Índice

SELECT (SELECIONAR)

É usado para selecionar dados de um banco de dados.

Os dados retornados são armazenados em uma tabela de resultados, chamada conjunto de resultados.

SELECT *coluna1*, coluna2*, etc*
FROM *nome_tabela*;

Aqui, coluna1, coluna2, etc, são os nomes dos campos da tabela da qual você deseja selecionar os dados. Se você deseja selecionar todos os campos disponíveis na tabela, use a seguinte sintaxe:

SELECT * FROM *nome_tabela*;

Ex:

SELECT * Exemplo	SELECT Exemplo Colunas
SELECT * FROM Customers;	SELECT CustomerName, City FROM Customers;

SELECT DISTINCT (SELECIONAR DISTINTO)

É usada para retornar apenas valores distintos (diferentes).

Dentro de uma tabela, uma coluna geralmente contém muitos valores duplicados; e às vezes você só deseja listar os valores diferentes (distintos). Professor recomenda utilizar na maioria dos casos o GROUP BY para realizar essa função. (não é uma boa pratica mais é útil em determinadas situações).

Sintaxe SELECT DISTINCT

```
SELECT DISTINCT coluna1, coluna2, ...
FROM nome_tabela;
```

Ex:

SELECT NORMAL	SELECT the country FROM Customers;
SELECT DISTINCT	SELECT DISTINCT COUNTRY FROM Customers;
SELECT DISTINCT + COUNT	SELECT COUNT (DISTINCT Country) FROM Customers;

WHERE (ONDE)

É usada para filtrar registros. Usado para extrair apenas os registros que atendem a uma condição especificada.

Sintaxe WHERE

```
SELECT coluna1, coluna2, etc
FROM nome_tabela
WHERE condição
```

Exemplos de cláusula WHERE

A seguinte instrução SQL seleciona todos os clientes do país "Brasil", na tabela "Clientes":

Ex:

```
SELECT * FROM   Clientes
WHERE País = 'Brasil';
```

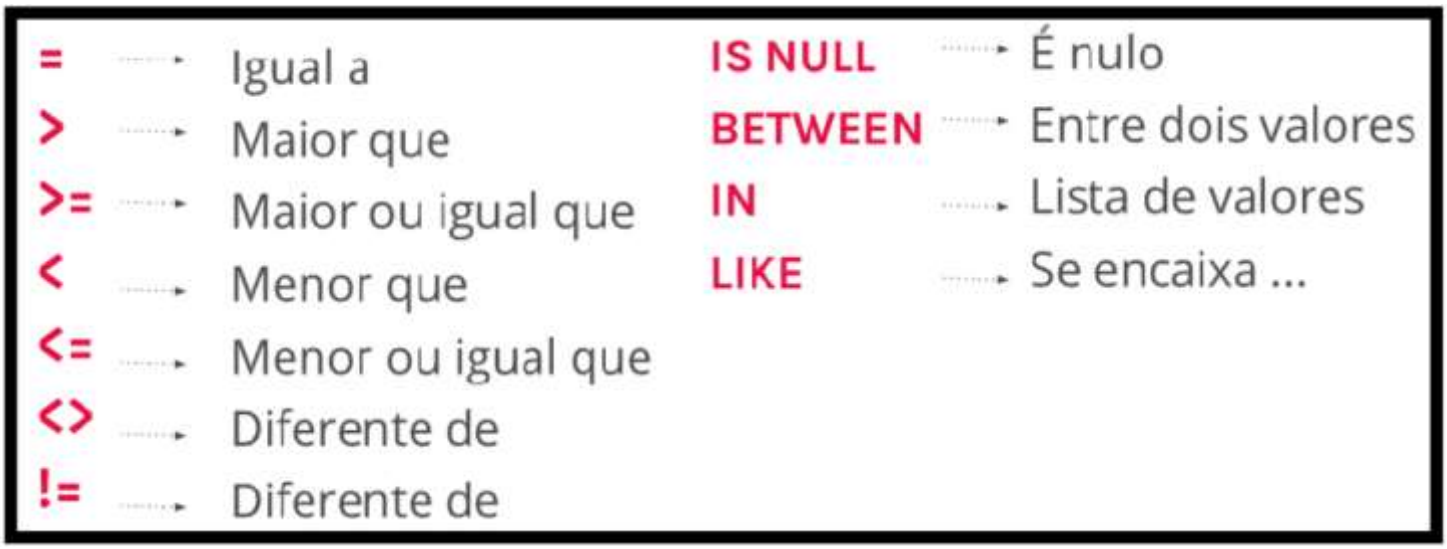
Campos de texto vs. campos numéricos.

O SQL requer aspas simples em torno dos valores de texto (a maioria dos sistemas de banco de dados também permite aspas duplas).
No entanto, os campos numéricos não devem ser colocados entre aspas:

Ex:

```
SELECT * FROM   Clientes
WHERE CustomerID = 1;
```

Operadores na cláusula WHERE
Os seguintes operadores podem ser usados na cláusula WHERE, OR e AND



AND, OR e NOT (E, OU, NÃO)

A cláusula WHERE pode ser combinada com os operadores AND, OR e NOT.

Os operadores AND e OR são usados para filtrar registros com base em mais de uma condição:

O operador AND exibe um registro se:
TODAS as condições separadas por AND forem TRUE.

O operador OR exibe um registro se:
ALGUMA das condições separadas por OR for TRUE.

O operador NOT exibe um registro se:
As condições NÃO forem verdadeiras.

AND Sintaxe:

```
SELECT column1, column2, etc
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

Ex:

```
SELECT
* FROM Customers
WHERE Country='Germany' AND City='Berlin';
```

OR Sintaxe:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3, etc;
```

Ex:

```
SELECT
* FROM Customers
WHERE City='Berlin' OR City='München';
```

NOT Sintaxe:

```
SELECT column1, column2, ... FROM table_name
WHERE NOT condition;
```

Ex:

```
SELECT
* FROM Customers
WHERE NOT Country='Germany';
```

Combinando AND, OR e NOT
Você também pode combinar os operadores AND, OR e NOT.

A seguinte instrução SQL seleciona todos os campos de "Clientes" onde o país é "Alemanha" E a cidade deve ser "Berlim" OU "Munique" (use parênteses para formar expressões complexas):

Ex:

```
SELECT
* FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München');
```

A seguinte instrução SQL seleciona todos os campos de "Clientes" onde o país NÃO é "Alemanha" e NÃO "EUA":

Ex:

```
SELECT
* FROM Customers
WHERE NOT Country='Germany' AND NOT Country='USA';
```

ORDER BY (ORDENE POR)

É usada para classificar o conjunto de resultados em ordem crescente ou decrescente.

A palavra-chave ORDER BY classifica os registros em ordem crescente (ASC) por padrão. Para classificar os registros em ordem decrescente, use a palavra-chave DESC.

Sintaxe ORDER BY

```
SELECT column1, column2,etc
FROM table_name
ORDER BY column1, column2, etc ASC ou DESC;
( você escolhe quais colunas quer em ordem).
```

Exemplo ORDER BY padrão (lembrando que por padrão fica ASC):

```
SELECT
* FROM Customers
ORDER BY Country;
```

Exemplo ORDER BY ascendente:

```
SELECT
* FROM Customers
ORDER BY Country ASC;
```

Exemplo ORDER BY descendente:

```
SELECT
* FROM Customers
ORDER BY Country DESC ;
```

INSERT INTO (INSIRA DENTRO)

É usada para inserir novos registros em uma tabela.

Sintaxe INSERT INTO
É possível escrever a instrução INSERT INTO de duas maneiras:

1. Especifique os nomes das colunas e os valores a serem inseridos:

INSERT INTO table_name (column1, column2, column3, etc)
VALUES (value1, value2, value3, etc);

Ex:
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

2. Se você estiver adicionando valores para todas as colunas da tabela, não será necessário especificar os nomes das colunas na consulta SQL. No entanto, certifique-se de que a ordem dos valores esteja na mesma ordem das colunas da tabela.
Aqui, a sintaxe INSERT INTO seria a seguinte:

INSERT INTO table_name
VALUES (value1, value2, value3, etc);

Ex:
INSERT INTO Cliente
VALUES ('Daniel', 'Passos', 'Idade', 'Documento');
(depente da sua tabela pra você retornar os campos sequencialmente e corretamente).

NULL (NULO)

É um campo sem valor.

Se um campo em uma tabela for opcional, é possível inserir um novo registro ou atualizar um registro sem adicionar um valor a este campo. Em seguida, o campo será salvo com um valor NULL.

Observação: um valor NULL é DIFERENTE de um valor zero ou de um campo que contém espaços (vazio). Um campo com um valor NULL é aquele que foi deixado em branco durante a criação do registro!

Como testar valores NULL?
Não é possível testar os valores NULL com operadores de comparação, como =, <ou <>.

Teremos que usar os operadores IS NULL e IS NOT NULL.

Sintaxe IS NULL
```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

Sintaxe IS NOT NULL
```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

Exemplo IS NULL	SELECT CustomerName, ContactName, Address FROM clientes WHERE Address IS NULL;
Exemplo IS NOT NULL	SELECT CustomerName, ContactName, Address FROM Customers WHERE Address IS NOT NULL;

UPDATE (ATUALIZE) / SET(DEFINA)

É usada para modificar os registros existentes em uma tabela.

Sintaxe de UPDATE

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Nota: Tenha cuidado ao atualizar os registros em uma tabela! Observe a cláusula WHERE na instrução UPDATE. A cláusula WHERE especifica quais registros devem ser atualizados. Se você omitir a cláusula WHERE, todos os registros na tabela serão atualizados!

Tabela UPDATE padrão: A instrução SQL a seguir atualiza o primeiro cliente (CustomerID = 1) com uma nova pessoa de contato e uma nova cidade.	UPDATE Customers SET ContactName = 'Alfredo PéDeAlface', City= 'Batatada' WHERE CustomerID = 1;
UPDATE em vários registros: A seguinte instrução SQL atualizará o ContactName para "Juan" para todos os registros em que country seja "Mexico	UPDATE Customers SET ContactName='Juan' WHERE Country='Mexico';

Aviso de atualização!
Tenha cuidado ao atualizar os registros. Se você omitir a cláusula WHERE, TODOS os registros serão atualizados!

Ex:
UPDATE Customers SET ContactName='Juan';

DELETE (APAGUE)

É usada para excluir registros existentes em uma tabela. (deve ser usada com cuidado pois deletar tabelas inteiras não é algo natural).

Sintaxe DELETE

```
DELETE FROM table_name
WHERE condition;
```

Instrução de exemplo DELETE
A seguinte instrução SQL exclui o cliente "Alfreds Futterkiste" da tabela "Clientes":

Exemplo

```
DELETE FROM Customers
WHERE CustomerName='Alfreds Futterkiste';
```

Apagar todos os registros
É possível excluir todas as linhas de uma tabela sem excluir a tabela.
Isso significa que a estrutura, os atributos e os índices da tabela ficarão intactos:

```
DELETE FROM nome_tabela;
```

Exemplo:

```
DELETE FROM Customers;
```

LIMIT (LIMITE)

É usada para especificar o número de registros a serem retornados.

A cláusula LIMIT é útil em grandes tabelas com milhares de registros. Retornar um grande número de registros pode afetar o desempenho.

Nota: Nem todos os sistemas de banco de dados suportam a cláusula LIMIT. O SQLServer suporta a cláusula SELECT TOP para selecionar um número limitado de registros, enquanto o Oracle usa FETCH FIRST n ROWS ONLY e ROWNUM.

MySQL Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

Exemplo sem WHERE	SELECT * FROM Customers LIMIT 3;
Exemplo com WHERE	 SELECT * FROM Customers WHERE Country='Germany' LIMIT 3;

MIN() e MAX()

As funções SQL MIN () e MAX ()

A função MIN () retorna o menor valor da coluna selecionada.

A função MAX () retorna o maior valor da coluna selecionada.

Sintaxe MIN ()	Sintaxe MAX ()
SELECT MIN(nome_coluna) FROM nome_tabela WHERE condicao;	SELECT MAX(nome_coluna) FROM nome_tabela WHERE condicao;
Ex: SELECT MIN(Preco) AS MaiorPreco FROM Produtos;	Ex: SELECT MAX(Preco) AS MaiorPreco FROM Produtos;

Mais utilizado até o momento com tabela onde tem preços ou valores numerais.

COUNT(), AVG() and SUM() / (CONTAGEM, MÉDIA e SOMA)

A função COUNT () retorna o número de linhas que correspondem a um critério especificado.

COUNT () Sintaxe

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

Ex:

```
SELECT COUNT(ProductID)
FROM Products;
```

A função AVG () retorna valor médio em relação a linha selecionada correspondente a um critério especificado.

Sintaxe AVG ()

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

Ex:

```
SELECT AVG(Price)
FROM Products;
```

Nota: Valores NULL são ignorados.

A função SUM () retorna a soma em relação a linha selecionada correspondente a um critério especificado.

Sintaxe SUM ()

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

Ex:

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

SELECT INTO (SELECIONE EM)

SELECT INTO
Copia dados de uma tabela para uma nova tabela.

SELECT INTO Sintaxe

Copie todas as colunas em uma nova tabela:

```
SELECT * INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

Copie apenas algumas colunas em uma nova tabela:

```
SELECT column1, column2, column3 INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

A nova tabela será criada com os nomes e tipos de coluna definidos na tabela antiga. Você pode criar novos nomes de coluna usando a cláusula AS.

Exemplos:

A seguinte instrução de exemplo do SQL cria uma cópia de backup dos clientes:

```
SELECT * INTO CustomersBackup2017
FROM Customers;
```

A seguinte instrução de exemplo do SQL usa a cláusula IN para copiar a tabela em uma nova tabela em outro banco de dados:

```
SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'
FROM Customers;
```

A seguinte instrução SQL copia apenas algumas colunas em uma nova tabela:

```
SELECT CustomerName, ContactName INTO CustomersBackup2017
FROM Customers;
```

A seguinte instrução SQL copia apenas os clientes alemães em uma nova tabela:

```
SELECT * INTO CustomersGermany
FROM clients
WHERE Country = 'Germany';
```

A seguinte instrução SQL copia dados de mais de uma tabela em uma nova tabela:

```
SELECT Customers.CustomerName, Orders.OrderID
INTO CustomersOrderBackup2017
FROM clients
LEFT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

Dica: SELECT INTO também pode ser usado para criar uma nova tabela vazia usando o esquema de outra. Basta adicionar uma cláusula WHERE que faz com que a consulta não retorne dados:

```
SELECT * INTO newtable
FROM the old table
WHERE 1 = 0;
```

INSERT SELECT INTO (INSERIR SELECIONANDO EM)

INSERT INTO SELECT

Copia os dados de uma tabela e os insere em outra tabela.

A instrução INSERT INTO SELECT requer que os tipos de dados nas tabelas de origem e destino correspondam.

Nota: Os registros existentes na tabela de destino não são afetados.

Sintaxe de INSERT INTO SELECT
Copie todas as colunas de uma tabela para outra:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

Copie apenas algumas colunas de uma tabela para outra:

```
INSERT INTO table2 (column1, column2, column3, etc)
SELECT column1, column2, column3, etc
FROM table1
WHERE condition;
```

Exemplos de SQL INSERT INTO SELECT

A seguinte instrução SQL copia "Fornecedores" em "Clientes"
(as colunas que não são preenchidas com dados conterão NULL):

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country
FROM Suppliers;
```

A seguinte instrução SQL copia "Fornecedores" em "Clientes"
(preencha todas as colunas):

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode,
Country)
SELECT SupplierName, ContactName, Address, City, PostalCode, Country
FROM Suppliers;
```

LIKE (ESPECIFIQUE)

É usado em uma cláusula WHERE para pesquisar um padrão especificado em uma coluna.

Existem dois curingas geralmente usados em conjunto com o operador LIKE:

O sinal de porcentagem (%) representa zero, um ou vários caracteres
O sinal de sublinhado (_) representa um único caractere.

O sinal de porcentagem e o sublinhado também podem ser usados em combinações!

LIKE Sintaxe

```
SELECT column1, column2, etc
FROM table_name
WHERE columnN LIKE pattern;
```

Descrição do operador LIKE

WHERE CustomerName LIKE 'a%'	Encontra todos os valores que começam com "a"
WHERE CustomerName LIKE '% a'	Encontra todos os valores que terminam com "a"
WHERE CustomerName LIKE '% ou%'	Encontra quaisquer valores que tenham "ou" em qualquer posição
WHERE CustomerName LIKE '_r%'	Encontra qualquer valor que tenha "r" na segunda posição
WHERE CustomerName LIKE 'a_%'	Encontra qualquer valor que comece com "a" e tenha pelo menos 2 caracteres de comprimento
WHERE CustomerName LIKE 'a __%'	Encontra qualquer valor que comece com "a" e tenha pelo menos 3 caracteres de comprimento
WHERE ContactName LIKE 'a% o'	Encontra todos os valores que começam com "a" e terminam com "o"

Ex:
A seguinte instrução SQL seleciona todos os clientes com um CustomerName começando com "a":

```
SELECT
* FROM Customers
WHERE CustomerName LIKE 'a%';
```

A seguinte instrução SQL seleciona todos os clientes com um CustomerName terminando com "a":

```
SELECT
* FROM Customers
WHERE CustomerName LIKE '%a';
```

GROUP BY (AGRUPAR POR)

Agrupa linhas que têm os mesmos valores em linhas de resumo, como "encontre o número de clientes em cada país".

A instrução GROUP BY é frequentemente usada com funções agregadas (COUNT (), MAX (), MIN (), SUM (), AVG ()) para agrupar o conjunto de resultados por uma ou mais colunas.

Sintaxe de GROUP BY

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Ex:

A seguinte instrução SQL lista o número de clientes em cada país:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

A seguinte instrução SQL lista o número de clientes em cada país, classificados de cima para baixo:

```
SELECT COUNT (customer ID), country
FROM clients
GROUP BY Country
ORDER BY COUNT (CustomerID) DESC;
```

Exemplo GROUP BY com JOIN
A seguinte instrução SQL lista o número de pedidos enviados por cada remetente:

```
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
LEFT JOIN Shippers
ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

HAVING (TENDO)

HAVING foi adicionada ao SQL porque a palavra-chave WHERE não pode ser usada com funções agregadas.

HAVING Sintaxe

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

Ex:

A instrução SQL a seguir lista o número de clientes em cada país. Incluir apenas países com mais de 10 clientes:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 10;
```

A seguinte instrução SQL lista o número de clientes em cada país, classificados de cima para baixo (inclua apenas países com mais de 5 clientes):

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

IN (NO)

Permite que você especifique vários valores em uma cláusula WHERE.

O operador IN é uma abreviação para várias condições OR.

Sintaxe IN

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, etc);
```

OU

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECIONE DECLARAÇÃO);
```

Ex:

A seguinte instrução SQL seleciona todos os clientes que NÃO estão localizados na "Alemanha", "França" ou "Reino Unido":

```
SELECT
* FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

A seguinte instrução SQL seleciona todos os clientes que são dos mesmos países que os fornecedores:

```
SELECT
* FROM Customers
WHERE Country IN (SELECT Country FROM Suppliers);
```

BETWEEN

Seleciona valores dentro de um determinado intervalo. Os valores podem ser números, texto ou datas.

O operador BETWEEN é inclusivo: os valores inicial e final são incluídos.

BETWEEN Sintaxe

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Exemplo BETWEEN
A seguinte instrução SQL seleciona todos os produtos com um preço entre 10 e 20:

```
SELECT
* FROM Products
WHERE Price BETWEEN 10 AND 20;
```

Exemplo NOT BETWEEN
Para exibir os produtos fora da faixa do exemplo anterior, use NOT BETWEEN:

```
SELECT
* FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

Exemplo BETWEEN com IN
A seguinte instrução SQL seleciona todos os produtos com um preço entre 10 e 20. Além disso; não mostre produtos com um CategoryID de 1,2 ou 3:

```
SELECT
* FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
```

ALIAS

São usados para dar a uma tabela ou coluna de uma tabela um nome temporário.

Os apelidos costumam ser usados para tornar os nomes das colunas mais legíveis.

Um alias existe apenas durante essa consulta.

Um alias é criado com a palavra-chave AS.

Sintaxe da coluna de ALIAS

```
SELECT column_name AS alias_name
FROM table_name;
```

Sintaxe da tabela de ALIAS

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

JOINS (JUNTAR)

Uma cláusula JOIN é usada para combinar linhas de duas ou mais tabelas, com base em uma coluna relacionada entre elas.

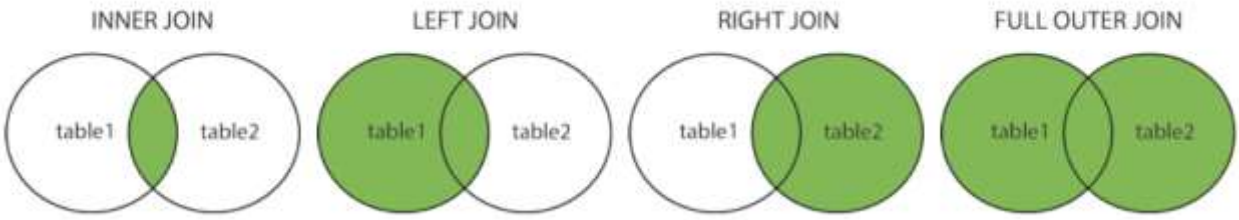
Diferentes tipos de SQL JOINs
Aqui estão os diferentes tipos de JOINs em SQL:

INNER JOIN: Retorna registros que possuem valores correspondentes em ambas as tabelas

LEFT JOIN: Retorna todos os registros da tabela da esquerda e os registros correspondentes da tabela da direita

RIGHT JOIN: Retorna todos os registros da tabela da direita e os registros correspondentes da tabela da esquerda

FULL JOIN: Retorna todos os registros quando há uma correspondência na tabela da esquerda ou da direita



Ex: JOIN

```
SELECT
  cancoes.nome,
  generos.nome
FROM cancoes
JOIN generos
ON cancoes.id = generos.id;
```

JOIN é a mesma coisa que digitar INNER JOIN.

INNER JOIN (JUNÇÃO INTERNA)

Seleciona registros que possuem valores correspondentes em ambas as tabelas.

Sintaxe INNER JOIN

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

Ex:

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Ex:

```
SELECT
  cancoes.nome,
  generos.nome
FROM cancoes
INNER JOIN generos
ON cancoes.id = generos.id;
```

Ex:

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

Nota: A palavra-chave INNER JOIN seleciona todas as linhas de ambas as tabelas, desde que haja uma correspondência entre as colunas. Se houver registros na tabela "Pedidos" que não tenham correspondências em "Clientes", esses pedidos não serão mostrados!

LEFT JOIN (JUNÇÃO PRIORIZANDO ESQUERDA)

Retorna todos os registros da tabela esquerda (tabela1) e os registros correspondentes da tabela direita (tabela2). O resultado é 0 registros do lado direito, se não houver correspondência.

Sintaxe de LEFT JOIN

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Ex:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Nota: A palavra-chave LEFT JOIN retorna todos os registros da tabela da esquerda , mesmo se não houver correspondências na tabela da direita.

É a mais usada já que você pode mesmo numa situação de prioridade da direita alternar as tabelas para usar join pela esquerda.

RIGHT JOIN (JUNÇÃO PRIORIZANDO DIREITA)

Retorna todos os registros da tabela direita (tabela2) e os registros correspondentes da tabela esquerda (tabela1). O resultado é 0 registros do lado esquerdo, se não houver correspondência.

Sintaxe de RIGHT JOIN

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Ex:

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Pedidos
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

Observação: em alguns bancos de dados, RIGHT JOIN é denominado RIGHT OUTER JOIN.

Nota: A palavra-chave RIGHT JOIN retorna todos os registros da tabela da direita (Funcionários), mesmo se não houver correspondências na tabela da esquerda (Pedidos).