

Laboratório de Banco de Dados

Stored Procedures

1. Introdução

Uma sequência de comandos SQL pode ser agrupada, formando um objeto de catálogo que chamamos de “stored procedure”. Ela possui um nome e, diferentemente de uma simples sequência de comandos, que deixa de existir ao final de sua execução, uma stored procedure é permanente até que resolvamos removê-la. A invocação pelo seu nome dispara a execução e isto pode ser feito do interior de outra “stored procedure”. Além disso, como veremos em seguida, é muito mais do que um substituto ao agrupamento de comandos SQL que chamamos de “batch”.

2. Considerações sobre stored procedure

- a) Se a sequência de comandos é submetida várias vezes, não há a necessidade de submeter toda a sequência novamente, bastando invocar o nome da procedure. É mais prático do ponto de vista de codificação. Quando é preciso alterar alguma coisa, basta alterar na procedure e, conseqüentemente, a modificação se espalha para todos os pontos em que a sequência de código aparece na aplicação.
- b) Os comandos são pré-compilados e armazenados no banco de dados em formato de execução pelo banco de dados. Ou seja, não há a necessidade de recompilação do código fonte sempre que a execução for requisitada. O gerenciador já fez uma análise do código fonte no momento da criação da procedure e não precisa repeti-la em todas as execuções. Para ser mais exato, as análises léxica e sintática já foram validadas e concluída e está armazenada em forma de uma árvore de execução. A análise semântica é realizada quase na sua totalidade, mas existem alguns pontos que são adiados para o momento da execução, por exemplo, a referência a objetos temporários que estarão presentes posteriormente. A parte que precisa ser reavaliada na execução é o plano de acesso, pois este pode mudar de acordo com os dados.
- c) Diminui a quantidade de mensagens trocadas entre o requisitante e o servidor, porque toda a sequência é executada no escopo do servidor. Do contrário, cada comando da sequência é enviado ao servidor e retornado ao requisitante antes do envio do próximo comando. Entre estes comandos podem ser transferidos dados em volume significativo em termos de resultados intermediários ou mesmo finais a serem aplicados definitivamente no banco de dados. Dependendo da quantidade de comandos e do volume de dados trocados pela aplicação e o sistema de banco de dados, o tempo de execução é significativamente mais baixo utilizando “stored procedures”.
- d) Existe um ganho na segurança. Você não concede acesso às tabelas, mas concede somente o direito de execução sobre as procedures, estabelecendo com mais rigor quais operações o usuário pode executar. É possível um método misto de controle de acesso aos dados através de concessões a objetos executáveis e dados seletivamente de acordo com as necessidades.
- e) Os comandos de controle transacional podem ser deslocados para dentro do SGBD. Quando comandado pela aplicação, eventos de comunicação podem produzir bloqueios demorados sobre os recursos, por conexões aparentemente inativas. Neste aspecto específico, a contrapartida pode ser o deslocamento de algumas regras de negócio para o SGBD. Este fato, por si só, pode não representar um problema, a menos que contribua para a sobrecarga do sistema ou afete conceitualmente a arquitetura da solução.
- f) Podem ser passados parâmetros de execução.
- g) A portabilidade das aplicações é diminuída com a utilização de “stored procedures”.

3. Alguns comandos que podem fazer parte de uma procedure

As stored procedure tem certa similaridade com as procedures da Linguagem Pascal ou functions em C no que se refere à estrutura. Admitem parâmetros de entrada/saída. Permitem a utilização das construções SQL já estudadas, inclusive comandos de controle do fluxo de execução, repetições, declaração de variáveis e tratamento de transações. É importante ressaltar que neste nível de trabalho passar a ser comum a utilização de estruturas proprietárias.

Todos os comandos citados nas aulas anteriores, inclusive na aula sobre “SQL Batch” serão utilizados na codificação de “stored procedures”. Abaixo, seguem outros que não citamos.

3.1. Abandonar a execução da stored procedure:

Para interromper o fluxo de execução de uma procedure, utilizamos o comando “return”, como mostra o exemplo, mas a sua utilização não é necessária e, por vezes, desaconselhada.

```
if (condition)
begin
    ...
    return
end
```

3.2. Comando para criação de stored procedure

```
create procedure pr_exemplo (lista de parâmetros e respectivos tipos) as
begin
    .
    .
    .
    comandos sql ....
    .
    .
    .
end
```

Embora a sintaxe para “stored procedure” seja contemplada na definição padrão, os sistemas gerenciadores de banco de dados, em geral, não são completamente aderentes a ela. Na própria criação da procedure a palavra reservada “AS” é desnecessária, mas alguns sistemas, inclusive aqui do laboratório, a exigem. Outros utilizam “IS”; outros admitem ambas as palavras.

3.3. Comando para remoção de stored procedure

```
drop procedure pr_lista_cidade
```

3.4. Comando para emitir uma mensagem

Esta possibilidade de enviar mensagem à saída padrão pode depender do produto utilizado; no ambiente do laboratório, isto pode ser feito na forma mostrada abaixo. Construções mais elaboradas com formatação de texto são admitidas, mas a inclusão delas está fora do escopo neste ponto do curso.

```
print "Esta é uma mensagem..."
```

3.5. Comentários

Você pode incluir comentários nos seguintes formatos tanto em procedures como em “SQL Batch”:








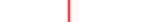






```
-- Isto é um comentário a partir deste ponto até o final da linha
```

```
/* Isto é um comentário  
com várias linhas  
e muito mais linhas  
e mais linhas.
```

```
    Pode também ser utilizado numa única linha.  
*/
```

4. Exemplo de criação de stored procedure

A procedure abaixo recebe como parâmetro o código de uma cidade. Apresenta os dados da cidade; caso não exista, apresenta uma mensagem de erro. No exemplo abaixo, existem trechos desnecessários que são apresentados apenas para exemplo.

```
create procedure pr_lista_cidade (  Comando de criação da procedure  
    @cod_cid varchar(15),  Parâmetros da procedure. Separados por vírgula.  
    @p1 numeric(5,2), @p2 float)  
as  
begin  
    declare @nome varchar(15),  Declaração de variáveis .  
        @habitantes int  
  
    select @nome = nome,  
        @habitantes = habitantes  Atribuição de valores às variáveis com  
        from tb_cidades1  valores obtidos do banco de dados.  
    where codigo = @cod_cid  
  
    set @nome = 'Exemplo',  Atribuição de valores às variáveis que  
        @habitantes = @habitantes + 2000  não são obtidos do banco de dados.  
  
    select codigo,  
        nome,  
        habitantes  Obtem os dados da cidade  
        qtd_ruas,  cujo código foi recebido  
        estado  como parâmetro.  
    from tb_cidades1  
    where codigo = @cod_cid  
  
    if (@@rowcount = 0)  Verifica se a cidade foi  
    begin  encontrada. Caso contrário,  
        print "cidade não existe."  apresenta mensagem de  
        return  erro.  
    end  
end -- fim do código fonte
```

Os comandos apenas ilustram o formato e organização de uma procedure; no exemplo não se destinam a resolver um problema concreto.

5. Observações sobre o exemplo

- a) Neste exemplo específico, os resultados poderiam ser obtidos de forma mais simples. A codificação exemplifica as formas de utilização de algumas estruturas.
- b) A última linha contém somente um comentário "--" após o símbolo delimitador de bloco ("end").
- c) Neste caso, todo o texto refere-se a um único comando de criação de procedure e, portanto, terá de executá-lo de uma só vez. Se você tentar executar mais que um comando, por exemplo, um **drop procedure pr_lista_cidade** seguido do código fonte para a sua criação, ocorrerá erro. Será necessária a colocação do símbolo que submete cada um deles na sua integralidade. No ambiente do laboratório, nenhum símbolo adicional ao código fonte deve ser utilizado porque o “botão” de envio indica o término da sequência de comandos. Estas questões dependem da ferramenta cliente utilizada. Não faz parte da procedure; Apenas informa que todo o arquivo fonte deve ser submetido ao servidor como um único bloco de comandos.
- d) Observe como a atribuição de variáveis é realizada. Esta sintaxe não é comum a todos os gerenciadores.
- e) Para executar a procedure, substitua os valores dos parâmetros por algum que seja válido, nas chamadas abaixo:

```
exec pr_lista_cidade 'aaa'  
execute pr_lista_cidade 'bbb'
```

Se os parâmetros fossem delimitados como uma sequência entre parênteses a sintaxe ficaria mais consistente com as linguagens de programação usuais. Em alguns gerenciadores é assim que deve ser. No entanto, em outros não podem ser colocados estes delimitadores, inclusive aqui no laboratório.

6. Distância média

Para estudarmos na prática “stored procedure”, vamos utilizar o assunto visto em “SQL Batch” para codificação de exemplos. Naquela aula, analisamos a distribuição de pessoas através de uma rede de amizades, formando um grafo e definimos alguns termos específicos sobre o universo tratado, como requisitantes ou predecessores, respondentes ou sucessores que podem ser diretos e indiretos. Empregamos a palavra “amigos” para indicar que podem ser respondentes ou requisitantes, mas de qualquer forma, podem ser também diretos e indiretos. No roteiro daquela aula, apresentamos um “SQL Batch” que mostra o conjunto de todos os amigos diretos e indiretos de um indivíduo. Para efeito deste roteiro, definimos a distância média como sendo a média da quantidade de arestas que separa o indivíduo de todos os seus amigos diretos e indiretos calculada de forma ponderada. Para melhor explicação, vamos falar em termos de vértices e arestas.

Considere um vértice v e o conjunto $V = \{v, v_1, v_2, v_3, \dots, v_p\}$, ao qual pertencem todos os predecessores e sucessores diretos e indiretos de v na forma calculada em roteiro específico. Lembre-se de que o “SQL Batch”, apresentado naquela ocasião, inclui o vértice v no resultado com a distância zero, ou seja, uma pessoa é amiga dela mesma e está à distância de zero aresta. Analisando o subgrafo que engloba os vértices de V e as arestas que os interligam, concluímos que ele é conexo; mais do que isto, um subgrafo conexo maximal. Deste modo, todas as pessoas representadas pelos vértices do subgrafo possuem o mesmo conjunto V na forma definida. O que muda é a distância média entre eles cuja forma de cálculo será explicada a seguir.

Vamos denotar por

$d(v, w)$, a menor distância, em termos de arestas, entre os vértices v e w .

Deste modo, $d(v, v), d(v, v_1), d(v, v_2), d(v, v_3), \dots, d(v, v_p)$ são as distâncias de v a cada um dos vértices de V .

6.1. Utilizando a função de agregação “sum”

A distância média pode ser calculada segundo a seguinte fórmula:

$$dist_{media} = \frac{\sum_{i=1}^p (d(v, v_i))}{p + 1}.$$

6.2. Utilizando a função de agregação “sum” sobre o produto entre a distância e a quantidade do grupo de amigos inseridos a cada iteração

A cada iteração do algoritmo um conjunto de vértices é incluído e todos eles estão à mesma distância, ou seja, a contribuição deste grupo de elementos à distância média pode ser contabilizada parcialmente de modo incremental à medida que o algoritmo avança. Vamos explicar esta abordagem.

Sejam $V_0, V_1, V_2, V_3, \dots, V_m$ subconjuntos de V , tais que:

- a) $V = \cup_{i=0}^m V_i$
- b) $V_i \cap V_j = \emptyset$, para todo $i, j = 0, \dots, m$ com $i \neq j$, ou seja, os subconjuntos são disjuntos.
- c) Todos os vértices de V_i estão à mesma distância i de v , ou seja,
 $w \in V_i$ se $d(v, w) = i$ para todo $i = 0, 1, 2, \dots, m$.

Obviamente, $V_0 = \{v\}$.

Seja

$$n_i = |V_i| \text{ para } i = 0, \dots, m.$$

Podemos observar que n_i é a quantidade de vértices de V que estão à mesma distância de i arestas de v para todo

$$i = 0, 1, 2, 3, \dots, m.$$

Claramente, $\sum_{i=0}^m n_i = p + 1$.

A distância média do vértice v em relação ao conjunto V é definida da seguinte forma:

$$dist_media = \frac{\sum_{i=0}^m (i \times n_i)}{\sum_{i=0}^m n_i} \text{ ou } \frac{\sum_{i=0}^m (i \times n_i)}{p + 1}.$$

O primeiro termo na somatória do numerador é indiferente porque

$$n_0 = 1 \text{ e } i = 0.$$

Deste modo a contribuição do primeiro termo na somatória do numerador é zero, mas ele deve ser considerado na contagem dos elementos que aparece no denominador da expressão. Isto é decorrente da definição que

adotamos de distância média. Assim, podemos alternativamente escrever da seguinte forma, sem alterar o resultado:

$$dist_media = \frac{\sum_{i=1}^m (i \times n_i)}{(\sum_{i=1}^m n_i) + 1} = \frac{\sum_{i=1}^m (i \times n_i)}{\sum_{i=0}^m n_i} = \frac{\sum_{i=1}^m (i \times n_i)}{p + 1}.$$

Para compreendermos a situação do próprio vértice v para o qual calculamos a distância média, suponha a situação mostrada abaixo dos amigos diretos e indiretos de “P100000” que foi obtida do banco de dados executando o script apresentado no roteiro sobre “SQL Batch”. A linha marcada com a cor amarela é o próprio vértice (distância zero). Teoricamente, poderíamos definir a distância média de forma diferente e escolher por desconsiderá-la na contagem do número de vértices e teríamos resultados diferentes quando um vértice não for isolado, porém, na hipótese de ser isolado, precisaríamos tratar de forma diferente porque o número de elementos seria zero o que impossibilita a sua utilização no denominador, sendo necessário tratar este caso particular como uma exceção à fórmula.

No cálculo que vamos realizar, cada um dos vértices do grafo deve ser levado em conta somente uma vez, ou seja, as linhas em vermelho devem ser contadas apenas uma vez, pois se referem ao mesmo indivíduo que pode ser alcançado à mesma distância por vizinhos anteriores distintos. Apenas lembrando que, se o vértice for atingível a distâncias diferentes, o algoritmo apresenta somente os dados referentes à menor delas o que é correto segundo as nossas definições sobre o tema.

Encontrar os amigos diretos ou indiretos de

P100000

nivel	cod_pes	nome	data_nasc			cod_pes_anterior	nome_pes_anterior
-----	-----	-----	-----	-----	-----	-----	-----
0	P100000	Sandro	Jan	2	2000	0000000	NULL
1	P101000	Ludmila	Apr	18	2002	P100000	Sandro
1	P102000	Adriana	Jul	18	2001	P100000	Sandro
1	P103000	Santoro	Dec	9	2004	P100000	Sandro
1	P104000	Tobias	Oct	23	1999	P100000	Sandro
1	P104200	Marco	Feb	18	2002	P100000	Sandro
2	P101100	Jamil	Apr	20	2004	P101000	Ludmila
2	P101100	Jamil	Apr	20	2004	P102000	Adriana
2	P101100	Jamil	Apr	20	2004	P103000	Santoro
2	P104100	Andressa	Nov	9	2001	P104000	Tobias
3	P101110	Gian	Mar	25	2003	P101100	Jamil
3	P101120	Amanda	Oct	15	2000	P101100	Jamil
3	P101130	Fabiola	Nov	27	2000	P101100	Jamil
4	P101111	Cassia	Apr	24	2002	P101110	Gian

Aplicando a definição de distância média, obtemos o seguinte resultado:

$$dist_media = \frac{(0 \times 1) + (1 \times 5) + (2 \times 2) + (3 \times 3) + (4 \times 1)}{(1 + 5 + 2 + 3 + 1)} = \frac{(0 + 5 + 4 + 9 + 4)}{(1 + 5 + 2 + 3 + 1)} = \frac{22}{12} = 1,833$$

7. Casting

Existem situações em que há a necessidade de conversão de tipo de dados. Por exemplo, um número numa sequência de caracteres ou vice-versa; um número inteiro para decimal ou ponto flutuante; horário para “string”; “smallint” para “double”.

Este mecanismo foi tema de estudo anterior, mas vamos lembrá-lo. Os gerenciadores de bancos de dados disponibilizam funções que realizam esta tarefa, mas apresentam diferenças entre eles. Seguem alguns exemplos:

1. `cast(expression to new_data_type)`

```
select cast(data_nasc to varchar(20)) from tb_pessoas_teste
```

2. `cast(expression as new_data_type)`

```
select cast(data_nasc as varchar(20)) from tb_pessoas_teste
```

3. `convert(new_data_type, expression)`

```
select convert(varchar(20), data_nasc) from tb_pessoas_teste
```

No laboratório, vamos utilizar o segundo modelo de conversão que é utilizando também em alguns outros sistemas de banco de dados. Por exemplo:

```
select cast(data_nasc as varchar(20)) from tb_pessoas_teste
```

```
select count(*) / count(distinct data_nasc) as with_no_conversion,  
       (cast(count(*) as numeric(10,2)) / count(distinct data_nasc)) as with_conversion  
from tb_pessoas_teste
```

```
select nivel,  
       count(distinct cod_pes) as qtd,  
       nivel * count(distinct cod_pes) as multiplicacao,  
       (select count(*) from tb_pessoas_ligadas1) as qtd_total,  
       nivel * count(distinct cod_pes) /  
         (select count(*) from tb_pessoas_ligadas1) as with_no_conversion,  
       cast(  
         cast(nivel * count(distinct cod_pes) as numeric(5,2)) /  
         (select count(*) from tb_pessoas_ligadas1)  
         as numeric(5,2)) as with_conversion  
from tb_pessoas_ligadas1  
group by nivel
```