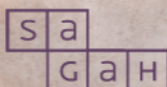


MODELAGEM E DESENVOLVIMENTO DE BANCO DE DADOS

Pedro Henrique Chagas Freitas



SOLUÇÕES
EDUCACIONAIS
INTEGRADAS



Consultas avançadas com *joins* e *subqueries*

Objetivos de aprendizagem

- Identificar uma consulta utilizando *joins* e *subqueries*.
- Exemplificar uma consulta utilizando *joins* e *subqueries*.
- Implementar uma consulta utilizando *joins* e *subqueries*.

Introdução

Os *joins* e *subqueries* são instruções utilizadas em conjunto ou de forma separada para elaborar consultas avançadas em bancos de dados.

Neste capítulo, você vai estudar a utilização dos *joins* e *subqueries* para elaboração de consultas avançadas, além de exemplos que utilizam *joins* e *subqueries*. Você vai estudar, também, a implementação em consultas avançadas com *joins* e *subqueries*.

Conceituando a utilização de consultas avançadas com *joins* e *subqueries*

Os *joins*, assim como as *subqueries*, são utilizados com comandos SQL para consulta, no caso os **SELECTs**, para realizar consultas avançadas nas tabelas dos bancos de dados, tendo em vista que, quando o dado necessário é pertencente a uma tabela, temos uma consulta simples. Por exemplo:

```
SELECT nome_campo FROM nome_tabela WHERE condição_esperada
```

Nesse caso, temos uma consulta simples, porque estamos referenciando um dado (nome_campo) que está em uma tabela (nome_tabela) e atribuímos uma condição (WHERE) no retorno dessa consulta. Observe:

```
SELECT Nome_Time, Região_Time FROM Times WHERE Nome_Time  
= 'Flamengo'
```

O surgimento dos *joins* e *subqueries* permeia a utilização do comando **DQL: SELECT**, levando-se em conta que, quando realizamos uma consulta, não necessariamente essa consulta será uma consulta simples; portanto, poderá ser necessário cruzar dados que estão em tabelas diferentes.

Quando realizamos o cruzamento de dados em uma consulta, temos uma junção de tabelas para originar o dado esperado pela consulta, ou seja, temos um *join* (junção). Quando realizamos duas consultas, uma dentro da outra, temos uma *subquery*, ou subconsulta, que também parte da ideia de que a consulta que será realizada (**SELECT**) deverá retornar dados de tabelas diferentes entre si. O comando **DQL SELECT** permite ao usuário de um banco de dados realizar uma consulta por meio de uma especificação de parâmetro, também conhecido como *query*.

A especificação da *query* apresenta a descrição do resultado de retorno desejado após a consulta. É importante observar que a derivação dessa consulta é um retorno esperado, isto é, quando realizamos uma consulta em uma base de dados, estamos realizando uma consulta de algo em algum lugar; logo, os parâmetros que envolvem a localização dos dados precisam apresentar exatidão conforme a complexidade da consulta.

Os *joins* combinados com os *selects* fazem surgir as consultas avançadas por meio do que conhecemos como **teoria dos conjuntos**, que referencia a junção ou cruzamento entre tabelas. Observe a Figura 1 a seguir.

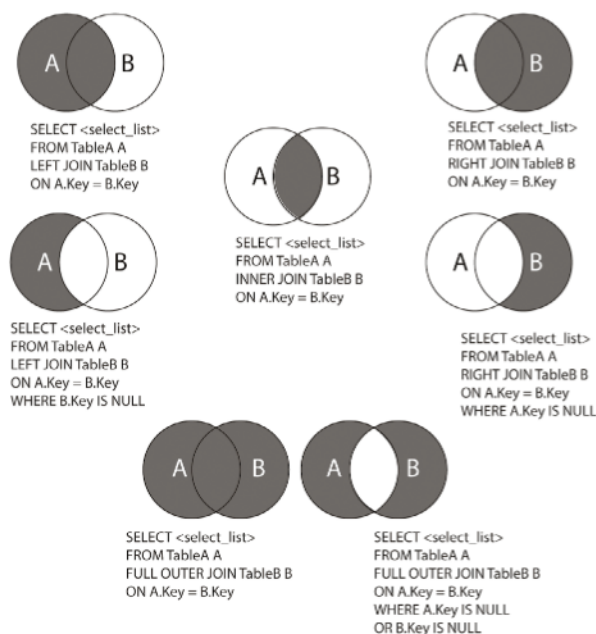


Figura 1. Teoria dos conjuntos.

Note que estamos fazendo a junção de conjuntos. Essa técnica se configura por meio dos *joins* dentro do contexto de banco de dados e é altamente necessária frente a algumas consultas que são realizadas, ou seja, caso não utilizássemos os *joins* ou as *subqueries*, algumas consultas não poderiam ser realizadas, dada a sua complexidade.

Dessa forma, os *joins* trouxeram ganhos à realização das consultas com a instrução **SQL SELECT**, o que, por sua vez, colaborou para difundir o SQL como a linguagem mais utilizada atualmente para a intercomunicação com bancos de dados.



Saiba mais

Para utilização de dois *joins*, são necessárias pelo menos duas tabelas, entre as quais deve haver algum nível de relacionamento para que se possa cruzar os dados. Por exemplo: podemos ter um campo comum entre as tabelas.

Já as *subqueries* são subconsultas ou consultas dentro de consultas. Na prática, é realizada uma busca dentro de outra busca com a instrução SQL `SELECT` para consulta. Assim, temos, então, um `SELECT` dentro de outro `SELECT`, com `SELECTs` internos.

A *subquery* é uma instrução `SELECT` que está condicionada dentro de outras instruções SQL, ou seja, uma subconsulta para criar um retorno de um resultado que, por meio de uma consulta simples, teria alta complexidade ou até mesmo impossibilidade. Utilizam-se as *subqueries* introduzidas em cláusulas `WHERE`, `FROM`, `HAVING` ou no próprio `SELECT`. Por exemplo, podemos usar uma instrução de uma subconsulta na condição de pesquisa com uma cláusula `WHERE`:

```
SELECT NomeCD, Preco FROM Loja
WHERE Preco = (SELECT MAX (Preco) FROM Loja)
```

Nesse caso, estamos fazendo uma consulta dentro de outra consulta, para listar todos os DVDs com o maior preço. Na *subquery*, buscamos qual o valor máximo encontrado na tabela loja, então buscamos todos os NOMEs e PREÇOS dos DVDs que contém preço igual ao máximo encontrado pela *subquery*.

Exemplificando a utilização de *joins* e *subqueries*

Considere, por exemplo, as tabelas a seguir:

TABELA_A

|CODIGO | NOME |

| 1 | UM |

| 2 | DOIS |

| 3 | TRES |

| 4 | QUATRO |

| 5 | CINCO |

TABELA_B

LANCA	CODIGO	VALOR

1	1	1.000
2	1	2.000
3	1	5.000
4	2	4.000
5	2	9.000
6	3	7.000
7	5	4.000
8	8	7.000

Vamos realizar, então, alguns *joins*. Para realizar um INNER JOIN, faríamos:

```
SELECT A.NOME "A.NOME",  
       B.VALOR "B.VALOR"  
FROM TABELA_A A  
     INNER JOIN TABELA_B B ON B.CODIGO = A.CODIGO
```

O resultado seria:

A.NOME	B.VALOR

1. UM	1.000
2. UM	2.000
3. UM	5.000
4. DOIS	4.000
5. DOIS	9.000
6. TRES	7.000
7. CINCO	4.000

Nesse exemplo, temos como resultado do **INNER JOIN** o retorno somente das linhas que são comuns nas duas tabelas.

Um exemplo de utilização de *subquery* poderia ser para verificar a média de preço: temos como base uma tabela de um gravador de CDs, por exemplo, na qual queremos retornar o nome e o preço no primeiro **SELECT** mostrando só os resultados em que o preço é maior que o preço médio.

Teríamos, portanto:

```
SELECT CODIGO_GRAVADORA, NOME_CD, PRECO_VENDA FROM CD
WHERE PRECO_VENDA > (SELECT AVG(PRECO_VENDA) FROM CD
WHERE CODIGO_GRAVADORA = a.CODIGO_GRAVADORA);
```

Supondo que a média fosse 10, teríamos o código do CD, o nome do CD e o preço médio de venda, como mostra a Figura 2:

Codigo_Gravadora	Nome_CD	Preco_Venda
4567	Viva la vida	12
5432	The night	13,5
13577	Dias atrás	15
321314	Bela rosa	18

Figura 2. Exemplo de *subquery*.

Implementando *joins* e *subqueries*

Considere, por exemplo, duas tabelas: tabela X e tabela Y. Pode-se realizar um *join* para unir tabelas por meio de um campo em comum; no caso, poderia ser o campo **nome**.

```
CREATE TABLE tabelaX (
  Nome varchar ( ) NULL
)
CREATE TABLE tabelaY (
  Nome varchar ( ) NULL
)
```

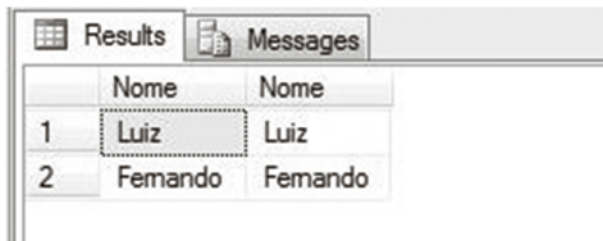
Agora vamos inserir dados nessas tabelas para realizarmos os *joins* (junções):


```
INSERT INTO tabelaX VALUES ('Fernanda')
INSERT INTO tabelaX VALUES ('Josefa')
INSERT INTO tabelaX VALUES ('Luiz')
INSERT INTO tabelaX VALUES ('Fernando')
INSERT INTO tabelaY VALUES ('Carlos')
INSERT INTO tabelaY VALUES ('Manoel')
INSERT INTO tabelaY VALUES ('Luiz')
INSERT INTO tabelaY VALUES ('Fernando')
```

Após inseridos os valores, vamos realizar os *joins*. Caso quiséssemos os registros em comum as duas tabelas, utilizaríamos o *inner join*, como você pode observar a seguir:

```
SELECT a.Nome, b.Nome
FROM tabelaX as A
INNER JOIN tabelaY as B
on a.Nome = b.Nome
```

Teremos, então, dois resultados de nomes (registros) em comum nas duas tabelas, como você pode ver na Figura 3.



	Nome	Nome
1	Luiz	Luiz
2	Fernando	Fernando

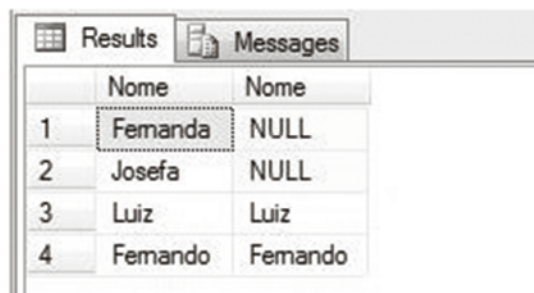
Figura 3. Exemplo de registros com *inner join*.

Temos, também, o *left join*, que é utilizado para articular e apresentar como resultado todos os registros que estão na tabela X (mesmo que não estejam na tabela Y) e os registrados na tabela Y em comum com a tabela X.

Fariamos o seguinte:


```
SELECT a.Nome, b.Nome  
FROM tabelaX as A  
LEFT JOIN tabelaY as B  
on a.Nome = b.Nome
```

Nesse caso, atribuiríamos que a consulta utilizaria as variáveis **a.Nome** e **b.Nome** para realizar a junção, como no exemplo anterior, mas utilizando, agora, o *left join*. A Figura 4 apresenta o resultado.



	Nome	Nome
1	Femanda	NULL
2	Josefa	NULL
3	Luiz	Luiz
4	Fernando	Fernando

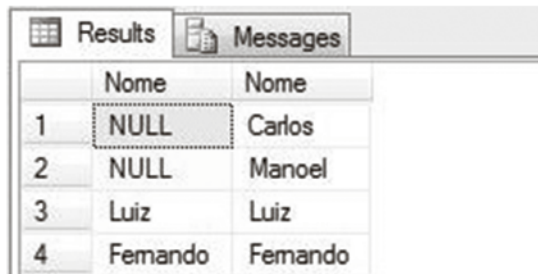
Figura 4. Exemplo de registros com *left join*.

Como temos o *left join*, temos também o *right join*. Neste caso, teremos como resultado todos os registros da tabela Y, mesmo os que não estejam na tabela X, e os registros da tabela X que são comuns à tabela Y.

Fariamos:

```
SELECT a.Nome, b.Nome  
FROM tabelaX as A  
RIGHT JOIN tabelaY as B  
on a.Nome = b.Nome
```

Observe o resultado na Figura 5.



	Nome	Nome
1	NULL	Carlos
2	NULL	Manoel
3	Luiz	Luiz
4	Fernando	Fernando

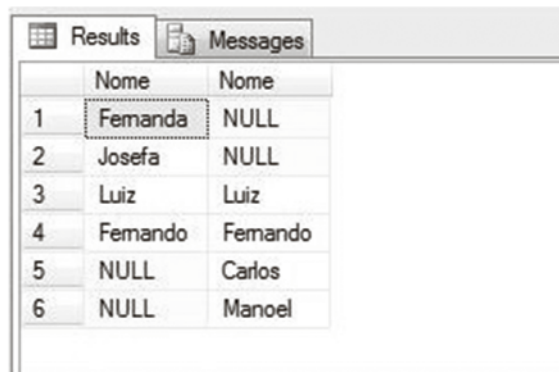
Figura 5. Exemplo de registros com *right join*.

Temos também o *outer join* ou *full outer join*, que tem como objetivo mostrar todos os registros que estão na tabela X e na tabela Y.

Fariamos:

```
SELECT a.Nome, b.Nome  
FROM tabelaX as A  
FULL OUTER JOIN tabelaY as B  
on a.Nome = b.Nome
```

Observe o resultado na Figura 6.



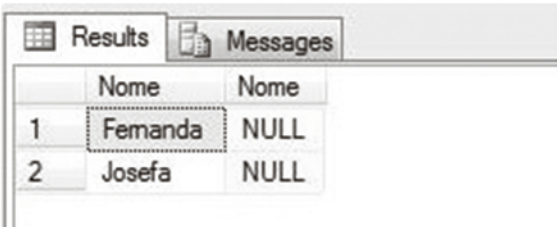
	Nome	Nome
1	Fernanda	NULL
2	Josefa	NULL
3	Luiz	Luiz
4	Fernando	Fernando
5	NULL	Carlos
6	NULL	Manoel

Figura 6. Exemplo de registros com *full outer join*.

Temos o *left excluding join*, que retorna todos os registros que estão da tabela X e que não estão na tabela Y.

```
SELECT a.Nome, b.Nome  
FROM tabelaX as A  
LEFT JOIN tabelaY as B  
on a.Nome = b.Nome  
WHERE b.Nome is null
```

Observe o resultado na Figura 7.



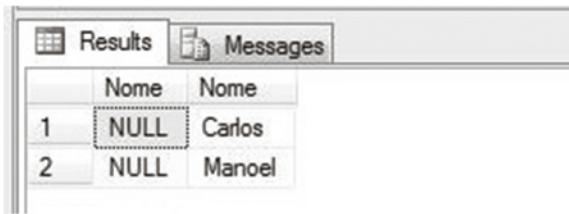
	Nome	Nome
1	Femanda	NULL
2	Josefa	NULL

Figura 7. Exemplo de registros com *left excluding join*.

Como temos o *left excluding join*, temos também o *right excluding join*, que é utilizado para retornar todos os registros que estão na tabela Y e que não estejam na tabela X.

```
SELECT a.Nome, b.Nome  
FROM tabelaX as A  
RIGHT JOIN tabelaY as B  
on a.Nome = b.Nome  
WHERE a.Nome is null
```

Observe o resultado na Figura 8.



The screenshot shows a database interface with a 'Results' tab. It displays a table with two columns, both labeled 'Nome', and two rows. The first row has '1' in the first column and 'Carlos' in the second. The second row has '2' in the first column and 'Manoel' in the second. The first column values are highlighted with a dashed border.

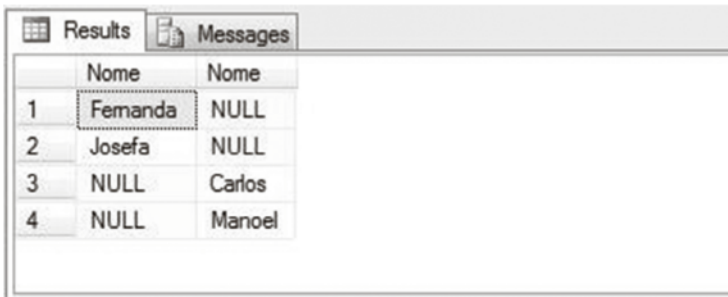
	Nome	Nome
1	NULL	Carlos
2	NULL	Manoel

Figura 8. Exemplo de registros com *right excluding join*.

Por fim, temos o *outer excluding join*. Neste caso, teremos como resultado todos os registros que estão na tabela Y, mas que não estejam na tabela X, e todos os registros que estão na tabela X, mas que não estão na tabela Y.

```
SELECT a.Nome, b.Nome
FROM tabelaX as A
FULL OUTER JOIN tabelaY as B
on a.Nome = b.Nome
WHERE a.Nome is null or b.Nome is null
```

Observe o resultado na Figura 9.



The screenshot shows a database interface with a 'Results' tab. It displays a table with two columns, both labeled 'Nome', and four rows. The first row has '1' in the first column and 'NULL' in the second. The second row has '2' in the first column and 'NULL' in the second. The third row has '3' in the first column and 'Carlos' in the second. The fourth row has '4' in the first column and 'Manoel' in the second. The first column values are highlighted with a dashed border.

	Nome	Nome
1	Fernanda	NULL
2	Josefa	NULL
3	NULL	Carlos
4	NULL	Manoel

Figura 9. Exemplo de registros com *outer excluding join*.

Vejamos então um exemplo de utilização de *subqueries* para realizar uma consulta dentro de outra consulta:

```
SELECT DISTINCT NomeFornecedor,  
(SELECT MAX (DtFatura) FROM Faturas  
WHERE Faturas.Id_Fornecedor = Fornecedores.Id_Fornecedor) AS  
UltimaFatura FROM Fornecedores  
ORDER BY UltimaFatura DESC;
```

Nesse caso, mostramos uma *subquery* (subconsulta) no **SELECT** por meio de uma correlação, ou seja, utilizamos o **SELECT** como subconsulta para, dentro da consulta, calcular a data máxima (*DtFatura*) das faturas para cada fornecedor na tabela *fornecedores*, referenciando a coluna ***Id_fornecedor*** presente na tabela *Faturas*.



Saiba mais

Podemos, na maioria dos casos, substituir uma *subquery* (subconsulta) por um *join* (junção), porque os *joins* são mais simples de entender, ler e executar.

Caso utilizássemos um *join* em vez da *subquery*, teríamos:

```
SELECT NomeFornecedor, MAX (DtFatura) AS UltimaFatura  
FROM  
Fornecedores LEFT JOIN Faturas ON Faturas.Id_fornecedor = Forne-  
cedores.Id_Fornecedor  
GROUP BY NomeFornecedor  
ORDER BY UltimaFatura DESC;
```

Essa consulta faz a junção ou o cruzamento das tabelas de *Fornecedores* e *Faturas*, agrupando as linhas por **NomeFornecedor** e, em seguida, usa a função **MAX** para calcular a data máxima da fatura para cada fornecedor. Como é possível verificar, essa consulta é mais simples e fácil de ser lida do que a anterior com a subconsulta. Além disso, essa consulta é executada de forma mais rápida, considerando que, por meio da junção (*join*), é realizada a busca uma única vez, mas, quando utilizamos subconsultas, é feita uma varredura para cada linha em execução, consumindo mais recursos do sistema gerenciador de banco de dados (SGBD).



Fique atento

Podemos, também, codificar uma subconsulta dentro de outra subconsulta, o que obviamente aumentaria a complexidade do SELECT; todavia, essa não é uma boa prática, apesar de ser possível. Nesse caso, temos uma leitura da instrução SQL mais complicada e uma diminuição do desempenho da consulta.

Há quatro formas de criar uma subconsulta em uma instrução SELECT:

- Por meio de uma cláusula WHERE como condição de pesquisa;
- Por meio de uma cláusula HAVING como condição de pesquisa;
- Por meio de uma cláusula FROM para especificação da tabela;
- Por meio de um SELECT para especificação de uma coluna.



Leituras recomendadas

ELMASRI, R.; NAVATHE, S. B. *Sistemas de banco de dados*. 6. ed. São Paulo: Pearson, 2010.

KORTH, H. F.; SILBERSCHATZ, A.; SUDARSHAN, S. *Sistema de banco de dados*. 6. ed. Rio de Janeiro: Campus, 2012.

HEUSER, C. A. *Projeto de banco de dados*. 6. ed. Porto Alegre: Bookman, 2010. (Série Livros Didáticos Informática UFRGS, v.4).

RAMAKRISHNAN, R. *Sistemas de gerenciamento de banco de dados*. 3. ed. Porto Alegre: Penso, 2009.

SETZER, V. W. *Banco de dados: conceitos, modelos, gerenciadores, projeto lógico, projeto físico*. 3. ed. São Paulo: Blücher, 2002.

Conteúdo:



