Técnico em Desenvolvimento de Sistemas

Consulta com múltiplas tabelas: união, intersecção, junção, agrupamento

Neste material, você expandirá seu conhecimento sobre consultas em banco de dados. Para os exemplos, será utilizado o banco de dados definido no conteúdo **Consulta de dados**, desta unidade.

Sabe-se que é possível realizar várias operações de consulta de dados em uma única tabela, como, por exemplo, a listagem de todas as informações que constam nas colunas da tabela de produtos, usando a instrução em SQL (*structured query language*). Observe o exemplo a seguir:

Exemplo 1:

```
SELECT * FROM Produtos
```

idProduto	nomeProduto	categoriaProduto	valorVendaProduto	quantidadeProduto
1	Refrigerante	Bebidas	5.50	300
2	Arroz 5kg	Mercearia	8.50	100
3	Feijão	Mercearia	6.50	800
4	Detergente	Limpeza	2.10	100
5	Leite	Laticínios	2.70	600
6	Bolacha recheada	NULL	1.50	200
7	Leite condensado	Mercearia	4.50	500

Tabela 1 – Listagem de todos os produtos

Nesta consulta simples, são mostrados os nomes de todas as colunas, utilizando o caractere curinga (*) depois da cláusula **SELECT**, e, na sequência, vem a cláusula **FROM** seguida do nome da tabela, que neste caso é **Produtos**.

Você deve ter estudado também que é possível realizar filtros para retornar apenas os registros que correspondem aos critérios fornecidos, por exemplo, produtos que são da categoria “Mercearia”, usando a seguinte instrução em SQL. Confira o exemplo a seguir.

Exemplo 2:

```
SELECT * FROM Produtos WHERE categoriaProduto='Mercearia'
```

idProduto	nomeProduto	categoriaProduto	valorVendaProduto	quantidadeProduto
2	Arroz 5kg	Mercearia	8.50	100
3	Feijão	Mercearia	6.50	800
7	Leite condensado	Mercearia	4.50	500

Tabela 2 – Listagem de produtos da categoria "Mercearia"

Consultar dados de várias tabelas é muito comum quando se trabalha com banco de dados relacionais. Será importante se você souber usar os operadores em SQL para fazer isso.

Os bancos de dados relacionais são construídos com várias tabelas que se referem umas às outras. As linhas de uma tabela referem-se a linhas específicas em outra tabela, que são conectadas por algumas colunas pelo seu ID.

Para que você possa realizar consultas entre tabelas que se relacionam, será necessário inicialmente fazer alguns ajustes no banco de dados atual. Você já pensou que todo produto tem fornecedor e uma marca? Então, você criará as tabelas **Fornecedor** e **Marca** e fará o relacionamento com a tabela de produtos.

A tabela de fornecedor pode ser criada com o seguinte *script*:

```
CREATE TABLE Fornecedor (
idFornecedor int(11) auto_increment,
nomeFornecedor varchar(200),
cnpjFornecedor varchar(14),
PRIMARY KEY(idFornecedor)
);
```

Inclua os seguintes registros para seus testes:

INSERT INTO Fornecedor (nomeFornecedor, cnpjFornecedor) VALUES ('Brasil Alimentos', '10000000000101'), ('Laticínios Norte Minas', '20000000000102'), ('Alimentos do Centro-Oeste', '30000000000103');

idFornecedor	nomeFornecedor	cnpjFornecedor
1	Brasil Alimentos	10000000000101
2	Laticínios Norte Minas	20000000000102
3	Alimentos do Centro-Oeste	30000000000103

Tabela 3 – Listagem de fornecedores

A tabela da marca tem os seguintes campos:

- **idMarca** (campo do tipo inteiro, autoincremento e chave primária)
- **nomeMarca** (campo do tipo **varchar**, tamanho 200)

O *script* de criação da tabela é o seguinte:

```
CREATE TABLE Marca (
idMarca int(11) auto_increment,
nomeMarca varchar(200),
PRIMARY KEY(idMarca)
);
```

Como exemplo, utilize os seguintes registros:

```
INSERT INTO Marca (nomeMarca) VALUES
('Cereais do bom'),
('Bebidas do Brasil'),
('Doce Brasil');
```

idMarca	nomeMarca
1	Cereais do bom
2	Bebidas do Brasil
3	Doce Brasil

Tabela 4 – Listagem de marcas

Após a criação das novas tabelas com seus respectivos registros, é preciso ajustar a tabela de produtos para que ela contenha colunas do fornecedor (**idFornecedor**) e da marca (**idMarca**) e os respectivos registros.

Para isso, deve-se criar as duas novas colunas e as respectivas chaves estrangeiras:

ALTER TABLE Produtos ADD idFornecedor INT;

ALTER TABLE Produtos ADD CONSTRAINT FK_Fornecedor FOREIGN KEY (idFornecedor) REFERENCES Fornecedor (idFornecedor);

ALTER TABLE Produtos ADD idMarca INT;

ALTER TABLE Produtos ADD CONSTRAINT FK_Marca FOREIGN KEY (idMarca) REFERENCES Marca (idMarca);

Após esse ajuste, realize as operações de **UPDATE** para preencher essas novas colunas de maneira que fiquem com valores como os apresentados na tabela 5.

idProduto	nomeProduto	categoriaProduto	valorVendaProduto	quantidadeProduto	idFornecedor	idMarca
1	Refrigerante	Bebidas	5.50	300	3	2
2	Arroz 5kg	Mercearia	8.50	100	1	1
3	Feijão	Mercearia	6.50	800	1	1
4	Detergente	Limpeza	2.10	100	3	1
5	Leite	Laticínios	2.70	600	2	3
6	Bolacha recheada		1.50	200	3	3
7	Leite condensado	Mercearia	4.50	500	2	3

Tabela 5 – Listagem dos produtos com as colunas de fornecedor e de marca

Consultando dados de múltiplas tabelas utilizando WHERE

Como visto, a tabela de produtos agora contém colunas referentes a informações que estão em outras tabelas, como, por exemplo, o nome da marca, nome e CNPJ do fornecedor. Suponha que você queira realizar uma consulta com o nome dos produtos, o preço e o nome da marca. Você pode unir as linhas da tabela de produtos com as linhas da tabela de marca utilizando para isso a cláusula **WHERE**. Confira o exemplo de consulta em SQL a seguir.

Exemplo 3:

SELECT Produtos.nomeProduto, Produtos.valorVendaProduto, Marca.nomeMarca FROM Produtos, Marca WHERE Produtos.idMarca = Marca.idMarca

nomeProduto	valorVendaProduto	nomeMarca
Arroz 5kg	8.50	Cereais do bom
Feijão	6.50	Cereais do bom
Detergente	2.10	Cereais do bom
Refrigerante	5.50	Bebidas do Brasil
Leite	2.70	Doce Brasil
Bolacha recheada	1.50	Doce Brasil
Leite condensado	4.50	Doce Brasil

Tabela 6 – Listagem dos produtos com as colunas de nome, valor e marca

O comando **SELECT**, nessa consulta, lista as colunas **nomeProduto** e **valorVendaProduto** da tabela de produtos e **nomeMarca** da tabela de marca. Cada nome da coluna é precedido pelo nome da tabela correspondente, separado por um ponto.

Em seguida, os nomes das tabelas são listados após a cláusula **FROM**, separados por vírgulas.

A última parte desta consulta é o **WHERE**, com uma condição que especifica como unir as linhas de ambas as tabelas. Nesse caso, os valores da coluna **idMarca** da tabela de produtos correspondem aos valores do ID da coluna da tabela de marca e as linhas são unidas quando os valores são iguais (**Produtos.idMarca** = **Marca.idMarca**).

O produto “feijão” tem um **idMarca 1**. O mesmo valor na coluna **idMarca** na tabela de marca refere-se a “cereais do bom”, conforme apresentado na tabela 6, sobre nome, valor e marca.

Pode existir situações em que o mesmo nome de coluna esteja em ambas as tabelas. Para esse caso, é preciso que as tabelas sejam diferenciadas na cláusula **SELECT** com o uso de alias. Observe:

Exemplo 4:

SELECT tabela1.nome AS Nome, tabela2.nome AS Cidade FROM tabela1, tabela2 WHERE tabela1.id = tabela2.id

Note que a **tabela1** e a **tabela2** têm a coluna com o mesmo nome. Para diferenciá-las, utiliza-se o comando alias, colocando nomes diferentes. Neste exemplo, no entanto, as tabelas de produtos, de marca e de fornecedor têm nome de colunas diferentes, sem nenhum em comum.

Com relação ao alias, ele pode ser utilizado para simplificar suas consultas. Confira seu uso no mesmo exemplo anterior de listagem de produtos com nome, valor e marca.

Exemplo 5:

SELECT P.nomeProduto, P.valorVendaProduto, M.nomeMarca FROM Produtos AS P, Marca AS M WHERE P.idMarca = M.idMarca

nomeProduto	valorVendaProduto	nomeMarca
Arroz 5kg	8.50	Cereais do bom
Feijão	6.50	Cereais do bom
Detergente	2.10	Cereais do bom
Refrigerante	5.50	Bebidas do Brasil
Leite	2.70	Doce Brasil
Bolacha recheada	1.50	Doce Brasil
Leite condensado	4.50	Doce Brasil

Em **FROM** é definido o alias **P** para a tabela de produtos e o alias **M** para a tabela de marca. Observe que, após isso, não foi preciso, nas cláusulas **SELECT** e **WHERE**, colocar diretamente o nome da tabela em questão, e sim o alias atribuído. Isso é uma prática largamente utilizada, principalmente quando se está trabalhando com muitas tabelas. Um ponto importante é que você pode definir o nome que quiser para o alias das tabelas, desde que eles sejam diferentes.

A definição de alias pode ser feita com o uso da palavra-chave **AS** (como em Produto AS P), mas pode também ser definida diretamente sem esse termo (exemplo Produto P). É preferível, por clareza, utilizar sempre o termo **AS** ao definir alias.

Confira agora como realizar uma consulta de envolva três tabelas para gerar uma listagem de todos os produtos com nome, marca, fornecedor e valor do produto. Será adotado neste exemplo o uso de alias.

Exemplo 6:

SELECT P.nomeProduto, M.nomeMarca, F.nomeFornecedor, P.valorVendaProduto FROM Produtos AS P, Marca AS M, Fornecedor AS F WHERE P.idMarca = M.idMarca AND P.idFornecedor = F.idFornecedor

nomeProduto	nomeMarca	nomeFornecedor	valorVendaProduto
Arroz 5kg	Cereais do bom	Brasil Alimentos	8.50
Feijão	Cereais do bom	Brasil Alimentos	6.50
Detergente	Cereais do bom	Alimentos do Centro-Oeste	2.10
Refrigerante	Bebidas do Brasil	Alimentos do Centro-Oeste	5.50
Leite	Doce Brasil	Laticínios Norte Minas	2.70
Bolacha recheada	Doce Brasil	Alimentos do Centro-Oeste	1.50
Leite condensado	Doce Brasil	Laticínios Norte Minas	4.50

Tabela 8 – Listagem de produtos com junção entre três tabelas

O comando **SELECT** nesta consulta lista as colunas **nomeProduto** e **valorVendaProduto** da tabela de produtos, **nomeMarca** da tabela de marca e **nomeFornecedor** da tabela de fornecedor. Cada nome da coluna é precedido pelo alias correspondente ao nome da tabela, separado por um ponto.

Em seguida, os nomes das tabelas são listados depois da cláusula **FROM**, seguido do correspondente alias, separados por vírgulas.

Por fim, há o comando **WHERE**, com uma condição que especifica como unir as linhas das tabelas. Aqui, os valores da coluna de chave estrangeira **idMarca** da tabela de produtos correspondem aos valores da coluna de ID da tabela de marca,

assim como o valor da coluna **idFornecedor** da tabela de produtos corresponde aos valores do ID da tabela de fornecedor. É dessa forma que se vincula a tabela principal de produtos às demais tabelas de marca e fornecedor.

Usar o comando **WHERE** para consultar dados de múltiplas tabelas é um padrão mais antigo no SQL, que ainda está disponível e é de fácil utilização. A seguir, veja um outro método, mais atual, que emprega o operador **JOIN**.

Consultando dados de múltiplas tabelas usando junção – JOIN

Atualmente, o método para juntar dados de várias tabelas é o operador **JOIN**, também conhecido como **INNER JOIN**.

A sintaxe de uma SQL usando o operador **JOIN** é:

```
SELECT * FROM tabela1 JOIN tabela2
ON tabela1.id1=tabela2.id2
```

Após a cláusula **FROM**, descreva o nome da primeira tabela que contém as colunas das quais você deseja obter informações. Depois, utilize a palavra-chave **JOIN**, seguida do nome da segunda tabela que deseja usar.

Em seguida, a instrução **ON** informa à consulta em quais colunas os dados devem ser unidos, o que geralmente envolve a chave estrangeira de uma tabela e a chave primária de outra, referenciada pela primeira. Isso é escrito como **tabela1.id1** e **tabela2.id2** e, após o nome da tabela, há um ponto e o nome da coluna.

Para ver seu funcionamento, você retornará ao exemplo anterior, juntando as tabelas de produtos e de marca. A consulta gerará uma listagem com todos os produtos, atribuindo o nome e valor do produto que está na tabela de produtos e o nome da marca que está na tabela de marca, ou seja, será necessário juntar as tabelas de produtos e de marca.

Exemplo 7:

```
SELECT P.nomeProduto, P.valorVendaProduto, M.nomeMarca
FROM Produtos AS P JOIN Marca AS M
ON P.idMarca=M.idMarca
```

A junção é feita pelo operador **JOIN**. Na cláusula **FROM**, ao nome da primeira tabela (**Produto**) é atribuído um alias **P** seguido da palavra-chave **JOIN**, depois pelo nome da segunda tabela (**Marca**) e o respectivo alias **M**. Isso é seguido também pela palavra-chave **ON** e pela condição de juntar as linhas das diferentes tabelas. O nome da marca é atribuído com base na coluna **idMarca** da tabela de marca com alias M, que é equivalente ao **idMarca** da tabela de produtos com alias **P**.

A seguir, veja o conjunto de resultados da pesquisa:

nomeProduto	valorVendaProduto	nomeMarca
Arroz 5kg	8.50	Cereais do bom
Feijão	6.50	Cereais do bom
Detergente	2.10	Cereais do bom
Refrigerante	5.50	Bebidas do Brasil
Leite	2.70	Doce Brasil
Bolacha recheada	1.50	Doce Brasil
Leite condensado	4.50	Doce Brasil

Tabela 9 – Listagem dos produtos com as colunas de nome, valor e marca, com a utilização do **JOIN**

Confira agora como realizar uma consulta que envolva três tabelas para gerar uma listagem de todos os produtos com nome, marca, fornecedor e valor do produto. Será adotado neste exemplo o uso de **JOIN**.

Exemplo 8:

```
SELECT P.nomeProduto, M.nomeMarca, F.nomeFornecedor, P.valorVendaProduto
FROM
Produtos AS P JOIN Marca AS M ON P.idMarca = M.idMarca
JOIN Fornecedor AS F ON P.idFornecedor = F.idFornecedor
```

Na cláusula **FROM**, ao nome da primeira tabela (**Produto**) é atribuído um alias **P** seguido da palavra-chave **JOIN**, em seguida pelo nome da segunda tabela (**Marca**) e o respectivo alias **M**. Isso é seguido também pela palavra-chave **ON** e pela condição de juntar as linhas das diferentes tabelas. O nome da marca é atribuído com base na coluna **idMarca** da tabela de marca com alias **M**, que é equivalente ao **idMarca** da tabela de produtos com alias **P**. Como uma terceira tabela será adicionada, insere-se a palavra-chave **JOIN**, em seguida o nome da terceira tabela, que é **Fornecedor**, e o respectivo alias **F**. Por fim, o comando é seguido pela palavra-chave **ON** e pela condição de juntar as tabelas de produtos e de fornecedor, com base na coluna **idFornecedor** da tabela de fornecedor com alias **F**, que é equivalente ao **idFornecedor** da tabela de produtos com alias **P**.

Na cláusula **SELECT**, haverá os campos que serão mostrados, seguidos pelo alias e um ponto com o nome da coluna, separados por vírgula.

A seguir, está o conjunto de resultados da pesquisa:

nomeProduto	nomeMarca	nomeFornecedor	valorVendaProduto
Arroz 5kg	Cereais do bom	Brasil Alimentos	8.50
Feijão	Cereais do bom	Brasil Alimentos	6.50
Detergente	Cereais do bom	Alimentos do Centro-Oeste	2.10
Refrigerante	Bebidas do Brasil	Alimentos do Centro-Oeste	5.50
Leite	Doce Brasil	Laticínios Norte Minas	2.70
Bolacha recheada	Doce Brasil	Alimentos do Centro-Oeste	1.50
Leite condensado	Doce Brasil	Laticínios Norte Minas	4.50

Tabela 10 – Listagem de produtos com junção entre três tabelas, com a utilização do **JOIN**

Usar o operador **JOIN** é o método mais comum para juntar múltiplas tabelas em um banco de dados relacional. Como mencionado anteriormente, o operador **INNER JOIN** é equivalente ao **JOIN** e você pode usá-los da mesma maneira.

Filtrando dados de múltiplas tabelas usando INNER JOIN e WHERE

Utilizar o operador **INNER JOIN** para consultar dados de múltiplas tabelas também permite a realização de filtros usando a cláusula **WHERE**. Considere, por exemplo, que você deseja obter a listagem de todos os produtos, com nome, valor e marca, mas somente da categoria “Mercearia”, ordenada por ordem alfabética de nome do produto.

Exemplo 9:

```
SELECT P.nomeProduto, P.valorVendaProduto, M.nomeMarca
FROM Produtos AS P INNER JOIN Marca AS M
ON P.idMarca=M.idMarca
WHERE P.categoriaProduto='Mercearia'
ORDER BY P.nomeProduto
```

A primeira parte da consulta é igual à que você já estava trabalhando. Nesta consulta, entretanto, são considerados somente os produtos da categoria “Mercearia”. Para isso, filtre as linhas dos conjuntos de resultados, utilizando a cláusula **WHERE** para verificar se a categoria é realmente “Mercearia”.

Confira o resultado da consulta:

nomeProduto	valorVendaProduto	nomeMarca
Arroz 5kg	8.50	Cereais do bom
Feijão	6.50	Cereais do bom
Leite condensado	4.50	Doce Brasil

Tabela 11 – Listagem de produtos da categoria “Mercearia”

Tipos de JOINS no SQL

Os operadores de junção do tipo **INNER JOIN** retornam os registros nos quais houver pelo menos uma correspondência entre ambas as tabelas.

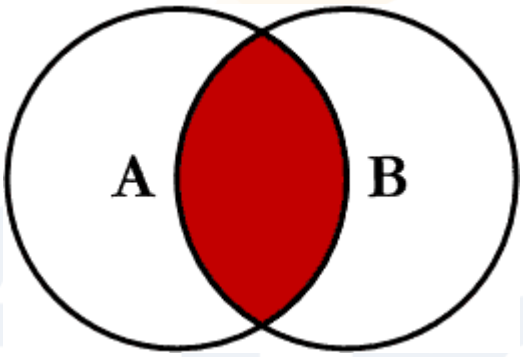

Figura 1 – **INNER JOIN**

Contudo, existem outros operadores de junção, como o **LEFT JOIN** e o **RIGHT JOIN**, que serão abordados a seguir:

Clique ou toque para visualizar o conteúdo.

LEFT JOIN

O operador de junção **LEFT JOIN** em SQL retorna todos os registros da tabela à esquerda e apenas os registros correspondentes da direita, ou seja, isso implica que, se uma linha específica estiver presente na tabela à esquerda, mas não à direita, o resultado incluiu essa linha, mas com um valor **NULL** em cada coluna da direita. Se um registro da tabela da direita não estiver na esquerda, ele não será incluído no resultado.

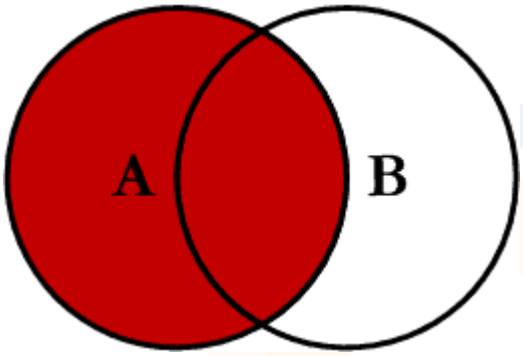

Figura 2 – **LEFT JOIN**

A sintaxe geral para **LEFT JOIN** é a seguinte:

```
SELECT coluna FROM tabelaA LEFT JOIN tabelaB
ON tabelaA.id1=tabelaB.id2
```

Como exemplo, considere o seguinte cenário: você precisa obter o total agrupado de vendas dos produtos, incluindo também os produtos que não tiveram nenhuma venda. Lembre-se de que a tabela de produtos contém sete produtos cadastrados, porém apenas dois produtos têm registros na tabela de vendas.

Para juntar essas duas tabelas e obter as informações de que precisa para a análise, utilize a seguinte consulta em SQL com **LEFT JOIN**:

Exemplo 10:

```
SELECT A.idProduto, A.nomeProduto,
sum(V.quantidadeVendida*V.valorVendido) AS TotalVendido
FROM Produtos A LEFT JOIN Vendas V
ON A.idProduto = V.idProduto
GROUP BY A.idProduto
```

idProduto	nomeProduto	TotalVendido
1	Refrigerante	16.50
2	Arroz 5kg	212.50
3	Feijão	NULL
4	Detergente	NULL
5	Leite	NULL
6	Bolacha recheada	NULL
7	Leite condensado	NULL

Tabela 12 – Total em venda dos produtos, com a utilização do **LEFT JOIN**

Com isso, você mantém todos os registros dos produtos e anexa o ID do produto, o nome e o somatório de vendas pela função de agregação **SUM**. Se você estivesse utilizando **INNER JOIN**, seriam listados apenas os produtos com vendas, no caso do exemplo, apenas dois produtos. Entretanto, quando se usa o **LEFT JOIN**, ele traz os produtos sem vendas, inserindo um valor **NULL** nas colunas relativas na tabela das vendas, o neste caso seriam a quantidade vendida e o valor vendido, uma vez que seus registros não estão presentes na tabela de vendas. É assim que funciona o **LEFT JOIN**. A cláusula **GROUP BY** é responsável por agrupamentos de dados, ou seja, agrupar os produtos pelo seu ID. Essa cláusula será abordada na última seção deste conteúdo.

RIGHT JOIN

O operador de junção **RIGHT JOIN** em SQL é muito semelhante ao **LEFT JOIN**, entretanto, aquele mantém todos os registros da tabela da direita, mesmo que eles não possam ser combinados na tabela da esquerda.

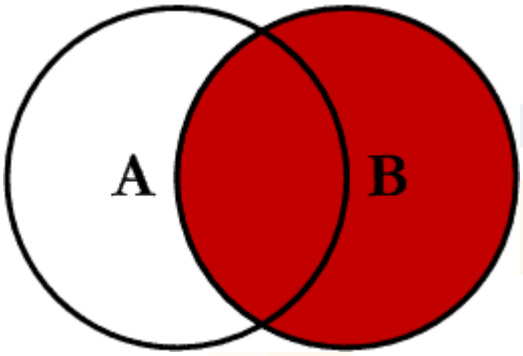

Figura 3 – **RIGHT JOIN**

A sintaxe geral para **RIGHT JOIN** também é semelhante:

```
SELECT coluna FROM tabelaA RIGHT JOIN tabelaB
ON tabelaA.id1=tabelaB.id2
```

Como exemplo, considere o seguinte cenário: na tabela de das vendas, ficou armazenado um registro de venda de um produto que não consta mais na tabela de produtos, ou seja, o produto foi excluído da tabela, porém seu registro ainda consta na tabela de vendas. Quando um registro da tabela da direita não estiver na esquerda, ele será incluído no resultado, se você utilizar o **RIGHT JOIN**, e os campos da tabela da esquerda serão preenchidos por **NULL**.

Exemplo 11:

```
SELECT A.idProduto, A.nomeProduto,
sum(V.quantidadeVendida*V.valorVendido) AS TotalVendido
FROM produtos A RIGHT JOIN vendas V
ON A.idProduto = V.idProduto
GROUP BY A.idProduto
```

idProduto	nomeProduto	TotalVendido
2	Arroz 5kg	212.50
1	Refrigerante	16.50
NULL	NULL	9.00

Tabela 13 – Total de registro dos produtos, com a utilização do **RIGHT JOIN**

Trabalhar com **JOINS** em SQL é um dos principais requisitos para qualquer pessoa que atue com banco de dados relacionais. Para poder dominar seu uso, é importante praticar cada um dos exemplos disponibilizados neste material, os quais contém aspectos reais de ambientes do mercado de trabalho.

O dono de um estabelecimento está precisando criar uma promoção de queima de estoque para alguns itens. Elabore uma consulta que exiba nome do produto, valor de venda, nome da marca e fornecedor de todos os produtos que custam entre 4.00 e 6.50, ordenando pelo valor de venda. Utilize a junção para elaboração desta consulta.

União

Os comandos em **SQL UNION** e **UNION ALL** são usados em SQL para obter dados de várias tabelas em seu banco de dados.

Você pode usar a cláusula **UNION** para combinar as linhas da tabela de duas consultas diferentes em um resultado. Ao contrário da junção (**JOIN**), que combina colunas de tabelas diferentes, a união combina linhas de tabelas diferentes.

Existe diferença entre utilizar UNION e UNION ALL:

- UNION só retorna registros únicos em ambas as tabelas.
- UNION ALL retorna todos os registros, incluindo os registros que possam estar em ambas as tabelas, ou seja, os duplicados.

A sintaxe geral para união é:

```
SELECT coluna1, coluna2
FROM tabelaA WHERE condicao
UNION
SELECT coluna1, coluna2
FROM tabelaB WHERE condicao
```

Para unir duas tabelas, existem alguns requisitos. O não cumprimento dessas regras resultará em mensagens de erro:

- O número de colunas usadas em sua primeira consulta e sua segunda consulta deve corresponder ao mesmo tipo de dados (por exemplo: INT, VARCHAR etc.).
- O nome das colunas incluídas em ambas as consultas pode ser diferente; quando isso ocorrer, o conjunto resultante de dados mostrará os nomes das colunas da primeira consulta.

Para exemplificar, crie uma nova tabela, **Funcionario**, que conterá os seguintes campos:

- **idFuncionario** (campo do tipo inteiro, autoincremento e chave primária)
- **nomeFuncionario** (campo do tipo varchar, tamanho 200)
- **cpfFuncionario** (campo do tipo varchar, tamanho 11)
- **cidadeFuncionario** (campo do tipo varchar, tamanho 200)

É possível usar o seguinte *script* SQL para criação da tabela:

```
CREATE TABLE Funcionario (
idFuncionario int(11) auto_increment,
nomeFuncionario varchar(200),
cpfFuncionario varchar(11),
cidadeFuncionario varchar(200),
PRIMARY KEY(idFuncionario)
);
```

Incluem-se então os seguintes registros para exemplificação:

```
INSERT INTO Funcionario (nomeFuncionario,cpfFuncionario,cidadeFuncionario) VALUES
('João dos Santos','10000000001','São Paulo'),
('Maria da Silva','20000000002','Rio de Janeiro'),
('Ana Maria','30000000003','Brasília'),
('Brasil Alimentos','40000000004','Cuiabá');
```

idFuncionario	nomeFuncionario	cpfFuncionario	cidadeFuncionario
1	João dos Santos	10000000001	São Paulo
2	Maria da Silva	20000000002	Rio de Janeiro
3	Ana Maria	30000000003	Brasília
4	Brasil Alimentos	40000000004	Cuiabá

Tabela 14 – Listagem de funcionários

No banco de dados, consta a tabela dos fornecedores, com os seguintes registros:

idFornecedor	nomeFornecedor	cnpjFornecedor
1	Brasil Alimentos	10000000000101
2	Laticínios Norte Minas	20000000000102
3	Alimentos do Centro-Oeste	30000000000103

Tabela 15 – Listagem de fornecedores

Continuando neste exemplo, imagine que será preciso gerar uma listagem unificada, com todos os nomes dos fornecedores e funcionários juntos em ordem alfabética do nome. Para isso, é possível utilizar o operador **UNION**, do seguinte modo:

Exemplo 12:

```
SELECT nomeFuncionario FROM Funcionario
UNION
SELECT nomeFornecedor FROM Fornecedor
ORDER BY nomeFuncionario
```

nomeFuncionario
Alimentos do Centro-Oeste
Ana Maria
Brasil Alimentos
João dos Santos
Laticínios Norte Minas
Maria da Silva

Tabela 16 – Listagem de funcionários e fornecedores

Observe que o operador **UNION** não permite duplicar registros. Além disso, foi colocada a cláusula **ORDER BY** na segunda consulta, porém com o nome do campo da primeira consulta, neste caso, **nomeFuncionario**.

Por fim, se você utilizar o **UNION ALL**, serão retornados também os registros duplicados, neste caso, “Brasil Alimentos”, que consta tanto em funcionários quanto em fornecedores.

Exemplo 13:

```
SELECT nomeFuncionario FROM Funcionario
UNION ALL
SELECT nomeFornecedor FROM Fornecedor
ORDER BY nomeFuncionario
```

nomeFuncionario
Alimentos do Centro-Oeste
Ana Maria
Brasil Alimentos
Brasil Alimentos
João dos Santos
Laticínios Norte Minas
Maria da Silva

Tabela 17 – Listagem de funcionários e fornecedores com registros duplicados

Intersecção

O operador **INTERSECT** retorna apenas as linhas distintas de duas ou mais consultas. Ao contrário do operador de união **UNION**, o **INTERSECT** retorna à intersecção entre as consultas.

A sintaxe geral para intersecção é:

```
SELECT coluna1
FROM tabelaA
INTERSECT
SELECT coluna1
FROM tabelaB
```

Para usar este operador em duas consultas, siga estas regras:

- A ordem e o número de colunas na lista de seleção das consultas devem ser os mesmos.
- Os tipos de dados das colunas correspondentes devem ser compatíveis (por exemplo: **INT**, **VARCHAR** etc.).

Cabe ressaltar que o banco de dados MySQL não tem suporte para o operador **INTERSECT**, entretanto, outros bancos de dados podem ter suporte a ele.

Agrupamento com GROUP BY

Para entender melhor sobre agrupamento de dados, você trabalhará inicialmente na visão de uma única tabela, a tabela de produtos, tratada na seção anterior.

A cláusula **GROUP BY** é uma forma muito comum de categorizar dados de uma consulta, realizando agrupamento a partir dos valores de uma ou mais colunas e permitindo cálculos ou sumarizações a partir desse agrupamento. A sintaxe básica dessa cláusula é a seguinte:

SELECT coluna FROM tabela GROUP BY colunas_para_agrupamento

Quando uma cláusula **GROUP BY** é adicionada a uma instrução na linguagem SQL, é dito para o banco de dados exibir uma única linha exclusiva para cada valor da coluna fornecida. Isso tem algumas implicações importantes.

Quando entendido que a cláusula **GROUP BY** é uma forma de representar várias linhas como uma única linha, o SQL só pode executar a consulta se puder calcular um valor para cada uma das colunas fornecidas, ou seja, isso significa que cada coluna identificada no **SELECT** deve ser incluída no **GROUP BY** para garantir que cada linha tenha um valor único.

Suponha que você quer agrupar as categorias de todos os produtos cadastros no banco de dados:

Exemplo 14:

```
SELECT categoriaProduto
FROM Produtos
GROUP BY categoriaProduto
```

categoriaProduto
Bebidas
Mercearia
Limpeza
Laticínios

Tabela 18 – Categoria dos produtos agrupadas

Caso você incluísse uma segunda ou terceira coluna nessa consulta, você obteria pelo MySQL apenas o primeiro registro encontrado na tabela com o valor agrupado correspondente. Por exemplo, caso incluísse **nomeProduto** no **SELECT**, obteria as categorias e o primeiro produto na tabela para cada categoria.

Exemplo 15:

```
SELECT categoriaProduto, nomeProduto
FROM Produtos
GROUP BY categoriaProduto
```

categoriaProduto	nomeProduto
Bebidas	Refrigerante
Laticínios	Leite
Limpeza	Detergente
Mercearia	Arroz 5kg

Tabela 19 – Agrupamento por categoria e coluna **nomeProduto**

Essa, de fato, não é a função real de um agrupamento por **GROUP BY**. Geralmente, o que se quer obter são contagens, estatísticas ou operações sobre os registros agrupados pela coluna escolhida. Para isso, é comum usar a funções

agregadas de SQL.

Clique ou toque para visualizar o conteúdo.

Utilizando agrupamento em funções agregadas

Como visto anteriormente, as funções de agregação **COUNT**, **SUM**, **AVG**, **MIN** e **MAX** permitem realizar o cálculo de um conjunto de linhas e retornar um único valor. A cláusula **GROUP BY** é frequentemente usada com funções agregadas para realizar cálculos e retornar um valor único.

Por exemplo, se você quisesse saber o número de produtos de cada categoria, poderia usar a função **COUNT** com a cláusula **GROUP BY** da seguinte forma:

Exemplo 16:

```
SELECT categoriaProduto, COUNT(*)
FROM Produtos
GROUP BY categoriaProduto
```

categoriaProduto	COUNT(*)
Bebidas	1
Mercearia	3
Limpeza	1
Laticínios	1
NULL	1

Tabela 20 – Quantidade de produtos por categoria

Se você quisesse exibir o valor do maior e do menor produto, o preço médio e o respectivo número de produtos em cada categoria, poderia usar a função **MIN**, **MAX**, **AVG**, **COUNT** com a cláusula **GROUP BY** da seguinte forma:

Exemplo 17:

```
SELECT categoriaProduto, MIN(ValorVendaProduto), MAX(ValorVendaProduto), AVG(ValorVendaProduto), COUNT(*)
FROM Produtos
GROUP BY categoriaProduto
```

categoriaProduto	MIN(valorVendaProduto)	MAX(valorVendaProduto)	AVG(ValorVendaProduto)	COUNT(*)
Bebidas	5.50	5.50	5.50	1
Mercearia	4.50	8.50	6.50	3
Limpeza	2.10	2.10	2.10	1
Laticínios	2.70	2.70	2.70	1
NULL	1.50	1.50	1.50	1

Tabela 21 – Listar valor do maior e do menor, média de preço de produtos por categoria

Em outro exemplo, ainda usando as funções de agregação, se você quisesse somar a quantidade em estoque de produtos por categoria, bastaria usar a função **SUM** com a cláusula **GROUP BY** da seguinte forma:

Exemplo 18:

```
SELECT categoriaProduto, SUM(quantidadeProduto)
FROM Produtos
GROUP BY categoriaProduto
```

categoriaProduto	SUM(quantidadeProduto)
Bebidas	300
Mercearia	1400
Limpeza	100
Laticínios	600
NULL	200

Tabela 22 – Somar a quantidade de produtos por categoria

Usando cláusula HAVING para filtrar grupos de registros

A cláusula **GROUP BY** é uma forma de resumir os dados exibindo vários registros em uma única linha representativa. Mas e se você quiser restringir esses grupos com base em fatores adicionais?

A cláusula **HAVING** é um modificador da cláusula **GROUP BY**, que permite especificar as condições que cada grupo deve satisfazer para ser incluído nos resultados.

A sintaxe geral é semelhante a esta:

SELECT coluna FROM tabela GROUP BY colunas_para_agrupamento HAVING condição

A operação é muito semelhante à cláusula **WHERE**, com a diferença de que a cláusula **WHERE** filtra registros únicos, enquanto a cláusula **HAVING** filtra grupo de registros.

Por exemplo, considere que você quer listar todas as categorias de produtos que tenha mais de um produto no banco de dados. Para isso, você usaria a função de agrupamento **COUNT** com a cláusula **GROUP BY**, seguidas da cláusula

HAVING, da seguinte forma:

Exemplo 19:

```
SELECT categoriaProduto, COUNT(*)
FROM Produtos
GROUP BY categoriaProduto
HAVING COUNT(*) > 1
```

categoriaProduto	COUNT(*)
Mercearia	3

Tabela 23 – Filtrando os produtos que tenham quantidade maior do que um na categoria

Utilizando agrupamento com múltiplas tabelas

Um dos recursos mais utilizados em agrupamento de registros é a realização da junção entre duas ou mais tabelas.

Para que se possa obter a quantidade e o total de vendas de todos os produtos agrupados por categoria, é preciso utilizar as funções de agregação **COUNT**, **SUM** e realizar uma junção entre as tabelas de produtos e vendas, usando **INNER JOIN**, e um agrupamento por categoria do produto, com a cláusula **GROUP BY**, da seguinte forma:

Exemplo 20:

```
SELECT A.categoriaProduto, COUNT(*),
SUM(B.quantidadeVendida*B.valorVendido) as TotalVendido
FROM Produtos A INNER JOIN Vendas B
```

```
ON (A.IdProduto=B.IdProduto)
GROUP BY A.categoriaProduto
```

categoriaProduto	COUNT(*)	TotalVendido
Mercearia	2	212.50
Bebidas	1	16.50

Tabela 24 – Total de produtos vendidos por categoria

Também é possível realizar agrupamentos em campos do tipo **date**, usando a função **YEAR**, que extrai os dados do ano de um campo **data**. Entretanto, é importante que a função esteja tanto na cláusula **SELECT** quanto na cláusula **GROUP BY**, conforme exemplo a seguir:

Exemplo 21:

```
SELECT YEAR(B.dataVenda), COUNT(*),
SUM(B.quantidadeVendida*B.valorVendido) as TotalVendido
FROM Produtos A INNER JOIN Vendas B
ON (A.IdProduto=B.IdProduto)
GROUP BY YEAR(B.dataVenda)
```

YEAR(B.dataVenda)	COUNT(*)	TotalVendido
2022	3	229.00

Tabela 25 – Agrupando as vendas de produtos por ano

Diferença entra a cláusula GROUP BY e DISTINCT

Se você usar a cláusula **GROUP BY** sem usar função de agregação na cláusula **SELECT**, a cláusula **GROUP BY** comportar-se-á como a cláusula **DISTINCT**.

A instrução a seguir usa a cláusula **GROUP BY** para agrupar todas as categorias na tabela de produtos.

Exemplo 22:

```
SELECT categoriaProduto
FROM Produtos
GROUP BY categoriaProduto
```

categoriaProduto
Bebidas
Mercearia
Limpeza
Laticínios
NULL

Tabela 26 – Agrupando produtos por categoria usando **GROUP BY**

Você pode obter um resultado semelhante, usando a cláusula **DISTINCT**.

Exemplo 23:

```
SELECT DISTINCT(categoriaProduto)
FROM Produtos
```

categoriaProduto
Bebidas
Mercearia
Limpeza
Laticínios
NULL

Tabela 27 – Agrupando produtos por categoria usando **DISTINCT**

Sobre o agrupamento com múltiplas tabelas, elabore uma consulta que agrupe todas as marcas e a quantidade dos respectivos produtos, ordenando pela marca.

Revisando e aplicando

No vídeo a seguir, você poderá expandir seu conhecimento e sua experiência com novos exemplos de consultas variadas. Acompanhe o passo a passo, pause ou retroceda sempre que necessário.

Encerramento

Como visto neste material, você pode consultar dados de múltiplas tabelas de diversas formas utilizando **WHERE** e **JOIN** em um banco de dados relacional. A junção que você usará depende diretamente da tarefa que precisa ser feita, e você saberá como proceder ao longo da sua trajetória.

Busque conhecer e praticar os operadores aqui apresentados. Além disso, tenha sempre em mente que, antes de utilizar qualquer operador em SQL, é preciso conhecer a estrutura de relacionamento das tabelas do seu banco de dados.