Laboratório de Banco de Dados

Continuando com nossos estudos sobre SQL, vamos tratar, neste roteiro, de dois temas. O primeiro é sobre visão ("view") que nos permite pré-definir um comando de seleção, explicitamente nomeada, para utilização futura de modo que constitui um elemento de catálogo, ou seja, permanente. Não estamos falando de dados, mas da definição do objeto. O segundo tema refere-se a tabelas derivadas.

1. View (Visão)

No início da disciplina de laboratório, definimos o conceito matemático de relação e o traduzimos como sendo popularmente uma tabela. Uma visão mantem as propriedades de uma relação no sentido estudado, mas é um objeto que, opcionalmente, pode não armazenar fisicamente os dados. Podemos pensar que é uma tabela que existe logicamente. Neste sentido, existe para os usuários, mas não existe para a infraestrutura de armazenamento. Ela é definida em termos do comando "select".

Podemos criar visões por diferentes motivos, por exemplo:

a) Apresentar os dados pertinentes somente ao interessado ou à área do interessado, restringindo o conjunto dos dados a um subconjunto definido pela visão;
b) Políticas de segurança como uma medida alternativa ou complementar;
c) Facilidade para expressar as consultas porque elas ficam salvas no catálogo do banco de dados e são referenciadas pelo nome da "visão", ocultando a complexidade do comando.

A seguir, um comando simplificado para criação de uma view. Este é apenas um exemplo. Várias considerações adicionais podem ser feitas, mas fica para um estudo aprofundado por parte do aluno.

```
create view vw_pessoas_marte (cod_pessoa,
                              nome_pessoa,
                              data_nascimento,
                              cidade_nascimento) as
       select cod,
              nome,
              dt_nasc,
              cidade_natal
         from tb_pessoas_prof
        where cidade_natal = 'Marte'
```

No exemplo, a visão contempla o nome e cidade natal das pessoas que nasceram em Marte. Uma seleção pode ser realizada diretamente sobre a visão:

```
select * from vw_pessoas_marte
```

Como já foi dito, a visão pode ser definida sobre tabelas, mas também pode envolver outras visões previamente definidas. O exemplo simples acima não representa a complexidade que pode envolver uma visão.

Para remover a view:

```
drop view vw_pessoas_marte
```

2. Atualização através de visões

Em geral, todas as operações de consulta aplicáveis sobre tabelas são também aplicáveis às visões. No entanto as operações que modificam os dados (insert, delete, update) somente podem ser realizadas se satisfizerem a várias restrições. Para lembrar um dos pontos que inviabilizam a inserção de dados, a visão pode ser definida através da operação de junção e, portanto, envolvendo diversas tabelas de forma que a operação de inserção que deve ocorrer sobre as tabelas base fica difícil de ser realizada. Uma segunda questão é que a visão pode estar definida sobre parte das colunas de uma tabela e uma inserção de dados pode ficar prejudicada. Outro obstáculo se refere às visões que envolvem funções de agregação. Algumas propriedades são dependentes de produto e devem ser verificadas na documentação dos mesmos que discorrem com profundidade sobre a condição de ser atualizável. De modo simplificado, podemos concluir que as atualizações, incluindo a remoção, inserção e alteração dos dados são restritas. De novo, embora isto dependa do produto específico, em geral, as “views” atualizáveis precisam ser definidas somente sobre uma tabela; as colunas “da select_list” não podem conter funções escalares ou de agregação nem expressões envolvendo colunas e constantes; A “select_list” não pode conter a cláusula “distinct”; as colunas ausentes na “select_list” precisam admitir valores indefinidos, inaplicáveis, desconhecidos ou não informados; o comando “select” que define a “view” não pode conter as cláusulas “group by” nem “having”. Outro efeito colateral da atualização de uma “view” ocorre nos eventos de alteração dos dados quando o resultado do comando tem como consequência o desaparecimento da tupla na apresentação dos seus dados; ou a inclusão de dados através da “view” que não aparecem no resultado da seleção sobre a mesma. Isto pode ser evitado com a utilização da cláusula “with check option” na definição da “view”.

2.1. Inserção, atualização e exclusão de dados através de visão

Como explicado acima, existem condições que estabelecem se uma visão é ou não atualizável.

A visão definida no exemplo acima aceita a inserção de dados. Por exemplo:

```
insert into vw_pessoas_marte (cod_pessoa,
                              nome_pessoa,
                              data_nascimento,
                              cidade_nascimento)
       values('C0008', 'Spock', '31/08/1950', 'Monte Santo')
```

O exemplo do comando de inserção possui uma anomalia sutil que discutiremos mais abaixo na próxima item. Consultando os dados, podemos observar que o comando de inserção através da visão foi concluído.

```
select * from tb_pessoas

cod    nome                                   dt_nasc          cidade_natal
------ -------------------------------------- ---------------- ------------------------
C0001  Letsgo Daqui                                Apr 30 2005 Marte
C0002  Hericlapiton dos Santos                     Jun 18 1999 Jijoca de Jericoacoara
C0003  Méki Um Dois e Três                         Sep  4 2001 Uauá
C0004  Simplício Simplório Ains Tain               Jan  1 2000 Saturno
C0005  India Ana Jones Brasil                      Nov 27 1985 Plutão
C0006  Cosette Fantine de Myriel Valjean           Mar 29 1934 Urano
C0007  Sofia Virgília Cubas de Quincas Borba       Oct 19 1986 Netuno
C0008  Spock                                       Aug 31 1950 Monte Santo
C0009  Comandante Kirk                             Dec 21 1947 Marte

(9 rows affected)
```

De forma análoga, os comandos de exclusão e atualização podem ser executados com as ressalvas das condições discutidas a seguir.

2.2. Efeitos colaterais nas modificações dos dados sobre visões

Veja que no exemplo anterior, inserimos uma pessoa através da visão à qual a linha inserida não pertence. Ou seja, a nova linha aparece no resultado da consulta à tabela tb_pessoas, mas não na visão que serviu para fazer a inserção, como podemos observar:

```
select * from vw_pessoas_marte

cod_pessoa nome_pessoa                        data_nascimento  cidade_nascimento
---------- -------------------------------------- ---------------- ------------------------
C0009      Comandante Kirk                         Dec 21 1947 Marte

(1 rows affected)
```

Se existisse outra visão para contemplar os nascidos na cidade de ‘Monte Santo’, essa linha recém-incluída apareceria como no resultado.

Comandos de atualização também podem produzir efeitos análogos. Suponha que, além da visão que define os nascidos em Marte, exista outra que define as pessoas nascidas em Saturno. Pode ser realizada uma alteração nos dados utilizando a primeira visão, mas que afetam os resultados da segunda. Ou um comando de atualização que transfere os dados de uma visão para outra. Ou seja, um usuário que esteja trabalhando com a segunda não entenderá o que aconteceu visto que ele não fez nada que pudesse alterar os dados do seu negócio.

Analogamente, o comando de exclusão pode provocar efeitos estranhos. Suponha que removamos, através de uma visão, uma tupla que não pertence ao conjunto de dados que a define. Isto é estranho porque removemos uma linha não que vemos.

Como mencionado acima, pode ser adicionada a cláusula “with check option” para que estes efeitos colaterais sejam verificados. Por exemplo:

```
create view vw_pessoas_marte (cod_pessoa,
                              nome_pessoa,
                              data_nascimento,
                              cidade_nascimento) as
       select cod,
              nome,
              dt_nasc,
              cidade_natal
         from tb_pessoas
        where cidade_natal = 'Marte'
  with check option
```

Com a visão criada desta forma, dados somente podem ser inseridos através dela se forem de pessoas nascidas em Marte. Do mesmo modo, alterações na tabela não podem alterar a cidade Natal para um valor não definido na visão. A remoção de linhas também devem afetar somente as linhas contempladas na visão, ou seja, uma tentativa de remover uma linha da tabela que não pertença à visão deve retornar uma mensagem de que nenhuma linha foi afetada pelo comando.

2.3. Visões não atualizáveis

Existem alguns requisitos necessários para que as visões sejam utilizadas em comandos que alteram os dados. Vamos mostrar algumas situações (não é uma lista completa) em que não é possível alterar o conteúdo das tabelas através de visões.

a) A “select_list” da definição da visão não contempla todas as colunas que não admitem valores nulos. É claro que neste caso, não será possível inserir dados através da visão.
b) Colunas com valores calculados. Por exemplo, suponha a seguinte visão:

```
create view vw_pessoas_aumento_salario(
       cod_pessoa,
       nome_pessoa,
       novo_salario)
    as select cod,
              nome,
              1.3709 * salario  /***** expressão *****/
         from tb_pessoas
```

É claro que, pelo motivo anterior, esta visão não pode ser utilizada para inserção ou atualização de dados.

```
update vw_pessoas_aumento_salario
   set novo_salario = 2000
 where cod_pessoa = '0001'
```

c) Visões utilizam funções de agregação com agrupamento que foram estudadas em aulas anteriores.

3. Materialized View (Visão materializada)

Na explicação acima, tratamos da visão (view) clássica. Muitos produtos oferecem a possibilidade da visão materializada, ou seja, ela não é apenas um objeto de catálogo, mas também um objeto que armazena fisicamente os dados. Neste tipo de visão, alguns aspectos precisam ser considerados e detalhes são muito dependentes de produtos. Segue uma lista incompleta de itens que merecem reflexão e que eventualmente não estão disponíveis em todos os produtos ou que sofrem restrições de acordo com as características específicas da visão.

a) Alocações de espaço;
b) Indexação;
c) Particionamento;
d) Compressão;
e) Método de materialização inicial (agendamento, execução de comando próprio de materialização, imediato);
f) Método de sincronismo (transacional síncrono, contínuo assíncrono, sob demanda por execução de procedimento específico, execução do comando gerador da própria view).

Claramente, os dados de uma visão materializada são redundantes, ou seja, derivados e como tal precisam de todos os cuidados para a manutenção da consistência. A primeira pergunta é sobre a real necessidade deste objeto uma vez que os dados já existem nos objetos que são base da visão. Considerando apenas as motivações citadas no item da visão convencional, não haveria a necessidade para isto. No entanto, em alguns casos, pode ser aplicável. Por exemplo, se a visão requer um processamento custoso para a apresentação dos dados que são pouco modificados e muito consultados. Visões frequentemente acessadas e que calculam funções de agregação com diferentes formas de agrupamentos sobre grande volume de dados são também exemplos que merecem análise de aplicabilidade.

Este objeto de banco de dados pode não estar disponível em alguns produtos.

4. Tabelas derivadas

Vimos na primeira aula que uma relação é um conjunto de tuplas e cada tupla é uma sequência de valores e cada um destes valores pertence a um domínio. Também, podemos ver que o resultado de um comando “select” tem as propriedades de uma relação matemática. Deste modo tanto as tabelas quanto os resultados dos comandos de busca são relações, compartilham as mesmas propriedades, exceto que as primeiras têm propriedades físicas de armazenamento e os segundos apenas existem para fins de apresentação dos dados, transitoriamente. A álgebra relacional define que toda a operação opera sobre relações e resultam em relações e isto concede o poder da álgebra com apenas poucas operações fundamentais, uma vez que operações podem ser aplicadas sobre os resultados.

Posto que a principal propriedade das relações é compartilhada por tabelas e por resultados das consultas aos bancos, podemos estender o conceito de tabela da seguinte forma:

a) Tabela física será chamada de tabela;
b) A relação resultante de comando (select) será chamada de tabela derivada ou virtual que existe apenas logicamente. Sobre o tempo de sua existência podemos dizer que tem a duração do comando, nem chegando à extensão da transação.

Assim, vamos estender o alcance dos comandos SQL a ambos os tipos de tabela de forma que as tabelas derivadas também possam participar de comandos de modo análogo às demais tabelas físicas. De algum modo, pode parecer que é similar às "sub-queries", porque são estruturas aninhadas, mas existe uma diferença fundamental: Como sub-query, o comando interno aparece no predicado ou na "select_list" e este não é o lugar destinado às tabelas. O lugar destes é na cláusula "from" onde colocamos a "table list".

A seguir, um exemplo simples para mostrar a diferença nas diversas formas de especificação do comando "select" numa estrutura aninhada. No exemplo, suponha que desejamos o código, nome da cidade e o nome do estado de localização das cidades cujos nomes têm o prefixo 'Sa' e pertencem aos estados cujos nomes têm o prefixo 'Rio'. O exemplo desfavorece a utilização de "sub-query" porque desejamos também os dados da tabela de estados. Ao contrário, se desejássemos somente os dados das cidades sendo a tabela de estados apenas para verificar a pertinência a estados específicos, o uso do artifício da "sub-query" seria beneficiado. Por curiosidade, veja que o sistema está ajustado para não fazer distinção entre letras maiúsculas e minúsculas e acentuações. Além disso, as cidades apresentadas são fictícias embora algumas sejam homônimas de cidades reais.

Exemplo 4.1. Comando de seleção utilizando "inner join" ou com restrição ao produto cartesiano

Considere os comandos seguintes que são equivalentes sobre as tabelas de cidades e estados cujos conteúdos irreais são mostrados na figura à frente.

```
select codigo, nome, est_nome
  from tb_cidades,
       tb_estados
 where tb_cidades.estado = tb_estados.est_sigla
   and nome like 'Sa%'
   and est_nome like 'Rio%'
```

ou

```
select codigo, nome, est_nome
  from tb_cidades
         inner join
       tb_estados
         on tb_cidades.estado = tb_estados.est_sigla
 where nome like 'Sa%'
   and est_nome like 'Rio%'
```

Considere os seguintes dados das tabelas de cidades e estados.

codigo	nome	habitantes	qtd_ruas	estado
1	SÃO PAULO	9000000	500000	SP
2	BRASÍLIA	890	50000	DF
4	SANTO ANDRÉ	300001	50001	RJ
5	SÃO CAETANO	47000	8600	RN
6	SÃO CARLOS	34235	501	RS
7	CEILÂNDIA	27000	6500	DF
8	BRASILÂNDIA	28000	8509	DF
9	MAUÁ	35001	5301	SP
10	OSASCO	550001	80001	SP
11	JK	10000	778	SP
20	MINAS DE OURO	5500	400	PE
21	RIO BONITO	540000	6000	CE
22	FEIRA LIVRE	520000	90000	SC

tb_cidades

est_sigla	est_nome	reg_sigla
AC	Acre	N
AL	Alagoas	NE
AM	Amazonas	N
AP	Amapá	N
BA	Bahia	NE
CE	Ceará	NE
DF	Distrito Federal	CO
ES	Espirito Santo	SE
GO	Goiás	CO
MA	Maranhão	NE
MG	Minas Gerais	SE
MS	Mato Grosso do Sul	CO
MT	Mato Grosso	CO
PA	Pará	N
PB	Paraíba	NE
PE	Pernambuco	NE
PI	Piauí	NE
PR	Paraná	S
RJ	Rio de Janeiro	SE
RN	Rio Grande do Norte	NE
RO	Rondonia	N
RR	Roraima	N
RS	Rio Grande do Sul	S
SC	Santa Catarina	S
SE	Sergipe	NE
SP	São Paulo	SE
TO	Tocantins	N

tb_estados

O comando aplicado resultaria nas seguintes tuplas:

```
codigo          nome                           est_nome
--------------- ------------------------------ --------------------
0004            SANTO ANDRÉ                    Rio de Janeiro
0005            SÃO CAETANO                    Rio Grande do Norte
0006            SÃO CARLOS                     Rio Grande do Sul
```

4.2. Comando de seleção com sub-query (Vide “search condition”).

Como dito no enunciado da questão, esta abordagem não é beneficiada neste caso porque existem dados da tabela de estados a serem apresentados e não basta a verificação de pertinência. Ela é apresentada para comparação.

```
select codigo, nome, est_nome
  from tb_cidades, tb_estados
 where tb_cidades.estado = tb_estados.est_sigla
   and est_nome like 'Rio%'
   and codigo in (select codigo
                    from tb_cidades
                   where nome like 'Sa%')
```

```
select codigo, nome, est_nome
  from tb_cidades, tb_estados
 where tb_cidades.estado = tb_estados.est_sigla
   and est_nome like 'Rio%'
   and exists (select 1 codigo
                 from tb_cidades cid_sub_query
                where nome like 'Sa%'
                  and cid_sub_query.codigo = tb_cidades.codigo)
```

Resultado:

```
 codigo          nome                           est_nome
 --------------- ------------------------------ --------------------
 0004            SANTO ANDRÉ                    Rio de Janeiro
 0005            SÃO CAETANO                    Rio Grande do Norte
 0006            SÃO CARLOS                     Rio Grande do Sul
```

4.3. Comando de seleção com tabela derivada.

Veja que as tabelas derivadas aparecem na “table list”, ou seja, na cláusula “from”. As renomeações, no exemplo, foram utilizadas para ilustrar o fato de que elas podem ser utilizadas da mesma forma como em outras situações aplicáveis e neste caso precisamos dos nomes para expressar a cláusula de junção.

```
select tb_cidades_virtual.codigo as "código",
       tb_cidades_virtual.nome as "nome da cidade",
       tb_estados_virtual.est_nome as "nome do estado"
  from (select codigo, nome, estado
          from tb_cidades
         where nome like 'Sa%') as tb_cidades_virtual,
       (select est_sigla, est_nome
          from tb_estados
         where est_nome like 'Rio%') as tb_estados_virtual
 where tb_cidades_virtual.estado = tb_estados_virtual.est_sigla
```

Resultado:

```
código          nome da cidade                 nome do estado
--------------- ------------------------------ --------------------
0004            SANTO ANDRÉ                    Rio de Janeiro
0005            SÃO CAETANO                    Rio Grande do Norte
0006            SÃO CARLOS                     Rio Grande do Sul
```

Tudo se passa como se existissem duas tabelas virtuais como mostradas na figura abaixo:

codigo	nome	estado
1	SÃO PAULO	SP
4	SANTO ANDRÉ	RJ
5	SÃO CAETANO	RN
6	SÃO CARLOS	RS

tb_cidades_virtual

est_sigla	est_nome
RJ	Rio de Janeiro
RN	Rio Grande do Norte
RS	Rio Grande do Sul

tb_estados_virtual

Utilizando tabelas derivadas, elas foram renomeadas para que as referências sejam realizadas corretamente.

Não devemos complicar o que é simples. Até por uma questão de legibilidade. Eventualmente, em alguns Sistemas Gerenciadores de Bancos de Dados, pode envolver questões de otimização.

Então quando devemos utilizar a tabela derivada? Em casos estritamente necessários. Quando precisarmos fazer uma junção com os dados que serão obtidos e que não podem ser especificados no comando principal. Exemplo: Cláusula de junção que envolve colunas que são resultados de função de agregação com agrupamento. Também é necessário quando se deseja fazer o agrupamento baseado em coluna computada (“computed column”).

Segue um exemplo:

Suponha a tabela de nome tb_orc_adicional que possui duas colunas: a primeira de nome “qtd_cid” que indica quantas cidades que os estados possuem; e a segunda coluna de nome “valor” que indica quanto o estado receberá de valor adicional ao orçamento por conta desta quantidade de cidades. Esta quantidade deve ser obtida através de uma junção com a tabela de cidades e função de agregação com agrupamento.

Abaixo, um pedaço dos dados da tabela tb_orc_adicional para seu entendimento. Suponha que ela está preenchida com todos os valores possíveis de quantidade de cidades por estado.

qtd_cid	valor
1	1000,00
2	2500,00
3	2700,00
4	4000,00
5	4500,00
...	...
...	...
...	...
1000	50000,00

tb_orc_adicional

Como apresentar a sigla, o nome do estado, quantidade de cidades e o valor do adicional ao orçamento? Este valor depende da quantidade de cidades de acordo com o que está estabelecido na tabela tb_orc_adicional. Abaixo, segue a solução utilizando a ideia de "tabela derivada" que está em "vermelho". Esta tabela virtual apresenta os estados com as respectivas quantidades de cidades.

```
select est_cid.est_sigla,
       est_cid.est_nome,
       est_cid.qtd_cid,
       o.qtd_cid,
       o.valor
  from (select est_sigla,
               est_nome,
               count(*) as qtd_cid
          from tb_estados e,
               tb_cidades c
         where c.estado = e.est_sigla
         group by est_sigla, est_nome) as est_cid,
       tb_orc_adicional as o
 where est_cid.qtd_cid = o.qtd_cid
 order by valor
```

Resultado:

```
 est_sigla est_nome             qtd_cid     qtd_cid     valor
 --------- -------------------- ----------- ----------- -------------
 AM        Amazonas                       1           1       1000.00
 CE        Ceará                          1           1       1000.00
 MG        Minas Gerais                   1           1       1000.00
 PE        Pernambuco                     1           1       1000.00
 PR        Paraná                         1           1       1000.00
 SC        Santa Catarina                 1           1       1000.00
 DF        Distrito Federal               4           4       4000.00
 RJ        Rio de Janeiro                 4           4       4000.00
 SP        São Paulo                      9           9       7500.00
```

Algumas das situações analisadas podem ser resolvidas com o apoio de visões, mas precisamos considerar que estas são objetos permanentes no catálogo do sistema que requerem atividades administrativas como gerenciamento de permissões e se forem necessárias para a execução de somente

um comando, precisam ser removidas do sistema. Além disso, pequenas alterações nos requisitos que poderiam ser resolvidas localmente no comando podem implicar em manutenção adicional das visões. Enfim, existem situações em que tabelas derivadas são adequadas. Por outro lado, se uma visão é útil em várias aplicações, justificando a condição de objeto permanente, não há razão para não utilizá-las.

Satoshi Nagayama.