Consultas com subqueries

- Principais características das subqueries;
- Subqueries introduzidas com IN e NOT IN;
- Subqueries introduzidas com sinal de igualdade (=);
- Subqueries correlacionadas;
- Diferenças entre subqueries e associações;
- Diferenças entre subqueries e tabelas temporárias.

Esta Leitura Complementar refere-se ao conteúdo das Aulas 24 e 25.

Editora IMPACTA

1.1. Introdução

Uma consulta aninhada em uma instrução **SELECT**, **INSERT**, **DELETE** ou **UPDATE** é chamada de subquery (subconsulta). Isso ocorre quando usamos um **SELECT** dentro de um **SELECT**, **INSERT**, **UPDATE** ou **DELETE**. Também é comum chamar uma subquery de query interna. Já a instrução em que está inserida a subquery pode ser chamada de query externa.

O limite máximo de aninhamento de uma subquery é de 32 níveis, limite este que varia de acordo com a complexidade das outras instruções que compõem a consulta e com a quantidade de memória disponível.

1.2. Principais características das subqueries

A seguir, temos a descrição das principais características das subqueries:

- As subqueries podem ser escalares (retornam apenas uma linha) ou tabulares (retornam linhas e colunas);
- **É possível** obter apenas uma coluna por subquery;
- Uma subquery, que pode ser incluída dentro de outra subquery, deve estar entre parênteses, o que a diferenciará da consulta principal;
- Em instruções **SELECT**, **UPDATE**, **INSERT** e **DELETE**, uma subquery é utilizada nos mesmos locais em que poderiam ser utilizadas expressões;
- Pelo fato de podermos trabalhar com consultas estruturadas, as subqueries permitem que partes de um comando sejam separadas das demais partes;
- Se uma instrução é permitida em um local, este local aceita a utilização de uma subquery;
- Alguns tipos de dados não podem ser utilizados na lista de seleção de uma subquery. São eles: **nvarchar(max)**, **varchar(max)** e **varbinary(max)**;
- Um único valor será retornado ao utilizarmos o sinal de igualdade (=) no início da subquery;
- As palavras-chave **ALL**, **ANY** e **SOME** podem ser utilizadas para modificar operadores de comparação que introduzem uma subquery. Podemos fazer as seguintes considerações a respeito delas:
    - **ALL** realiza uma comparação entre um valor escalar e um conjunto de valores de uma coluna. Então, retornará **TRUE** nos casos em que a comparação for verdadeira para todos os pares;
    - **SOME** (padrão ISO que equivale a **ANY**) e **ANY** realizam uma comparação entre um valor escalar e um conjunto de valores de uma coluna. Então, retornarão **TRUE** nos casos em que a comparação for verdadeira para qualquer um dos pares.

- Quando utilizamos um operador de comparação (=, < >, >, > =, <, ! >, ! < ou < =) para introduzir uma subquery, sua lista de seleção poderá incluir apenas um nome de coluna ou expressão, a não ser que utilizemos **IN** na lista ou **EXISTS** no **SELECT**;

- As cláusulas **GROUP BY** e **HAVING** não podem ser utilizadas em subqueries introduzidas por um operador de comparação que não seja seguido pelas palavras-chave **ANY** ou **ALL**;

- A utilização da cláusula **ORDER BY** só é possível caso a cláusula **TOP** seja especificada;

- Alternativamente, é possível formular muitas instruções do Transact-SQL com subqueries como associações;

- O nome de coluna que, ocasionalmente, estiver presente na cláusula **WHERE** de uma query externa deve ser associável com a coluna da lista de seleção da subquery;

- A qualificação dos nomes de colunas de uma instrução é feita pela tabela referenciada na cláusula **FROM**;

- Subqueries que incluem **GROUP BY** não aceitam a utilização de **DISTINCT**;

- Uma view não pode ser atualizada caso ela tenha sido criada com uma subquery;

- Uma subquery aninhada na instrução **SELECT** externa é formada por uma cláusula **FROM** regular com um ou mais nomes de view ou tabela, por uma consulta **SELECT** regular junto dos componentes da lista de seleção regular e pelas cláusulas opcionais **WHERE**, **HAVING** e **GROUP BY**;

- Subqueries podem ser utilizadas para realizar testes de existência de linhas. Nesse caso, é adotado o operador **EXISTS**;

- Por oferecer diversas formas de obter resultados, as subqueries eliminam a necessidade de utilização das cláusulas **JOIN** e **UNION** de maior complexidade;

- Uma instrução que possui uma subquery não apresenta muitas diferenças de performance em relação a uma versão semanticamente semelhante que não possui a subquery. No entanto, uma **JOIN** apresenta melhor desempenho nas situações em que é necessário realizar testes de existência;

- As colunas de uma tabela não poderão ser incluídas na saída, ou seja, na lista de seleção da query externa, caso essa tabela apareça apenas em uma subquery e não na query externa.

A seguir, temos os formatos normalmente apresentados pelas instruções que possuem uma subquery:

```
WHERE expressao [NOT] IN (subquery);
```

```
WHERE expressao operador_comparacao [ANY | ALL] (subquery);
```

```
WHERE [NOT] EXISTS (subquery).
```

Veja um exemplo de subquery que verifica a existência de clientes que compraram no mês de janeiro de 2014:

```
SELECT * FROM TB_CLIENTE
WHERE EXISTS (SELECT * FROM TB_PEDIDO
    WHERE CODCLI = TB_CLIENTE.CODCLI AND
          DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31');
-- OU
SELECT * FROM TB_CLIENTE
WHERE CODCLI IN (SELECT CODCLI FROM TB_PEDIDO
     WHERE DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31');
```

No próximo exemplo, é verificada a existência de clientes que não compraram em janeiro de 2014:

```
SELECT * FROM TB_CLIENTE
WHERE NOT EXISTS (SELECT * FROM TB_PEDIDO
   WHERE CODCLI = TB_CLIENTE.CODCLI AND
         DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31');
-- OU
SELECT * FROM TB_CLIENTE
WHERE CODCLI NOT IN (SELECT CODCLI FROM TB_PEDIDO
   WHERE DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31');
```

1.3. Subqueries introduzidas com IN e NOT IN

Uma subquery terá como resultado uma lista de zero ou mais valores caso tenha sido introduzida com a utilização de **IN** ou **NOT IN**. O resultado, então, será utilizado pela query externa.

Os exemplos adiante demonstram subqueries introduzidas com **IN** e **NOT IN**:

- **Exemplo 1**

```
-- Lista de empregados cujo cargo tenha salário inicial
-- inferior a 5000
SELECT * FROM TB_EMPREGADO
WHERE COD_CARGO IN (SELECT COD_CARGO FROM TB_CARGO
                    WHERE SALARIO_INIC < 5000);
```

- **Exemplo 2**

```
-- Lista de departamentos em que não existe nenhum
-- funcionário cadastrado
SELECT * FROM TB_DEPARTAMENTO
WHERE COD_DEPTO NOT IN
    (SELECT DISTINCT COD_DEPTO FROM TB_EMPREGADO
     WHERE COD_DEPTO IS NOT NULL)
-- O mesmo que
SELECT
  E.CODFUN, E.NOME, E.COD_DEPTO, E.COD_CARGO, D.COD_DEPTO, D.DEPTO
FROM TB_EMPREGADO E RIGHT JOIN TB_DEPARTAMENTO D ON E.COD_DEPTO =
D.COD_DEPTO
WHERE E.COD_DEPTO IS NULL
```

- **Exemplo 3**

```
-- Lista de cargos em que não existe nenhum
-- funcionário cadastrado
SELECT * FROM TB_CARGO
WHERE COD_CARGO NOT IN
(SELECT DISTINCT COD_CARGO FROM TB_EMPREGADO
 WHERE COD_CARGO IS NOT NULL)
-- O mesmo que
SELECT
   C.COD_CARGO, C.CARGO
FROM TB_EMPREGADO E RIGHT JOIN TB_CARGO C ON E.COD_CARGO = C.COD_
CARGO
WHERE E.COD_CARGO IS NULL
```

1.4. Subqueries introduzidas com sinal de igualdade (=)

Veja exemplos de como utilizar o sinal de igualdade (=) para inserir subqueries:

- **Exemplo 1**

```
-- Funcionário(s) que ganha(m) menos
SELECT * FROM TB_EMPREGADO
WHERE SALARIO = (SELECT MIN(SALARIO) FROM TB_EMPREGADO)
-- o mesmo que
SELECT TOP 1 WITH TIES * FROM TB_EMPREGADO
WHERE SALARIO IS NOT NULL
ORDER BY SALARIO
```

- **Exemplo 2**

```
-- Funcionário mais novo na empresa
SELECT * FROM TB_EMPREGADO
WHERE DATA_ADMISSAO =
      (SELECT MAX(DATA_ADMISSAO) FROM TB_EMPREGADO);

-- O mesmo que
SELECT TOP 1 WITH TIES * FROM TB_EMPREGADO
ORDER BY DATA_ADMISSAO DESC;
```

1.5. Subqueries correlacionadas

Quando uma subquery possui referência a uma ou mais colunas da query externa, ela é chamada de subquery correlacionada. É uma subquery repetitiva, pois é executada uma vez para cada linha da query externa. Assim, os valores das subqueries correlacionadas dependem da query externa, o que significa que, para construir uma subquery desse tipo, será necessário criar tanto a query interna como a externa.

Também é possível que subqueries correlacionadas incluam, na cláusula **FROM**, funções definidas pelo usuário, as quais retornam valores de tipo de dado **table**. Para isso, basta que colunas de uma tabela na query externa sejam referenciadas como argumento de uma função desse tipo. Então, será feita a avaliação dessa função de acordo com a subquery para cada linha da query externa.

Veja o exemplo a seguir, que grava, no campo **SALARIO** de cada funcionário, o valor de salário inicial contido na tabela de cargos:

```
UPDATE TB_EMPREGADO SET SALARIO = (SELECT SALARIO_INIC
FROM TB_CARGO
WHERE COD_CARGO = TB_EMPREGADO.COD_CARGO);
```

1.5.1. Subqueries correlacionadas com EXISTS

Subqueries correlacionadas introduzidas com a cláusula **EXISTS** não retornam dados, mas apenas **TRUE** ou **FALSE**. Sua função é executar um teste de existência de linhas, portanto, se houver qualquer linha em uma subquery, será retornado **TRUE**.

Sobre **EXISTS**, é importante atentarmos para os seguintes aspectos:

- Antes de **EXISTS** não deve haver nome de coluna, constante ou expressão;
- Quando **EXISTS** introduz uma subquery, sua lista de seleção será, normalmente, um asterisco.

Por meio do código a seguir, é possível saber se temos clientes que não realizaram compra no mês de janeiro de 2014:

```
SELECT * FROM TB_CLIENTE
WHERE NOT EXISTS (SELECT * FROM TB_PEDIDO
    WHERE CODCLI = TB_CLIENTE.CODCLI AND
          DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31');
```

1.6. Diferenças entre subqueries e associações

Ao comparar subqueries e associações (**JOINS**), é possível constatar que as associações são mais indicadas para verificação de existência, pois apresentam desempenho melhor nesses casos. Também podemos verificar que, ao contrário das subqueries, as associações não atuam em listas com um operador de comparação modificado por **ANY** ou **ALL**, ou em listas que tenham sido introduzidas com **IN** ou **EXISTS**.

Em alguns casos, pode ser que lidemos com questões muito complexas para serem respondidas com associações, então, será mais indicado usar subqueries. Isso porque a visualização do aninhamento e da organização da query é mais simples em uma subquery, enquanto que, em uma consulta com diversas associações, a visualização pode ser complicada. Além disso, nem sempre as associações podem reproduzir os efeitos de uma subquery.

O código a seguir utiliza **JOIN** para calcular o total vendido por cada vendedor no período de janeiro de 2014 e a porcentagem de vendas em relação ao total de vendas realizadas no mesmo mês:

```
SELECT P.CODVEN, V.NOME,
  SUM(P.VLR_TOTAL) AS TOT_VENDIDO,
  100 * SUM(P.VLR_TOTAL) / (SELECT SUM(VLR_TOTAL)
FROM TB_PEDIDO
     WHERE DATA_EMISSAO BETWEEN '2014.1.1' AND  '2014.1.31') AS POR-
CENTAGEM
FROM TB_PEDIDO P JOIN TB_VENDEDOR V ON P.CODVEN = V.CODVEN
WHERE P.DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31'
GROUP BY P.CODVEN, V.NOME;
```

Já o código a seguir utiliza subqueries para calcular, para cada departamento, o total de salários dos funcionários sindicalizados e o total de salários dos não sindicalizados:

```
SELECT COD_DEPTO,
 (SELECT SUM(E.SALARIO) FROM TB_EMPREGADO E
  WHERE E.SINDICALIZADO = 'S' AND
        E.COD_DEPTO = TB_EMPREGADO.COD_DEPTO) AS TOT_SALARIO_SIND,
 (SELECT SUM(E.SALARIO) FROM TB_EMPREGADO E
  WHERE E.SINDICALIZADO = 'N' AND
        E.COD_DEPTO = TB_EMPREGADO.COD_DEPTO) AS TOT_SALARIO_NAO_
SIND
FROM TB_EMPREGADO
GROUP BY COD_DEPTO;
```

1.7. Diferenças entre subqueries e tabelas temporárias

Embora as tabelas temporárias sejam parecidas com as permanentes, elas são armazenadas em **tempdb** e excluídas automaticamente após terem sido utilizadas.

As tabelas temporárias locais apresentam, antes do nome, o símbolo # e são visíveis somente durante a conexão atual. Quando o usuário desconecta-se da instância do SQL Server, ela é excluída.

Já as tabelas temporárias globais apresentam, antes do nome, dois símbolos ## e são visíveis para todos os usuários. Uma tabela desse tipo será excluída apenas quando todos os usuários que a referenciam se desconectarem da instância do SQL Server.

A escolha da utilização de tabelas temporárias ou de subqueries dependerá de cada situação e de aspectos como desempenho do sistema e até mesmo das preferências pessoais de cada usuário. O fato é que, por conta das diferenças existentes entre elas, o uso de uma, para uma situação específica, acaba sendo mais indicado do que o emprego de outra.

Assim, quando temos bastante RAM, as subqueries são preferíveis, pois ocorrem na memória. Já as tabelas temporárias, como necessitam dos recursos disponibilizados pelo disco rígido para serem executadas, são indicadas nas situações em que o(s) servidor(es) do banco de dados apresenta(m) bastante espaço no disco rígido.

Há, ainda, uma importante diferença entre tabela temporária e subquery: normalmente, esta última é mais fácil de manter. No entanto, se a subquery for muito complexa, a melhor medida a ser tomada pode ser fragmentá-la em diversas tabelas temporárias, criando, assim, blocos de dados de tamanho menor.

Veja o exemplo a seguir, que utiliza subqueries para retornar os pedidos da vendedora **LEIA** para clientes de **SP** que não compraram em janeiro de 2014, mas compraram em dezembro de 2013:

```
SELECT * FROM TB_PEDIDO
WHERE CODVEN IN (SELECT CODVEN FROM TB_VENDEDOR WHERE NOME = 'LEIA')
AND CODCLI IN (
                SELECT CODCLI FROM TB_CLIENTE
                WHERE CODCLI NOT IN (SELECT CODCLI FROM TB_PEDIDO
                                     WHERE DATA_EMISSAO BETWEEN
                                    '2014.1.1' AND '2014.1.31') AND
                      CODCLI IN (SELECT CODCLI FROM TB_PEDIDO
                                 WHERE DATA_EMISSAO BETWEEN
'2013.12.1'
                                       AND '2013.12.31')
                AND ESTADO = 'SP' );
```

Já no próximo código, em vez de subqueries, utilizamos tabelas temporárias para obter o mesmo resultado do código anterior:

```
-- Tabela temporária 1 - Código da vendedora LEIA
SELECT CODVEN INTO #VEND_LEIA FROM TB_VENDEDOR WHERE NOME = 'LEIA';

-- Tabela temporária 2 - Clientes que compraram em Jan/2014
SELECT CODCLI INTO #CLI_COM_PED_JAN_2014 FROM TB_PEDIDO
WHERE DATA_EMISSAO BETWEEN '2014.1.1' AND '2014.1.31';

-- Tabela temporária 3 - Clientes que compraram em Dez/2013
SELECT CODCLI INTO #CLI_COM_PED_DEZ_2013 FROM TB_PEDIDO
WHERE DATA_EMISSAO BETWEEN '2013.12.1' AND '2013.12.31';

-- Tabela temporária 4 - Clientes de SP que compraram
-- em Dez/2013, mas não compraram em Jan de 2014
SELECT CODCLI INTO #CLI_FINAL FROM TB_CLIENTE
WHERE CODCLI NOT IN (SELECT CODCLI FROM #CLI_COM_PED_JAN_2014)
      AND
      CODCLI IN (SELECT CODCLI FROM #CLI_COM_PED_DEZ_2013)
      AND
      ESTADO = 'SP';

-- SELECT de TB_PEDIDO
SELECT * FROM TB_PEDIDO
WHERE CODVEN IN (SELECT CODVEN FROM #VEND_LEIA)
      AND
      CODCLI IN (SELECT CODCLI FROM #CLI_FINAL);
```

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes da leitura.

- Uma consulta aninhada em uma instrução **SELECT**, **INSERT**, **DELETE** ou **UPDATE** é denominada subquery (subconsulta). As subqueries são também referidas como queries internas. Já a instrução em que está inserida a subquery pode ser chamada de query externa;

- Vejamos algumas das diversas características das subqueries: podem ser escalares (retornam apenas uma linha) ou tabulares (retornam linhas e colunas). Elas, que podem ser incluídas dentro de outras subqueries, devem estar entre parênteses, o que as diferenciará da consulta principal;

- Uma subquery retornará uma lista de zero ou mais valores caso tenha sido introduzida com a utilização de **IN** ou **NOT IN**. O resultado, então, será utilizado pela query externa;

- O sinal de igualdade (=) pode ser utilizado para inserir subqueries;

- Quando uma subquery possui referência a uma ou mais colunas da query externa, ela é chamada de subquery correlacionada. Trata-se de uma subquery repetitiva, pois é executada uma vez para cada linha da query externa. Desta forma, os valores das subqueries correlacionadas dependem da query externa, o que significa que, para construir uma subquery desse tipo, será necessário criar tanto a query interna como a externa;

- Ao comparar subqueries e associações (**JOINS**), é possível constatar que as associações são mais indicadas para verificação de existência, pois apresentam desempenho melhor nesses casos;

- A visualização do aninhamento e da organização da query é mais simples em uma subquery, enquanto que, em uma consulta com diversas associações, a visualização pode ser complicada;

- Tabelas temporárias são armazenadas no **tempdb** e excluídas automaticamente após terem sido utilizadas;

- As tabelas temporárias locais apresentam, antes do nome, o símbolo # e são visíveis somente durante a conexão atual. Quando o usuário desconecta-se da instância do SQL Server, ela é excluída. Já as tabelas temporárias globais apresentam, antes do nome, dois símbolos ## e são visíveis para todos os usuários. Uma tabela desse tipo será excluída apenas quando todos os usuários que a referenciam se desconectarem da instância do SQL Server;

- A escolha da utilização de tabelas temporárias ou subqueries dependerá de cada situação e de aspectos como desempenho do sistema e até mesmo das preferências pessoais de cada usuário.