nome, salario, cpf_supervisor, num_dpto)
        cpf_supervisor refencia Empregado
        num_dpto referenica Departamento
Departamento(numero, nome, salario_total, cpf_gerente)
        cpf_gerente referencia Empregado
```

- **Regras de negócio:**

1. **num_dpto** em empregado pode ser nulo, indicando que o empregado ainda não pertence a um departamento.
2. **salario_total**: é calculado como a soma dos salários de todos os empregados do departamento.

Pode ser atualizado através de uma regra ativa

Potenciais aplicações para Regras Ativas

- Notificação de certas condições
  - Exemplo: inserir um registro de excedente de temperatura sempre que um registro de temperatura for inserido e o valor for maior do que um limite.
- Impor restrições de integridade.
  - Exemplo: Não permitir que o salário de um funcionário seja atualizado para um valor superior ao do seu supervisor
- Manutenção automática de dados derivados
  - Exemplo: atualização de visões materializadas
- Auditoria de sistema
  - Exemplo: manter tabelas de histórico.

Triggers no PostgreSQL

- PostgreSQL e Oracle permitem definir tanto triggers por linha como triggers por comando SQL.
- Trigger por linha
  - A função da trigger é invocada uma vez para cada linha (ou registro) afetado pelo comando que chamou a trigger.
- Trigger por comando
  - A função da trigger é invocada apenas uma vez quando o comando é executado, não importando o número de linhas (ou registros) que foram afetadas pelo comando.
  - Mesmo que nenhuma linha seja afetada, a função da trigger será executada.

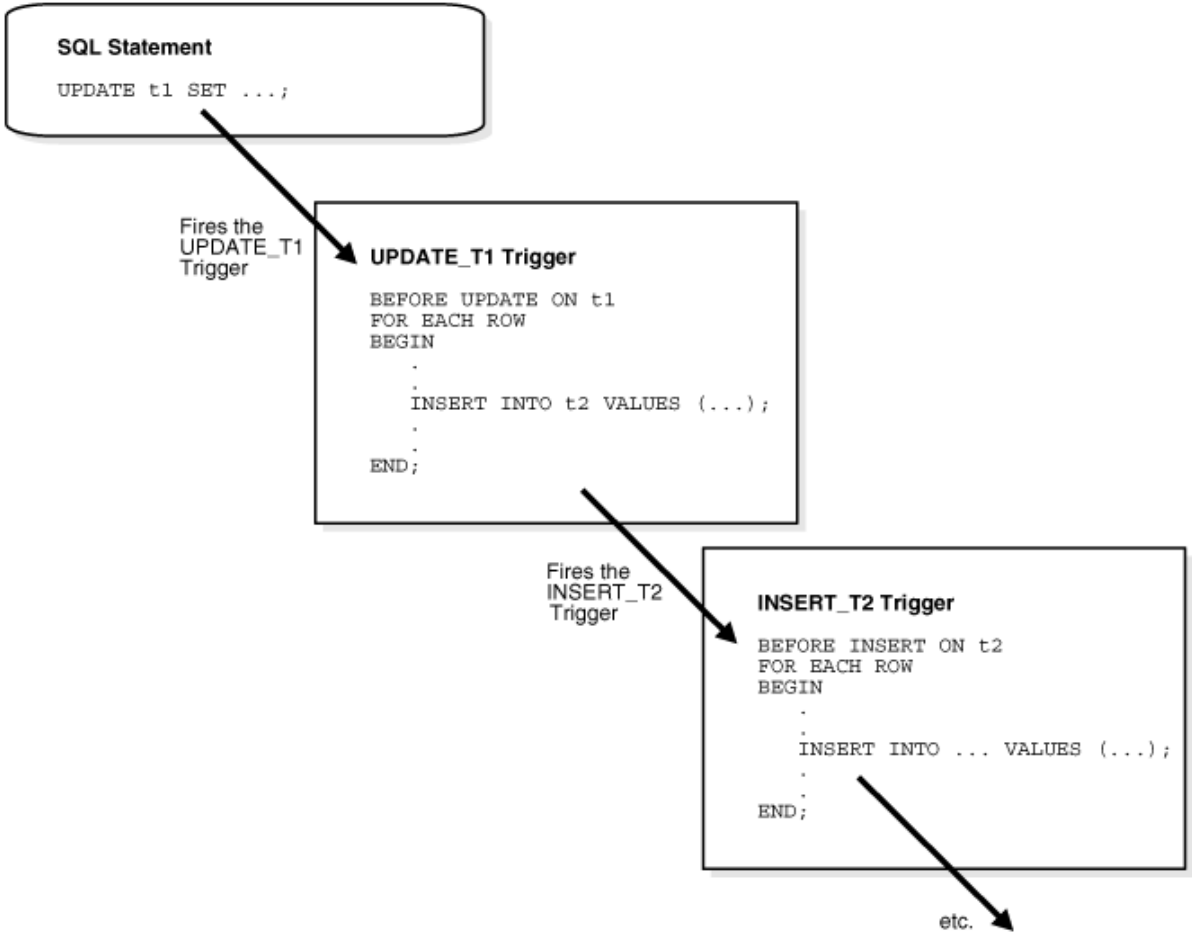
before vs *after*

- Nível de instrução
  - *before*: trigger é disparada antes da execução da instrução.
  - *after*: trigger é disparada depois que a instrução termina.
- Nível de linha
  - *before*: trigger é disparada antes da linha ser afetada
  - *after*: trigger é disparada depois que a linha é afetada.

Valor de retorno

- Funções de triggers por instrução devem sempre retornar NULL.
- Funções de triggers por linha podem retornar:
  - NULL: indicando que a operação não deve ser realizada (triggers na sequência não são disparadas, e a operação não ocorre)
  - Linha de tabela (INSERT ou UPDATE): a linha retornada será a linha inserida ou atualizada.
- O valor de retorno é ignorado para triggers a nível de instrução e para triggers de nível de linha de evento *after*.

Triggers em cascata



Triggers no PostgreSQL

- Se é definida mais de uma trigger para um mesmo evento, as triggers são disparadas em ordem alfabética.
- Os comandos SQL de uma trigger pode disparar outras triggers (*cascading triggers*).
  - A mesma trigger pode ser reinvocada em virtude do efeito em cascata (recursivamente).
  - O programador deve tratar possibilidade de recursão infinita.
- Qualquer tipo de trigger pode ser abortada se for levantado um erro.
  - Exemplo: RAISE EXCEPTION 'empname cannot be null';

CREATE TRIGGER

```
CREATE [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF }
{ event [ OR ... ] }
    ON table_name
    [ FROM referenced_table_name ]
    [ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY
DEFERRED ] ]
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( condition ) ]
    EXECUTE PROCEDURE function_name ( arguments )
```

https://www.postgresql.org/docs/9.5/static/sql-createtrigger.html

Vide tarefas no Moodle sobre Funções, Procedimentos e Triggers