Banco de Dados

AULA 11 – STORED PROCEDURES

Introdução

- O desenvolvimento de uma aplicação deve contemplar a forma de acesso ao SGBD e definições quanto à arquitetura e manutenção das regras de negócio.
- Regras de negócio estáticas são definidas como restrições (constraints) e associadas às tabelas, campos ou outros objetos que fazem parte do esquema do banco de dados.
- Regras mais complexas ou dinâmicas envolvem conceitos temporais, valores armazenados, etc.

Introdução

- Em diversas situações a manutenção de rotinas e a realização do processamento no servidor é uma alternativa importante.
- A partir da versão SQL:1999 é permitida a criação de procedimento (procedure), função (function) ou gatilho (trigger) usando construções procedurais da SQL ou outras linguagens como C, Java, etc.
- Diversos sistemas de bancos de dados aceitam suas próprias extensões procedurais da SQL, como PL/SQL, PL/pgSQL, Transact/SQL.

Introdução

- Além de regras de negócio para garantir a consistência e integridade do banco de dados, qualquer processo (consultas, atualizações) pode ser criado como uma procedure, function ou trigger.
- Cabe ao DBA projetar e implementar estas rotinas.
- É preciso definir se a rotina deverá ser executada sob demanda ou automaticamente.
- Desenvolvedores podem executar e manipular os resultados de um procedimento ou função.

Procedimento Armazenado (Stored Procedure)

- Procedure é um código procedural, semelhante ao utilizado em linguagens estruturadas.
- Stored Procedure é um código escrito em uma linguagem procedural ou SQL, e armazenado no banco de dados, disparado pelo usuário.
- Seu uso é aconselhado para funções que pertencem ao gerenciamento dos dados.
- Sua vantagem sobre outras alternativas diz respeito a pré-compilação e pré-normalização do código escrito, melhorando a performance de execução.

Stored Procedure

- Possui grande aplicabilidade em regras de negócio e em consultas que são utilizadas com mais frequência.
- Possui flexibilidade no uso de parâmetros em diferentes aplicações.
- Uma stored procedure é formada por um conjunto de instruções SQL e extensões SQL.
- Uma stored procedure fica gravada em um determinado banco de dados ou servidor e pode ser executada por qualquer aplicação cliente.

Stored Procedure

- Podem ser definidos diversos parâmetros de entrada e saída.
- Podem retornar valores, dependendo do tipo de stored procedure.
- Rotinas criadas como functions são criadas para retornar um ou mais valores.
- Alguns SGBDs possuem recursos para criação de procedures e functions e outros somente functions.
- Consultas e processos complexos podem ser realizados através de rotinas pré-definidas.

Vantagens de uso

- Comandos são compilados na primeira execução e mantidos em memória cache ou disco para posterior execução, o que permite execuções subsequentes mais rápidas.
- Ocultamento da complexidade de acesso ao banco.
- Disponibilidade em diversas aplicações.
- Centralização das operações.
- Flexibilidade através de parâmetros de entrada e saída que definem a execução e retorno.

Vantagens de uso

- Execução em servidor permite a redução de carga de processamento e tráfego de rede.
- Facilidade para gerência de segurança associada à determinados processos.
- Geração de dados em tabelas para testes.
- Execução de consultas com parâmetros.
- Execução de funções e processos que leem ou atualizam dados de diversas tabelas.
- Verificação de integridades complexas.

Implementação

- No PostgreSQL não existe apenas uma linguagem, mas um conjunto delas.
- Oficialmente distribuídas com o servidor são: PL/pgSQL, PL/TCL, PL/Python e PL/Perl.
- Ainda existem outras que podem ser instaladas e são providas por projetos paralelos como: PL/Java, PL/Ruby, PL/R.
- Na instalação, ou após ela, é necessário habilitar uma determinada linguagem.

Implementação

- Essa flexibilidade é obtida separando o mecanismo do banco de dados.
- Do executor das funções, o processo do PostgreSQL desconhece completamente o funcionamento interno de funções escritas em alguma PL (Procedure Language), ele apenas delega a sua execução para um trecho de código em C (compilado em uma biblioteca dinâmica).
- Inicialmente, nenhuma linguagem está habilitada (salvo na instalação para o banco template1), e a instrução para habilitar uma linguagem é:

```
CREATELANG linguagem banco_destino;
```

Implementação

- O PostgreSQL possui um mecanismo único de funções para implementação de stored procedures.
- Funções retornam valores, mesmo que sejam rotinas cujo principal objetivo é modificar dados em tabelas.
- Caso o retorno da função não seja importante é possível especificá-la como void (procedure).
- Funções podem ser executadas em comandos SELECT, que mostra o resultado ou retorna para a aplicação.

Implementação

- Comando para criação de uma função:

```
CREATE OR REPLACE FUNCTION nome (par tipo, …)
RETURNS tipo AS $$
BEGIN
...
END
$$ LANGUAGE plpgsql;
```

Implementação

- A linguagem define se as instruções serão apenas comandos SQL ou outra linguagem, tradicionalmente a PL/pgSQL.
- É possível criar em diversas linguagens desde que o suporte às mesmas tenha sido instalado.

Implementação

- A utilização de CREATE OR REPLACE FUNCTION recria a função sem alterar o ID anterior da função.
- Desta forma, uma função que referencia a mesma não precisa ser recriada.
- Este é um aspecto importante, uma vez a recriação de diversas funções pode ser uma atividade complexa.
- A linguagem permite a sobreposição de funções, portanto a eliminação de uma função envolve definição dos tipos dos parâmetros:

```
DROP FUNCTION nome (tipo);
```

Instruções comuns

```
DECLARE: Abre a seção de declaração das variáveis;

WHILE <condição> LOOP
...
END LOOP;

nome_variável := valor : Atribuição;

RETURN: Encerra e retorna os dados;

IF <condição> THEN
...
END IF;
```

Exemplo 1

Inserção de valores automáticos na tabela "Cidade".

```
CREATE OR REPLACE FUNCTION InsCidades (cidadeini INTEGER,
        cidademax INTEGER, pestado CHAR(2))
    RETURNS INTEGER
    AS $$
    DECLARE vcont INTEGER;
    BEGIN
        vcont := cidadeini;
        WHILE vcont <= cidademax LOOP
            INSERT INTO Cidade
            VALUES(vcont, 'Cidade ' || vcont,
                   pestado);
            vcont := vcont + 1;
        END LOOP;
        RETURN 0;
    END;
    $$ LANGUAGE plpgsql;

SELECT inscidades (250, 300, 'RS');
```

Cursores

- Um cursor permite executar uma consulta e armazenar o resultado em um buffer de memória.
- Instruções permitem navegar pelos registros.
- É possível verificar se chegou ao fim dos registros ou outro problema ocorreu.

Instruções para manipulação de cursores

Declaração:

```
DECLARE nome_cursor CURSOR FOR SELECT...
```

Abertura:

```
OPEN nome_cursor;
```

Leitura de registro:

```
FETCH nome_cursor INTO var1, var2, ...;
```

Fecha e libera:

```
CLOSE nome_cursor;
```

Exemplo 2

Utilização de um cursor para atualizar o número de atividades em um projeto.

```
CREATE OR REPLACE FUNCTION patutotproj ()
    RETURNS INTEGER
    AS $$
    DECLARE
        Curativ CURSOR FOR
            SELECT idproj, custoest
            FROM atividade;
        vidproj INTEGER;
        vcustoest DECIMAL(11,2);
        vcont INTEGER;
    BEGIN
        vcustoest = 500;
        OPEN Curativ;
        FETCH Curativ INTO vidproj, vcustoest;
        WHILE FOUND LOOP
            UPDATE projeto
            SET custoest:= custoest + vcustoest
            WHERE id = vidproj;
            FETCH Curativ INTO vidproj, vcustoest;
        END LOOP;
        CLOSE Curativ;
        RETURN 0;
    END;
    $$ LANGUAGE plpgsql;

SELECT patutotproj ();
```

Armazenamento de resultados

Instruções SELECT podem armazenar o resultado em variáveis ou tipos de registros, declarados como RECORD ou como cópia do tipo da tabela:

```
Ttabela tabela%ROWTYPE;
```

Exemplo 3

Utilização de uma variável de tipo registro (%ROWTYPE), retornando um valor, com diferentes formas de acesso a parâmetros.

```
CREATE OR REPLACE FUNCTION fmaiorduration (DATE, INT)
    RETURNS DECIMAL(11,2)
    AS $$
    DECLARE
        linhaativ atividade%ROWTYPE;
        vdataativ ALIAS FOR $1;
        vseqativ ALIAS FOR $2;
        vmaiorduracao DECIMAL(11,2);
    BEGIN
        IF EXISTS (SELECT *
                   FROM atividade
                   WHERE sequencia < vseqativ
                   AND datainiprev >= vdataativ) THEN
            SELECT *
            INTO linhaativ
            FROM atividade
            WHERE sequencia < vseqativ
            AND datainiprev >= vdataativ
            AND duracaoest >= ALL (SELECT duracaoest
                                   FROM atividade
                                   WHERE sequencia < vseqativ
                                   AND datainiprev >= vdataativ);
            vmaiorduracao := linhaativ.duracaoest;
        ELSE
            vmaiorduracao := null;
        END IF;
        RETURN vmaiorduracao;
    END;
    $$ LANGUAGE plpgsql;

SELECT fmaiorduration ('01/01/12', 5);
```

Armazenamento de resultados

Algumas instruções permitem funcionalidades semelhantes a outras, como no caso da simulação de um cursor usando a instrução FOR ... LOOP:

```
FOR registro IN
        SELECT *
        FROM tabela LOOP
    RETURN NEXT registro;
END LOOP;
```

Exemplo 4

Retorna um conjunto de dados do tipo cliente.
Declara uma variável de tipo RECORD.

```
CREATE OR REPLACE FUNCTION conscliente (pcodcid INTEGER)
    RETURNS SETOF cliente
    AS $$
    DECLARE inscli RECORD;
    BEGIN
        FOR inscli IN (SELECT *
                       FROM cliente c
                       WHERE c.codcid = pcodcid) LOOP
            RETURN NEXT inscli;
        END LOOP;
    END;
    $$ LANGUAGE plpgsql;

SELECT *
FROM conscliente (1);
```

Exemplo 5

Executa uma consulta com parâmetros. Retorna o resultado da mesma forma que uma view. A referência aos parâmetros é pela sequência.

```
CREATE OR REPLACE FUNCTION csativperiodo1 (
        pdataini DATE, pdatafim DATE)
    RETURNS SETOF atividade
    AS $$
    SELECT a.*
    FROM atividade a
    WHERE a.datainiprev BETWEEN $1 AND $2
    $$ LANGUAGE sql;

SELECT *
FROM csativperiodo1 ('01/01/10', '31/12/10');
```

```
CREATE TYPE tcsatividade AS (
    idproj CHAR(10),
    sequencia INT,
    nomeproj VARCHAR(150),
    nomeativ VARCHAR(150));

CREATE OR REPLACE FUNCTION csativperiodo2 (
        pdatainI DATE, pdatafim DATE)
    RETURNS SETOF tcsatividade
    AS $$
    SELECT a.idproj, a.sequencia, p.nome, a.nome
    FROM atividade a, projeto p
    WHERE a.datainiprev BETWEEN $1 AND $2
    $$ LANGUAGE sql;

SELECT *
FROM csativperiodo2 ('01/01/12', '30/04/12');
```

Leitura recomendada

- SILBERSCHATZ, Abraham. Sistemas de bancos de dados. 3. ed. São Paulo: Makron Books, 1999. Páginas 105 a 108.

Exercícios

Ver Lista 10