Sub consultas na cláusula select (“select_list”)

1. Introdução

Em aulas anteriores, apresentamos algumas formas de utilização de “subqueries”. Neste texto, usaremos as expressões “subquery”, “sub-select” e “subconsulta” para indicar a mesma coisa. As seguintes situações já foram estudadas em outros roteiros:

a) Teste de pertinência com a cláusula “IN/NOT IN” na cláusula “where”;
b) Teste de relação vazia com a cláusula “EXISTS/NOT EXISTS” na cláusula “where”;
c) Tabela derivada na cláusula FROM;
d) Comparações envolvendo resultados de sub selects na cláusula where com as palavras reservadas “some”/”all”.

2. Restrições na utilização de “sub-query” numa comparação

Em particular no caso do item (d), dissemos que uma comparação de uma coluna ou valor pelo resultado de um comando de sub-select é inviável porque não é possível comparar um valor com um conjunto, visto que um comando de seleção retorna sempre um conjunto que pode vazio, unitário ou com muitas tuplas. A única operação plausível é de pertinência de um valor a um conjunto. Ou seja, são ilícitas e inaplicáveis as seguintes comparações:

$a = \{ a, b, c \}$?
$a > \{ a, b, c \}$?
$a \neq \{ a, b, c \}$?

As verificações de pertinência de um valor a um conjunto de valores são pertinentes:

$a \in \{ a, b, c \}$?
$a \notin \{ a, b, c \}$?

3. Sub-query escalar

As sub-queries que retornam uma única linha com somente um atributo são chamadas de “sub-queries escalares”. Elas equivalem a expressões. Diferentemente de outras situações, somente neste caso, podemos entender que o sistema extrai o valor do conjunto unitário resultante e, consequentemente, pode realizar as operações como se fosse um valor e não um conjunto com um valor.

No item anterior, afirmamos que o teste aplicável entre um valor e um conjunto é o de pertinência; outras comparações não são válidas. No entanto, esclarecemos que a SQL aceita uma exceção: quando o “sub-select” é escalar, pois neste caso, é comparável a uma expressão.

Vamos analisar, neste roteiro, a utilização de “subqueries” na cláusula “select”, ou seja, na “select_list”. Já sabemos que o comando de seleção opera sobre relações e retorna relações, ou seja, opera e retorna conjuntos de tuplas.

4. Relação

No roteiro inicial, apresentamos a definição de uma relação no sentido matemático que é utilizada nos bancos de dados relacionais. O termo “banco de dados relacional” origina-se desta definição e a palavra tabela é o nome popularmente utilizado para “relação. Relembrando a definição de relação: Dados domínios $\boldsymbol{A_1, A_2, A_3, ... A_n}$, dizemos que ***r*** é uma relação sobre os domínios se

$$r = \{ (a_{11} , a_{21} , a_{31} , ... , a_{n1}), (a_{12} , a_{22} , a_{32} , ... , a_{n2}), (a_{13} , a_{23} , a_{33} , ... , a_{n3}), ..., (a_{1m} , a_{2m} , a_{3m} , ... , a_{nm}) \}$$

onde

$$a_{1i} \in A_1 , \ 1 \leq i \leq m$$
$$a_{2i} \in A_2 , \ 1 \leq i \leq m$$
$$a_{3i} \in A_3 , \ 1 \leq i \leq m$$
$$.$$
$$.$$
$$.$$
$$a_{ni} \in A_n , \ 1 \leq i \leq m$$

Redundante afirmar que os domínios são os conjuntos dos valores que os atributos podem assumir.

Sem prejudicar o formalismo, vamos usar o mesmo símbolo para indicar o atributo e o respectivo domínio dos seus valores. Tomando como exemplo, suponha os atributos A, B e C, com os domínios:

$$A = \{ a_1 , a_2 , a_3 ,..., a_m \}$$
$$B = \{ b_1 , b_2 , b_3 ,..., b_n \}$$
$$C = \{ c_1 , c_2 , c_3 ,..., c_p \}$$

O conjunto

$$r = \{ (a_1 , b_1 , c_1), (a_2 , b_2 , c_2), (a_3 , b_3 , c_3) \}$$

é uma relação válida sobre os domínios dos atributos A, B e C porque

$$a_i \in A, \quad b_i \in B \quad e \quad c_i \in C$$

para todo

$$1 \leq i \leq 3.$$

O quadro abaixo mostra a relação ***r*** em formato tabular:

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

Até aqui tudo conhecido. Mas veja o seguinte quadro:

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3
	b_4	
	b_5	

A figura acima mostra três tuplas que poderíamos escrever da seguinte forma:

$t_1 = (a_1 , b_1 , c_1)$
$t_2 = (a_2 , b_2 , c_2)$
$t_3 = (a_3 , \{ b_3 , b_4 , b_5 \} , c_3)$

Estas três tuplas não fazem parte de uma relação segundo a definição matemática porque jamais podemos dizer que um conjunto pertence a outro conjunto. É matematicamente equivocado escrever:

$\{ b_3 , b_4 , b_5 \} \in C$, como seria necessário para que fosse uma relação.

Podemos dizer que

$\{ b_3 , b_4 , b_5 \} \subseteq C$,

mas isto não faz parte da definição de relação.

Logo, a figura não representa uma relação matemática. E rigorosamente falando, não é uma tabela.

5. Sub-select na select_list

Qual o objetivo das reflexões anteriores? O propósito desta discussão é pensar sobre o que podemos especificar na "select_list" (cláusula "select"). O que já sabemos que podem ser incluídos:

- nome de colunas;
- valores constantes;
- funções;
- expressões envolvendo os elementos acima.

Agora, é o momento para a seguinte pergunta:

Uma “sub-query” pode ser incluída na “select_list”? Depois de toda a reflexão acima, a resposta é imediata: Pode ser incluída, desde que seja uma sub-query escalar. Como nas demais situações, pode ser utilizada a operação de renomeação (alias) para o valor resultante e qualquer renomeação da coluna interna à “subquery” é totalmente desnecessária e, às vezes, inaplicável. A “sub consulta” escalar ou equivalente a uma expressão deve aparecer na “select_list” entre parênteses.

6. Exemplos

A seguir, alguns exemplos de utilização desde a mais simples e desnecessária a outros que utilizam o recurso justificadamente.

6.1. Sub Query independente com utilização desnecessária

No exemplo abaixo, a sub consulta, é independente do comando externo, ou seja, não correlacionada. O comando interno apenas acrescenta a coluna com o nome do estado. Este exemplo não tem nenhuma serventia exceto para propósito do assunto que estamos estudando.

```
select codigo,
       nome,
       habitantes,
       qtd_ruas,
       estado,
       (select est_nome
          From tb_estados
         Where est_sigla = 'SP') as "nome do estado"
  from tb_cidades
 where estado = 'SP'
```

6.2. Sub query correlacionada, mas desnecessária

Situações mais interessantes ocorrem com “subqueries” correlacionadas, mas neste exemplo específico não há a necessidade do recurso de “sub query” na “select_list”.

```
select codigo,
       nome,
       habitantes,
       qtd_ruas,
       estado,
       (select est_nome
          from tb_estados as est
         where est_sigla = cid.estado) as "nome do estado"
  from tb_cidades as cid
 order by nome
```

Neste caso específico, poderia ser resolvido de forma mais simples e deve ser o método preferido pelo simples fato de não complicar o que é simples.

```
select codigo,
       nome,
       habitantes,
       qtd_ruas,
       estado,
       est.est_nome as "nome do estado"
  from tb_cidades as cid,
       tb_estados as est
 where est.est_sigla = cid.estado
 order by nome
```

Ou de forma equivalente, utilizando a operação de junção:

```
select codigo,
       nome,
       habitantes,
       qtd_ruas,
       estado,
       est.est_nome as "nome do estado"
  from tb_cidades as cid
  join tb_estados as est
    on est.est_sigla = cid.estado
 order by nome
```

6.3. Utilização indicada

Já utilizamos, em roteiros anteriores, a tabela de nome “tb_cidades” que contem dados de algumas cidades que, por algum critério, escolhemos como relevantes para serem registradas no nosso banco de dados de tal modo que existem estados sem cidades. O comando seguinte apresenta os estados com a correspondente quantidade de cidades, incluindo os que não têm nenhuma cidade registrada.

```
select est_sigla,
       est_nome,
       (select count(*)
          from tb_cidades as cid
         where cid.estado = est.est_sigla) as "Quantidade de cidades"
  from tb_estados as est
 order by est_nome
```

Se utilizarmos função de agregação com agrupamento, o resultado será semelhante, mas faltarão no resultado os estados que não tem cidades registradas.

```
select est_sigla,
       est_nome,
       count(*) as "total de cidades"
  from tb_cidades as cid
  join
       tb_estados as est
    on cid.estado = est.est_sigla
 group by est_sigla,
          est_nome
 order by est_nome
```

À primeira vista, vocês poderão dizer que podemos resolver esta pequena distorção com “outer join” que será assunto de aula futura. No entanto, isto não resolve de imediato porque os estados sem cidades serão apresentados como se possuíssem UMA cidade. No entanto, se pensarmos um pouco mais nesta última alternativa, aproveitando o fato de que a função de agregação “count(*)” conta todas as linhas, mas count(coluna) ignora as linhas com valores nulos na coluna considerada, temos como resolver esta questão. De qualquer maneira, o comportamento das funções de agregação com respeito a valores nulos e conjuntos vazios pode variar de forma muito sutil e sua utilização requer cuidados. Então, podemos dizer que encontramos uma situação em que a “subquery” na “select_list” é útil.

6.4. Utilização indicada com vários “subquery” na “select_list”

No exemplo abaixo, apresentamos uma situação em que são necessárias duas sub-queries escalares que envolvem tabelas diferentes no mesmo comando de modo que um simples join não resolveria o problema. O comando apresenta os dados das disciplinas com a quantidade de alunos que as cursam e também a quantidade de professores que são habilitados a ministrá-las. O resultado inclui as disciplinas sem alunos e sem professores habilitados.

```
select dis_sigla,
       dis_nome,
       dis_qt_creditos,
       (select count(*) from tb_alunos_disc as ad
               where ad.dis_sigla = disc.dis_sigla) as qtd_alunos,
       (select count(*) from tb_prof_habilitados_disc as phd
               where phd.dis_sigla = disc.dis_sigla) as qtd_professores
   from tb_disciplinas as disc
  order by dis_nome
```

7. Sub Query não escalar

Como a “sub-query” precisa ser escalar na definição inicialmente apresentada, as modificações abaixo são inconsistentes. No primeiro caso, a “sub-query” tem mais que um atributo e no segundo, como os estados podem ter mais que uma cidade, várias tuplas podem ser encontradas e, neste caso, o erro acontecerá em tempo de execução.

```
select est_sigla,
       est_nome,
       (select count(*), max(habitantes)
          from tb_cidades as cid
         where cid.estado = est.est_sigla) as "Quantidade de cidades"
  from tb_estados as est
 order by est_nome
```

```
select est_sigla,
       est_nome,
       (select nome,
          from tb_cidades as cid
         where cid.estado = est.est_sigla) as "Cidade"
  from tb_estados as est
 order by est_nome
```

Prof. Satoshi Nagayama.