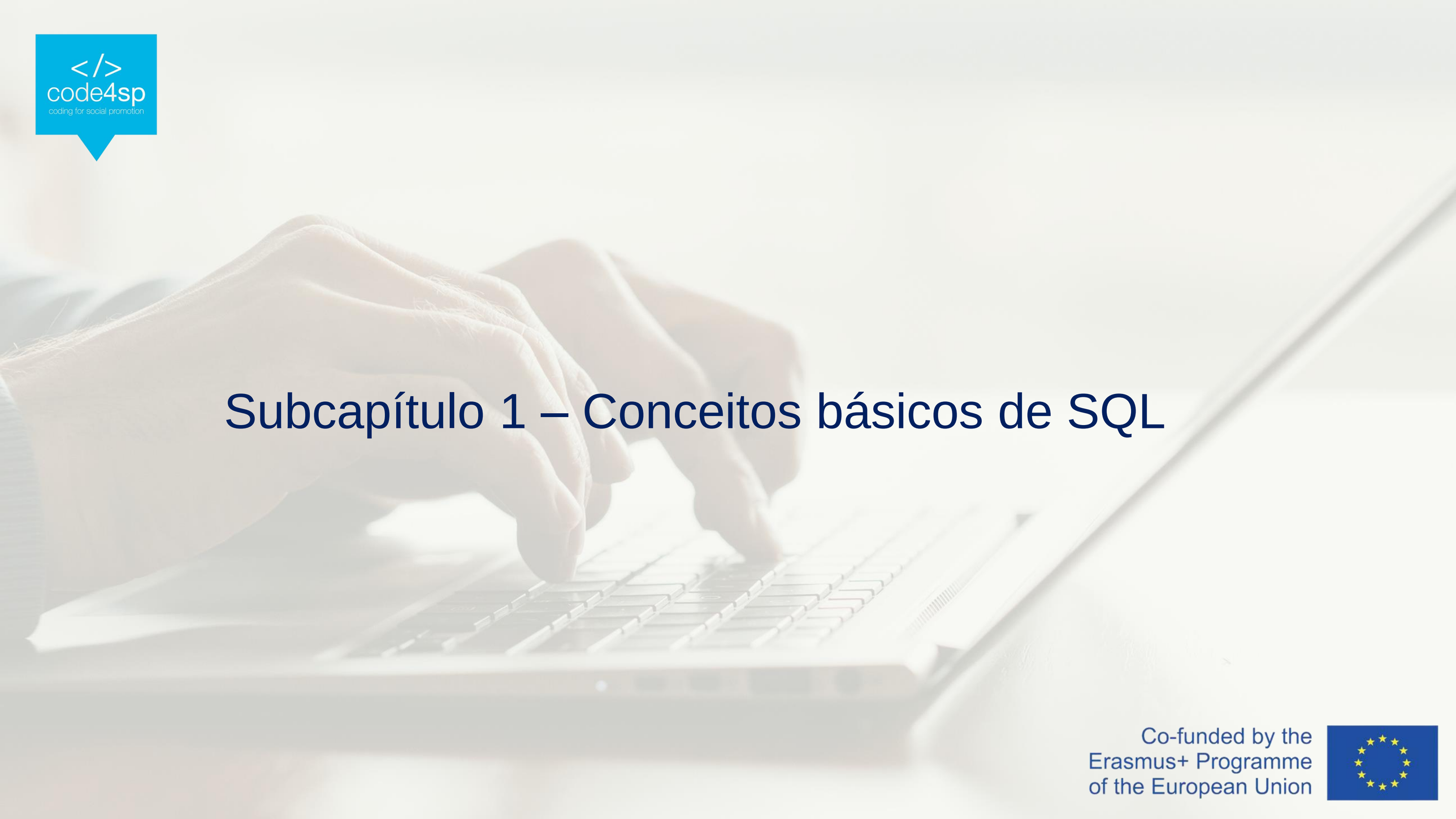
Co-funded by the
Erasmus+ Programme
of the European Union

code4sp
coding for social promotion

Materiais de Formação SQL
Subcapítulo 1 – Conceitos básicos de SQL

WP3: Materiais de Formação do Code4SP

Preparado por CITIZENS IN POWER



Subcapítulo 1 – Conceitos básicos de SQL

Co-funded by the
Erasmus+ Programme
of the European Union

SQL- Introdução

- SQL (*Structured Query Language*) é uma linguagem informática para **armazenamento, manipulação e recuperação de dados armazenados numa base de dados relacional.**

Uma base de dados relacional é uma coleção de itens de dados com relações pré-definidas entre eles.

Algumas das suas muitas aplicações são:

- Permitir aos utilizadores o **acesso a dados** nos sistemas de gestão de bases de dados relacionais.
- Permitir aos utilizadores a **descrição dos dados**.
- Permitir aos utilizadores **definirem os dados** numa base de dados e manipularem esses dados.

Vamos experimentar?
https://www.w3schools.com/html/tryit.asp?filename=tryhtml_intro

Co-funded by the Erasmus+ Programme of the European Union

A síntaxe da SQL

A maioria das ações a realizar numa base de dados são feitas com instruções SQL

Uma instrução de SQL é composta por uma sequência de palavras-chave, identificadores, etc., e finalizada com uma semi-vírgula (;).

Exemplo:

```
SELECT emp_name, hire_date, salary FROM employees WHERE salary > 5000;
```

Para melhor legibilidade, a mesma instrução pode ser escrita da seguinte forma:

```
SELECT emp_name, hire_date, salary
FROM employees
WHERE salary > 5000;
```

Co-funded by the
Erasmus+ Programme
of the European Union

O uso de maiúsculas e minúsculas em SQL

O uso de letras maiúsculas ou minúsculas é **indiferente** em SQL, o que significa que 'SELECT' é o mesmo que 'select'.

- Considere a seguinte instrução de SQL que faculta registos de uma tabela "Employees":

```
SELECT emp_name, hire_date, salary FROM employees;
```

- A mesma instrução pode também ser escrita da seguinte forma:

```
select emp_name, hire_date, salary from employees;
```

*** Contudo, as bases de dados e nomes de tabelas podem ser sensíveis ao uso de maiúsculas e minúsculas dependendo do sistema operativo. De maneira geral, as plataformas Unix e Linux são sensíveis a maiúsculas e minúsculas, e as plataformas Windows não.**

Co-funded by the Erasmus+ Programme of the European Union

O comando ‘Select’ em SQL

O comando ‘SELECT’ **seleciona ou obtém os dados de uma ou mais tabelas.**

Este comando pode ser usado para obter a informação de todas as linhas de uma tabela de uma vez só, ou para selecionar apenas as linhas que correspondem a uma condição específica ou a uma combinação de condições.

- *Select All from Table:* devolve todas as filas da tabela dos Empregados.

```
SELECT * FROM employees;
```

- *Select Specific Columns from Table:* devolve todas as filas das colunas especificadas

```
SELECT emp_id, emp_name, hire_date, salary
FROM employees;
```

Co-funded by the
Erasmus+ Programme
of the European Union

O comando ‘SELECT DISTINCT’

No slide anterior, vimos como selecionar todos os valores de uma tabela ou de colunas específicas.

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Mesa de Clientes – ex. SELECT DISTINCT (Fonte: https://www.w3schools.com/sql/sql_distinct.asp)

- *Escrever uma declaração para selecionar todos os valores da coluna “Country” na tabela “Customers”*

Conseguem notar alguma coisa? Os valores são diferentes ou são os mesmos?

Co-funded by the Erasmus+ Programme of the European Union

O comando ‘SELECT DISTINCT’ em SQL

O comando ‘SELECT DISTINCT’ omite valores duplicados numa consulta.

Síntaxe:

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

Exemplo 1: Select distinct countries from the Customers table

```
SELECT DISTINCT Country FROM Customers;
```

Exemplo 2: List the number of different Customer countries

```
SELECT COUNT(DISTINCT Country) FROM Customers;
```

Note-se que este exemplo não funciona no Firefox, pois o comando ‘COUNT(DISTINCT column_name)’ não é suportado em MS Access.

Co-funded by the Erasmus+ Programme of the European Union

O comando ‘WHERE’ em SQL

Normalmente é necessário selecionar, atualizar ou apagar apenas os registos que correspondem a determinadas condições, como por exemplo utilizadores que pertencem a uma determinada faixa etária, país, etc. .

O comando ‘WHERE’ é usado com ‘select, ‘update’ e ‘DELETE’.

Este comando é usado com ‘SELECT’ para extrair apenas os registos que correspondem a determinadas condições.

Síntaxe:

```
SELECT column_list
FROM table_name
WHERE condition;
```

Co-funded by the
Erasmus+ Programme
of the European Union

O comando ‘WHERE’ em SQL

Exemplo 1: Select all employees from the Employees table whose salary is greater than 7000

```
SELECT * FROM employees WHERE salary > 7000;
```

Exemplo 2: Select all employees with department id =1:

```
SELECT * FROM employees WHERE dept_id=1;
```

O comando ‘WHERE’ apenas filtra e exclui os dados indesejados.

Co-funded by the
Erasmus+ Programme
of the European Union

O comando ‘WHERE’ em SQL

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Tabela “Operators” com o comando ‘WHERE’ (Fonte: https://www.w3schools.com/sql/sql_where.asp)

Exercícios:

- Selecionar todos os empregados da tabela de empregados cujo salário seja inferior a 7000
- Selecionar todos os empregados da tabela de empregados cujo dept_id é igual a 5

Co-funded by the Erasmus+ Programme of the European Union

Os operadores ‘AND’, ‘OR’ e ‘NOT’ em SQL

O comando ‘WHERE’ pode ser usado com os operadores ‘AND’, ‘OR’, e ‘NOT’.

O **operador ‘AND’** exibe um registo caso as condições envolvidas sejam verdadeiras. .

Síntaxe:

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

Exemplo: Selecionar todos os campos da tabela de Clientes onde o país é “Germany" e a cidade é "Berlin".

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

Co-funded by the Erasmus+ Programme of the European Union

Os operadores ‘AND’, ‘OR’ e ‘NOT’ em SQL

O **operador ‘OR**’ exibe um registo caso as condições envolvidas sejam verdadeiras.

Síntaxe:

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

Exemplo 1: Selecionar todos os campos da tabela de Clientes onde a cidade é "Berlin" ou "München".

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

Exemplo 2: Selecionar todos os campos da tabela de Clientes onde o país é “Germany” ou “Spain”.

```
SELECT * FROM Customers
WHERE Country='Germany' OR Country='Spain';
```

Co-funded by the
Erasmus+ Programme
of the European Union

Os operadores ‘AND’, ‘OR’ e ‘NOT’ em SQL

O **operador** ‘**NOT**’ exibe um registo caso as condições envolvidas não sejam verdade.

Síntaxe:

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

Exemplo: Selecionar todos os campos da tabela “Customers” em que o valor no campo “Country” não seja “Germany”.

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

Co-funded by the
Erasmus+ Programme
of the European Union

Os operadores ‘AND’, ‘OR’ e ‘NOT’ em SQL

Conjugar os operadores ‘AND’, ‘OR’ e ‘NOT’

Exemplo 1: Selecionar todas as linhas da tabela “Customers” nas quais o campo “Country” seja “Germany” e o campo ‘City’ seja “Berlin” ou “München”

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

Exemplo 2: Selecionar todas as linhas da tabela “Customers” nas quais o campo “Country” não seja “Germany” nem “USA”

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

Co-funded by the
Erasmus+ Programme
of the European Union

O comando ‘ORDER BY’ em SQL

O comando ‘ORDER BY’ apresenta os resultados requeridos por ordem crescente ou decrescente. O comando ORDER BY ordena os resultados em ordem crescente por defeito. De forma a organizar os resultados por ordem decrescente, use ‘DESC’

Síntaxe:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Co-funded by the
Erasmus+ Programme
of the European Union

O comando ‘ORDER BY’ em SQL

Nesta bateria de exemplos, usaremos as tabelas de clientes que poderão ser acedidas aqui.

Exemplo 1: Selecionar todos os clientes da tabela “Customers” e organizá-los por coluna “Country”.

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

Exemplo 2: Selecionar todos os clientes da tabela e organizá-los por coluna “Country” em ordem decrescente.

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

Co-funded by the
Erasmus+ Programme
of the European Union

O comando ‘ORDER BY’ em SQL

Exemplo 3: Selecionar todos os clientes da mesma tabela e organizá-los por “Country” e “Customer Name”.

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

* Neste exemplo, os dados são, inicialmente, organizados por “Country”. Contudo, se existirem linhas com o mesmo valor no campo “Country”, estas serão organizadas por “Customer Name”.

Exemplo 4: Selecionar todos os clientes da mesma tabela e organizar os dados por “Country” em ordem crescente, e por “Customer Name” em ordem decrescente.

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

Co-funded by the Erasmus+ Programme of the European Union

O comando ‘ORDER BY’ em SQL

Exercícios:

- Selecione todos os clientes da tabela de Clientes e ordene-os pela coluna País por ordem ascendente
- Selecione todos os clientes da tabela de Clientes e ordene-os por ordem decrescente por Nome do Cliente
- Selecione todos os clientes da tabela Clientes e ordene-os por ordem decrescente por Cidade e ordem ascendente por Nome do Cliente

Co-funded by the
Erasmus+ Programme
of the European Union

O comando ‘INSERT INTO’ em SQL

O comando ‘INSERT INTO’ insere novos registos numa tabela. De forma a que o Código seja processado corretamente, os nomes das colunas e os valores a ser inseridos deverão ser especificados.

Síntaxe:

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

Exemplo: para acrescentar um novo registo à tabela “Customers”, use a seguinte instrução:

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

Co-funded by the
Erasmus+ Programme
of the European Union

Valores ‘NULL’ em SQL

Um campo com um valor ‘NULL’ é um campo **sem um valor**. Se um campo de uma tabela for opcional, é possível inserir um novo registo ou atualizar um registo sem acrescentar um novo valor a este campo.

*** Um valor ‘NULL’ é diferente de um valor zero ou de um campo que contém espaços. Um campo com um valor ‘NULL’ é um campo que foi deixado em branco durante a criação do registo!**

É impossível testar para obter valores ‘NULL’ com operadores de comparação, como ‘=’, ‘<’ ou ‘<>’.

Em vez destes operadores, teremos de usar os operadores **‘IS NULL’** e **‘IS NOT NULL’**.

Co-funded by the
Erasmus+ Programme
of the European Union

Valores ‘NULL’ em SQL

Síntaxe IS NULL :

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

Síntaxe IS NOT NULL :

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

Co-funded by the
Erasmus+ Programme
of the European Union

Valores ‘NULL’ em SQL

Exemplo de IS NULL: Seleciona todos os clientes com valores ‘NULL’ (ex., valores vazios) na coluna ‘Address’:

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

Exemplo de IS NULL: Seleciona todos os clientes com valores ‘NOT NULL’ (ex., valores que não estejam vazios) na coluna ‘Address’:

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

Exercícios:

- Procurar valores nulos nas colunas PostalCode e City
- Procure por valores não vazios na coluna ContactName

Co-funded by the Erasmus+ Programme of the European Union

O comando ‘UPDATE’ em SQL

O comando ‘UPDATE’ altera os registos existentes de uma tabela.

Síntaxe:

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

*** É necessário algum cuidado ao atualizar os registos de uma tabela. Note-se o uso do comando ‘WHERE’ na instrução ‘UPDATE’. O comando ‘WHERE’ especifica que registos devem ser atualizados!**

Exemplo: Para atualizar ‘CustomerID=1’ da tabela “Customers”, utilize a seguinte instrução:

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

Co-funded by the Erasmus+ Programme of the European Union

O comando ‘UPDATE’ em SQL

O comando WHERE determina quantos registos serão atualizados.

Exemplo: Para atualizar ‘CustomerID=1’ da tabela “Customers”, utilize a seguinte instrução.

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

Co-funded by the
Erasmus+ Programme
of the European Union

O comando ‘DELETE’ em SQL

O comando ‘DELETE’ **apaga** os registos da tabela existentes.

Síntaxe:

DELETE FROM table_name WHERE condition;

Exemplo:

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

É possível eliminar todas as linhas de uma tabela sem eliminar a tabela. Quer isto dizer que a estrutura, os atributos e índices da tabela ficarão intactos:

DELETE FROM table_name;

*** Note-se que é necessária alguma caução aquando apagar registos da tabela! Repare no uso do comando ‘WHERE’ na instrução ‘DELETE’. O comando ‘WHERE’ determina que registo(s) deve(m) ser eliminado(s).**

Se o comando ‘WHERE’ for omitido, todos os registos da tabela serão eliminados!

Co-funded by the Erasmus+ Programme of the European Union

O comando ‘SELECT TOP’ em SQL

O comando ‘SELECT TOP’ é usado para **determinar o número de registos que serão acedidos.**

Síntaxe SQL Server/MS Access :

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE condition;
```

Síntaxe MySQL:

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

*** Note-se que nem todos os sistemas de bases de dados suportam o comando ‘SELECT TOP’. O MYSQL suporta o comando ‘LIMIT’ para selecionar um número limitado de registos, ao passo que o Oracle usa ‘FETCH FIRST n ROWS ONLY’ e ‘ROWNUM’.**

Co-funded by the Erasmus+ Programme of the European Union

O comando ‘SELECT TOP’ em SQL

Síntaxe Oracle 12 :

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s)
FETCH FIRST number ROWS ONLY;
```

Para aceder a uma síntaxe para versões antigas do Oracle, clique aqui.

Exemplo: Use a seguinte instrução para selecionar os primeiros três registos da tabela ‘Customers’ em SQLServer/MS Access:

```
SELECT TOP 3 * FROM Customers;
```

MySQL: SELECT * FROM Customers LIMIT 3;

Oracle: SELECT * FROM Customers FETCH FIRST 3 ROWS ONLY;

Co-funded by the Erasmus+ Programme of the European Union

O comando ‘SELECT TOP’ em SQL

TOP PERCENT

Para selecionar 50% dos registos da tabela “Customers”:

SQL Server/MS Access: SELECT TOP 50 PERCENT * FROM Customers;

Oracle: SELECT * FROM Customers FETCH FIRST 50 PERCENT ROWS ONLY;

Exemplos do comando ‘ADD a WHERE’

No exemplo seguinte, iremos selecionar os primeiros três registos da tabela “Customers”, em que o campo “Country” seja o valor “Germany” para SQL Server/MS Access:

SQL Server/MS Access: SELECT TOP 3 * FROM Customers WHERE Country='Germany’;

- MySQL: SELECT * FROM Customers WHERE Country='Germany’ LIMIT 3;
- Oracle: SELECT * FROM Customers WHERE Country='Germany’ FETCH FIRST 3 ROWS ONLY;

Co-funded by the Erasmus+ Programme of the European Union

A função ‘MIN’ e ‘MAX’ em SQL

A função ‘MIN()’ apresenta os **valores mais baixos** das colunas selecionadas.

Síntaxe do MIN()

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

A função ‘MAX()’ apreesenta os **valores mais altos** das colunas selecionadas

Síntaxe MAX()

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

Co-funded by the
Erasmus+ Programme
of the European Union

A função ‘MIN’ e ‘MAX’ em SQL

Nos exemplos seguintes, usaremos uma tabela de produtos que pode ser encontrada aqui.

Exemplo 1: Encontrar o preço do produto mais barato

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

Exemplo 2: Encontrar o preço do produto mais caro

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

Co-funded by the
Erasmus+ Programme
of the European Union

As funções ‘COUNT’, ‘AVG’ E ‘SUM’ EM SQL

A função ‘COUNT’ apresenta o número de linhas que correspondem a critérios específicos**.**

Síntaxe COUNT() :

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

A função ‘AVG()’ apresenta **o valor médio de uma coluna numérica.**

Síntaxe AVG() :

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

Co-funded by the
Erasmus+ Programme
of the European Union

As funções ‘COUNT’, ‘AVG’ E ‘SUM’ EM SQL

A função ‘SUM()’ apresenta **a soma total de uma coluna numérica.**

Síntaxe SUM():

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

Co-funded by the
Erasmus+ Programme
of the European Union

As funções ‘COUNT’, ‘AVG’ E ‘SUM’ EM SQL

Nos exemplos seguintes, usar-se-á a mesma tabela estamos a utilizar a mesma tabela no SQL Min e Max, a tabela de Produtos::

Exemplo COUNT(): Para executar uma consulta para enocntrar o número de produtos

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

Exemplo AVG(): Para executar uma pesquisa para encontrar o preço médio de todos os produtos

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

Co-funded by the
Erasmus+ Programme
of the European Union

As funções ‘COUNT’, ‘AVG’ E ‘SUM’ EM SQL

No exemplo seguinte, usar-se-á uma tabela OrdersDetails, que pode ser acedida aqui, de forma a encontrar a soma da “Quantity”.

Exemplo de SUM():

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

Exercícios:

Encontrar a Quantidade média na tabela OrdersDetails;

Encontrar o número de Encomendas na tabela OrdersDetails.

Co-funded by the
Erasmus+ Programme
of the European Union

O operador ‘LIKE’ em SQL

O operador ‘LIKE’ é usado numa instrução ‘WHERE’ para **pesquisar** um padrão específico numa coluna**.**

Síntaxe LIKE:

SELECT column1, column2, ...

FROM table_name

WHERE column LIKE pattern;

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Operadores ‘LIKE’ (Fonte: https://www.w3schools.com/sql/sql_like.asp)

Co-funded by the Erasmus+ Programme of the European Union

O operador ‘LIKE’ em SQL

Exemplo 1: Para selecionar todos os clientes com nome a começar por ‘a’:

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

Exemplo 2: Para selecionar todos os clientes cujo nome termina em ‘a’:

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

Exemplo 3: Para selecionar todos os clientes cujo nome contém ‘or’ em qualquer posição

```
WHERE CustomerName LIKE '%or%';
```

Co-funded by the
Erasmus+ Programme
of the European Union

O operador ‘LIKE’ em SQL

Exemplo 4: Para selecionar todos os nomes de clientes que têm ‘r’ como segunda letra:

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

Exemplo 5: Para selecionar todos os nomes de clientes que comecem com ‘a’ e tenham pelo menos três carateres:

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

Exemplo 6: Para selecionar os nomes de clientes que comecem com ‘a’ e terminem em ‘o’:

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

Co-funded by the
Erasmus+ Programme
of the European Union

Wildcards em SQL

Um wildcard **substitui um ou mais carateres numa cadeia**. É usado com o operador ‘LIKE’.

Para mais *Wildcards* usados em RDBMS, clicar aqui.

Exemplos do % Wildcard

Neste exemplo, selecionamos todos os clientes com uma “City” a começar por “ber”:

```
SELECT * FROM Customers
WHERE City LIKE 'ber%';
```

No exemplo seguinte, selecionamos todos os clientes com uma “City” que contenha “es”:

```
SELECT * FROM Customers
WHERE City LIKE '%es%';
```

Co-funded by the
Erasmus+ Programme
of the European Union

Wildcards em SQL

Exemplos do % Wildcard

Neste exemplo, são selecionados todos os clientes com uma “City” que se inicie por qualquer letra seguida de “ondon”:

```
SELECT * FROM Customers
WHERE City LIKE '_ondon';
```

Para selecionar todos os clientes com uma “City” a iniciar-se com a letra ‘L’, seguida por qualquer carater seguido por “on”:

```
SELECT * FROM Customers
WHERE City LIKE 'L_n_on';
```

Co-funded by the
Erasmus+ Programme
of the European Union

Wildcards em SQL

Exemplos com o Wildcard [charlist]

Para selecionar todos os clientes com uma “City” a iniciar com ‘b’, ‘s’ ou ‘p’:

```
SELECT * FROM Customers
WHERE City LIKE '[bsp]%';
```

No exemplo seguinte, selecionaremos todos os clientes com uma “City” a começar com ‘a’, ‘b’, ou ‘c’:

```
SELECT * FROM Customers
WHERE City LIKE '[a-c]%';
```

Co-funded by the
Erasmus+ Programme
of the European Union

Wildcards em SQL

Exemplos com o wildcard ‘[!charlist]’

O ponto de exclamação mostra carateres que não contêm uma cadeia específica. Por exemplo, queremos selecionar todos os clientes com uma “City” que não começa por ‘b’, ‘s’ ou ‘p’

```
SELECT * FROM Customers
WHERE City LIKE '[!bsp]%';
```

Em alternativa, podemos usar a seguinte instrução:

```
SELECT * FROM Customers
WHERE City NOT LIKE '[bsp]%';
```

Co-funded by the
Erasmus+ Programme
of the European Union

O operador ‘IN’ em SQL

O operador ‘IN’ é usado para especificar **valores múltiplos** numa instrução ‘WHERE’. Pode considerar-se que satisfaz várias condições.

Síntaxe 1:

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

Síntaxe 2:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

Co-funded by the Erasmus+ Programme of the European Union

O operador ‘IN’ em SQL

Suponhamos que temos uma tabela “Customers” que contém as seguintes colunas: “CustomerID”, “CustomerName”, “ContactName”, “Address”, “City”, “PostalCode” and “Country” (ver tabela aqui).

Exemplo 1: Como exemplo, queremos selecionar todos os clientes localizados na Alemanha (‘Germany’), França (‘France’) ou Reino Unido (‘UK’):

```
SELECT * FROM Customers
WHERE Country IN (‘Germany’, ‘France’, ‘UK’)
```

Exemplo 2: Outro exemplo, é selecionar todos os clientes que não estão localizados na Alemanha (‘Germany’), França (‘France’) ou Reino Unido (‘UK’)

```
SELECT * FROM Customers
WHERE Country NOT IN ( ‘Germany’,  ‘France’, ‘UK’)
```

Co-funded by the Erasmus+ Programme of the European Union

O operador ‘IN’ em SQL

Exemplo 3: Consideremos ainda um terceiro exemplo no qual queremos selecionar os clientes que são do mesmo país que os fornecedores (suppliers):

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

Exercícios:

- Selecionar clientes localizados na Cidade de Berlim
- Selecionar clientes localizados nas Cidades de Londres e Madrid
- Selecionar os clientes que não são do mesmo país que os fornecedores.

Co-funded by the Erasmus+ Programme of the European Union

O operador ‘BETWEEN’ em SQL

O operador ‘BETWEEN’ faculta um conjunto de valores dos quais escolher. Os valores podem ser texto, números ou datas. O operador ‘BETWEEN’ inclui os **valores iniciais e finais.**

Síntaxe:

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

Seguem vários exemplos com os seguintes operadores: ‘BETWEEN’, ‘NOT BETWEEN’, ‘BETWEEN’ com ‘IN’, ‘BETWEEN’ e ‘NOT BETWEEN’ com valores em texto e datas ‘BETWEEN’

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Between

Exemplo de BETWEEN: Para selecionar todos os produtos com um intervalo de preço entre 10 e 20

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

Exemplo de NOT BETWEEN : Exibe todos os produtos fora do intervalo definido no exemplo anterior.

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

Exemplo BETWEEN com IN: Seleciona todos os produtos com um intervalo de preços entre 10 e 20 e não mostra os produtos com a “CategoryID” 1, 2 ou 3.

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

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Between

Exemplos de 'BETWEEN' com valores em texto: Seleciona todos os produtos com um "ProductName" entre "Carnarvorn Tigers" e "Mozzarella di Giovanni".

```
SELECT * FROM Products
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

Exemplos de 'NOT BETWEEN' com valores em texto: Seleciona todos os produtos com um "ProductName" que não esteja entre "Carnarvorn Tigers" e "Mozzarella di Giovanni".

```
SELECT * FROM Products
WHERE ProductName NOT BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Between

Exemplo de datas 'BETWEEN': Selecionar todas as encomendas com data entre '01-July-1996' e '31-July-1996' (para encontrar a tabela usada neste exemplo, clicar aqui)

Isto pode ser feito de duas formas, usando um cardinal (#) ou aspas (""):

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;
```

OU

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```

Co-funded by the Erasmus+ Programme of the European Union

SQL *Aliases*

Os aliases **atribuem um nome temporário a uma tabela ou coluna de uma tabela**. Um alias existe apenas durante uma pesquisa e é, normalmente, usado para tornar os nomes das colunas mais legíveis. Um aliasé criado através do uso da palavra-chave **AS**.

Síntaxe para colunas alias:

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

Síntaxe para tabelas alias:

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

Co-funded by the
Erasmus+ Programme
of the European Union

SQL *Aliases*

Colunas Aliases

Analisemos um exemplo que cria dois aliases, um para cada coluna:

```
SELECT CustomerID AS ID, CustomerName AS Customer
FROM Customers;
```

Outro exemplo para criar dois aliases, novamente:

```
SELECT CustomerName AS Customer, ContactName AS [Contact Person]
FROM Customers;
```

*** Note-se que este é colocado entre parêntesis retangulares ([ ]) porque o alias contém espaços. As aspas podem ser usadas em alternativa aos parêntesis.**

Co-funded by the
Erasmus+ Programme
of the European Union

SQL *Aliases*

Podemos também criar um alias com uma ou mais colunas, consideremos o exemplo em baixo:

```
SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ', ' + Country AS Address
FROM Customers;
```

A instrução em cima citada difere em MySQL:

```
SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,', ',Country) AS Address
FROM Customers;
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL *Aliases*

Tabelas Aliases: O exemplo seguinte seleciona todas as encomendas da tabela de clientes com “CustomerID=4” (“Around the Horn”).

Neste exemplo, são usados aliases para encurtar a pesquisa:

```
SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;
```

Uma pesquisa em aliases assemelhar-se-á a isto:

```
SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName
FROM Customers, Orders
WHERE Customers.CustomerName='Around the Horn' AND
Customers.CustomerID=Orders.CustomerID;
```

Co-funded by the
Erasmus+ Programme
of the European Union

A operação ‘JOIN’ em SQL

A operação ‘JOIN’ combina linhas de duas ou mais tabelas tendo por base uma coluna comum em ambas as tabelas. Existem quatro junções (‘joins’) diferentes em SQL:

1. *(INNER) JOIN:* exibe registos com valores equivalentes e mambas as tabelas;
2. *LEFT (OUTER) JOIN:* apresenta todos os registos da tabela da esquerda e os registos correspondentes da tabela da direita;
3. *RIGHT (OUTER) JOIN:* apresenta todos os registos da tabela da direita e os valores correspondentes da tabela da esquerda;
4. *FULL (OUTER) JOIN:* exibe todos os registos quando há uma correspondência na tabela da esquerda ou da direita.

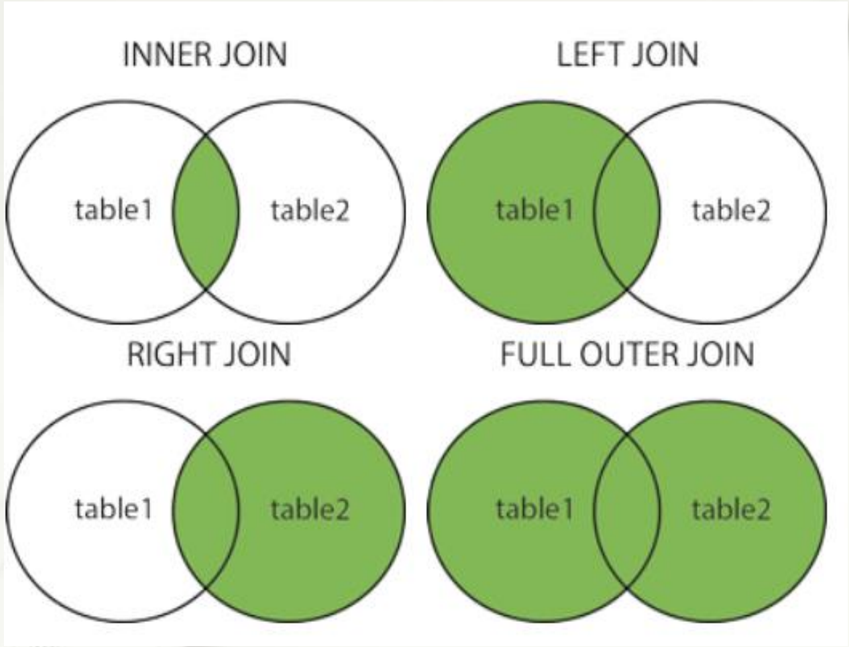
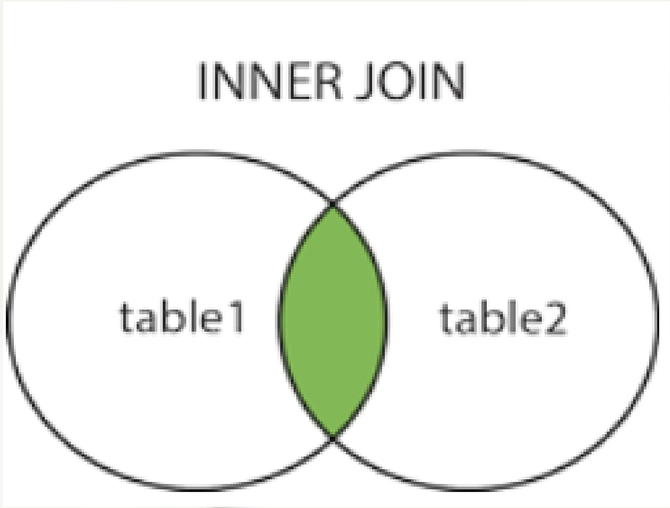


Figura - Diferentes tipos de ‘JOINS’
(Fonte: https://www.w3schools.com/sql/sql_join.asp)

Co-funded by the Erasmus+ Programme of the European Union

INNER JOIN' em SQL

A operação 'INNER JOIN' seleciona registos que têm valores equivalentes em ambas as tabelas.



Inner Join
(Fonte: https://www.w3schools.com/sql/sql_join.asp)

Síntaxe:

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

Neste exemplo, queremos obter os nomes dos clientes e as suas "Order IDs" respetivas:

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

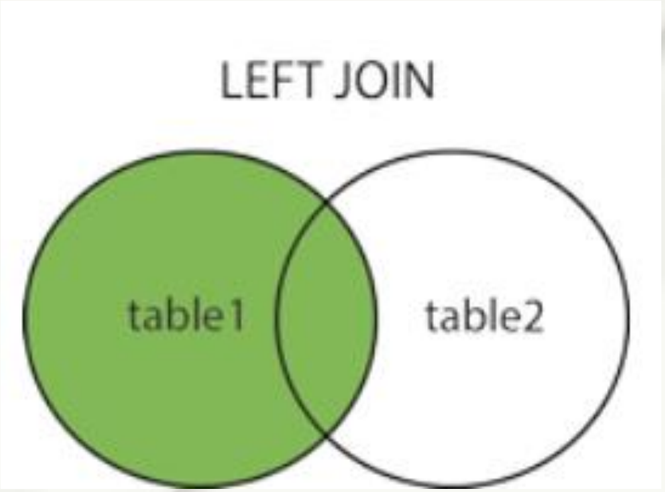
Co-funded by the Erasmus+ Programme of the European Union

‘LEFT JOIN’ em SQL

A operação ‘LEFT JOIN’ exibe todos os registos da tabela da esquerda e os registso equivalentes da tabela da direita.

Síntaxe:

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



Left Join

(Fonte: https://www.w3schools.com/sql/sql_join.asp)

Como exemplo, vamos selecionar todos os clientes e quaisquer encomendas que estes possam ter feito:

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

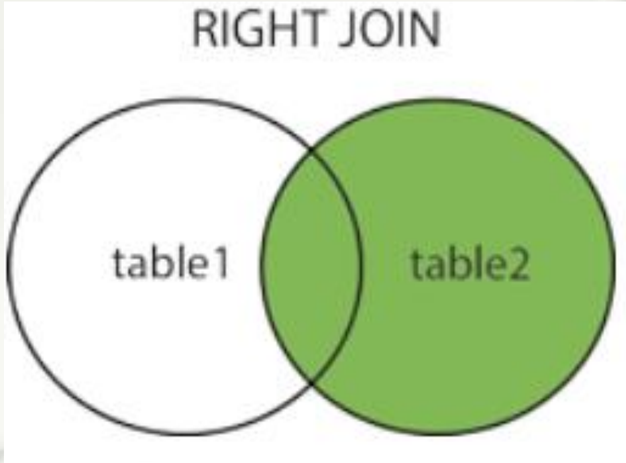
Co-funded by the Erasmus+ Programme of the European Union

‘RIGHT JOIN’ em SQL

A operação ‘RIGHT JOIN’ segue a mesma lógica descrita anteriormente, mas do lado direito em vez do lado esquerdo.

Síntaxe:

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



Right Join

(Fonte: https://www.w3schools.com/sql/sql_join.asp)

O exemplo seguinte exibirá todos os funcionários e quaisquer encomendas que possam ter feito:

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

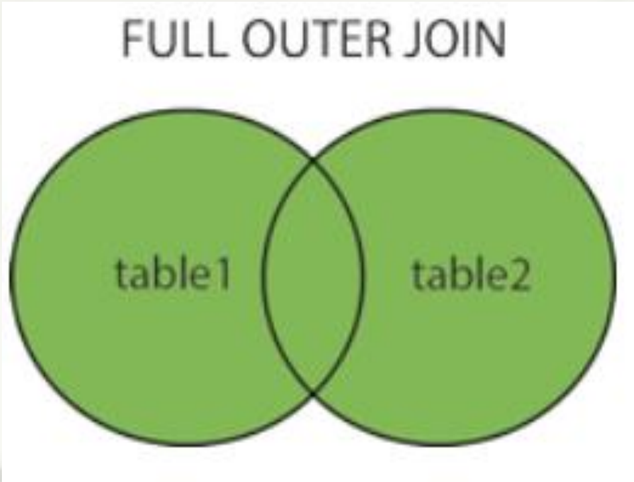
Co-funded by the Erasmus+ Programme of the European Union

‘FULL JOIN’ em SQL

A operação ‘FULL JOIN’ recupera todos os registos quando são encontrados registos correspondentes quern a tabela da direita, quern a tabela da esquerda.

Síntaxe:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```



Full (Outer) Join
(Fonte: https://www.w3schools.com/sql/sql_join.asp)

* **Aqui podemos verificar que esta operação exibe todos os resultados correspondentes de ambas as tabelas, mesmo que estes não sejam encontrados. Neste caso, é atribuído o valor null**.

Co-funded by the Erasmus+ Programme of the European Union

‘SELFJOIN’ em SQL

Um ‘SELF JOIN’ é considerado uma operação ‘JOIN’ normal, mas a tabela é incluída também.

Síntaxe:

```
SELECT column_name(s)
FROM table1 T1, table1 T2    (T1 and T2 are aliases used for the same table)
WHERE condition;
```

Exemplo: Neste exemplo, queremos selecionar clientes da mesma cidade:

```
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

Co-funded by the
Erasmus+ Programme
of the European Union

‘UNION’ em SQL

O operador ‘UNION’ é usado para combinar o conjunto de resultados de duas ou mais instruções ‘SELECT’.

Há alguns **requisitos** para viabilizar um ‘UNION’:

1. Ao utilizar o operador ‘UNION’, todas as instruções ‘SELECT’ devem ter o mesmo número de colunas;
2. As colunas devem ter tipos de dados semelhantes;
3. As colunas devem estar na mesma ordem em todas as instruções ‘SELECT’

Síntaxe:

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

Co-funded by the Erasmus+ Programme of the European Union

‘UNION’ em SQL

Note-se que o operador ‘UNION’ seleciona **apenas valores diferentes por defeito.**

To allow duplicate values, use UNION ALL:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

*** Note-se que os nomes das colunas são normalmente iguais nas duas instruções ‘SELECT’**

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Union

Exemplo 1: obter cidades diferentes

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

Exemplo 2: valores duplicados de ambas as tabelas

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

Exemplo 3: Neste exemplo, através do comando 'WHERE' serão exibidas as diferentes cidades alemãs presents nas tabelas "Customers" e "Suppliers":

```
SELECT City, Country FROM Customers
WHERE Country = 'Germany'
UNION
SELECT City, Country FROM Customers
WHERE Country = 'Germany'
ORDER BY City;
```

Co-funded by the Erasmus+ Programme of the European Union

‘GROUP BY’ em SQL

A instrução ‘GROUP BY’ agrupa linhas com os mesmos valores em linhas resumidas. Suponhamos, por exemplo, que queremos consultar o número de clientes de cada país.

Esta instrução é também frequentemente usada com funções combinadas, tais como ‘COUNT()’, ‘MAX()’, ‘MIN()’, ‘SUM()’, ‘AVG()’, para agrupar os resultados em uma ou mais colunas.

Síntaxe:

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

Co-funded by the
Erasmus+ Programme
of the European Union

‘GROUP BY’ em SQL

Exemplo 1: listar o número de clientes de cada país

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

Exemplo 2: listar novamente o número de clientes em cada país, mas por ordem decrescente.

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

Exemplo 3: listar o número de encomendas (orders) enviados por cada exportador (shipper)

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

Co-funded by the
Erasmus+ Programme
of the European Union

‘HAVING’ em SQL

A instrução ‘HAVING’ foi adicionada à SQL porque o comando ‘WHERE’ não pode ser usado com funções combinadas.

Síntaxe:

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

Exemplo: listar o número de clientes de cada país, mas também queremos incluir países que têm mais do que cinco clientes.

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

Co-funded by the
Erasmus+ Programme
of the European Union

‘HAVING’ em SQL

Exemplo 1: listar os funcionários de “Davolio” ou “Fuller” que tenham registado encomendas mais do que 25 vezes:

```
SELECT Employees.LastName,
COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID =
Employees.EmployeeID)
WHERE LastName = ‘Davolio’ OR LastName =
‘Fuller’
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

Exemplo 2: lista os funcionários que registaram mais de dez encomendas:

```
SELECT Employees.LastName,
COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID =
Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

Co-funded by the Erasmus+ Programme of the European Union

‘SELECT INTO’ em SQL

A instrução ‘SELECT INTO’ copia dados de uma tabela para uma nova tabela.

Síntaxe para copiar todas as colunas para uma nova tabela:

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

Síntaxe para copiar apenas algumas colunas para uma nova tabela:

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

Co-funded by the
Erasmus+ Programme
of the European Union

‘SELECT INTO’ em SQL

Exemplo de como criar uma cópia de segurança da tabela “Customers”:

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

Exemplo do uso do comando ‘IN’ para copiar a tabela para uma tabela nova em outra base de dados:

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

Exemplo de como copiar apenas algumas colunas para uma tabela nova:

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

Co-funded by the
Erasmus+ Programme
of the European Union

‘SELECT INTO’ em SQL

Este exemplo copia apenas os fornecedores alemães para a tabela “Customers”:

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

Este exemplo copia dados de múltiplias tabelas numa nova tabela:

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

SELECT INTO pode também ser usado para criar uma nova tabela em branco, usando o mesmo esquema de uma outra.

Para isso, adicionamos uma condição WHERE que não gera dados:

```
SELECT * INTO newtable
FROM oldtable
WHERE 1 = 0;
```

Co-funded by the Erasmus+ Programme of the European Union

‘INSERT INTO SELECT’ em SQL

A inscrição ‘INSERT INTO SELECT’ copia dados de uma tabela e insere-os em outra tabela. Isto requere que o tipo de dados da fonte correspondam ao tipo de dados da tabela para onde estes serão copiados.

Síntaxe para copiar todas as colunas de uma tabela para outra:

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

Síntaxe para copiar apenas algumas colunas de uma tabela para outra:

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

Co-funded by the Erasmus+ Programme of the European Union

‘INSERT INTO SELECT’ em SQL

Exemplo 1: copia a tabela “Suppliers” para a tabela “Customers” (note-se que as colunas sem dados serão registadas com valores ‘null’):

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

Exemplo 2: copia a tabela “Suppliers” para a tabela “Customers” para preencher todas as colunas:

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

Exemplo 3: copia apenas os fornecedores alemães para a tabela “Customers”:

```
INSERT INTO Customers (CustomerName,
City, Country)
SELECT SupplierName, City, Country FROM
Suppliers
WHERE Country='Germany';
```

Co-funded by the
Erasmus+ Programme
of the European Union

‘CASE’ em SQL

A instrução ‘CASE’ repassa uma série de condições e apresenta um valor quando a primeira condição é atendida. Pense nesta instrução como uma instrução ‘IF’ e depois ‘ELSE. Quando uma condição é verificada como verdadeira, esta instrução pára de procurar. Se nenhuma condição for dada como verdadeira, esta instrução apresentará como resultado o valor na instrução ‘ELSE’.

Note-se que se não existir uma instrução ‘ELSE’ e nenhuma condição for dada como verdadeira, o resultado será o valor ‘NULL’.

Síntaxe:

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

Co-funded by the Erasmus+ Programme of the European Union

‘CASE’ em SQL

Exemplo 1: analisa uma série de condições e apresenta um valor quando a primeira condição for atendida.

```
SELECT OrderID, Quantity,
CASE
        WHEN Quantity > 30 THEN 'The quantity is greater than 30'
        WHEN Quantity = 30 THEN 'The quantity is 30’
        ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

Exemplo 2: ordenar os clientes por cidade (city). Note-se que se o campo “City” for ‘NULL’ será ordenado por país (country).

```
SELECT CustomerName, City, Country
FROM Customers
ORDER BY
      (CASE
            WHEN City IS NULL THEN Country
            ELSE City
END);
```

Co-funded by the Erasmus+ Programme of the European Union

Funções ‘NULL’ em SQL

As funções ‘NULL’ incluem: ‘IFNULL()’, ‘ISNULL()’, ‘COALESCE()’ e ‘NVL()’.

Digamos que a coluna “UnitsOnOrder” é opcional e pode conter valores ‘NULL’.

Exemplo:

```
SELECT ProductName, UnitPrice * (UnitsInStock + UnitsOnOrder)
FROM Products;
```

Aqui podemos ver que se algum dos valores de “UnitsOnOrder” for ‘null’, o resultado também será ‘null’.

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Null Functions

Em MySQL, Podemos usar a função ‘ISNULL()’ que permite obter um valor alternative se uma expressão for ‘null’:

```
SELECT ProductName, UnitPrice * (UnitsInStock + IFNULL(UnitsOnOrder, 0))
FROM Products;
```

Ou podemos usar a função ‘COALESCE()’:

```
SELECT ProductName, UnitPrice * (UnitsInStock + COALESCE(UnitsOnOrder, 0))
FROM Products;
```

Co-funded by the
Erasmus+ Programme
of the European Union

SQL Null Functions

Em SQL Server, a função ‘ISNULL()’ faz o mesmo que em MySQL:

```
SELECT ProductName, UnitPrice * (UnitsInStock + IFNULL(UnitsOnOrder, 0))
FROM Products;
```

Em MS Access, a função ‘IsNull()’ apresenta ‘TRUE(-1)’ se a expressão for um valor ‘null’, em caso contrário, apresenta ‘FALSE (0)’:

```
SELECT ProductName, UnitPrice * (UnitsInStock + IIF(IsNull(UnitsOnOrder), 0, UnitsOnOrder))
FROM Products;
```

Em Oracle, a função ‘NVL()’ faz a mesma coisa:

```
SELECT ProductName, UnitPrice * (UnitsInStock + NVL(UnitsOnOrder, 0))
FROM Products;
```

Co-funded by the Erasmus+ Programme of the European Union

Comentários em SQL

Os comentários em SQL **explicam as seções das instruções SQL ou previnem a sua execução.**

Os comentários de uma linha começam com - - (dois travessões):

```
--Select all:
SELECT * FROM Customers;
```

Ou podem ser usados da seguinte forma para ignorar o fim da linha:

```
SELECT * FROM Customers -- WHERE City='Berlin';
```

Ou para ignorar uma instrução:

```
--SELECT * FROM Customers;
SELECT * FROM Products;
```

*** Nota que os exemplos desta secção não são suportados pelo Firefox, nem Microsoft Edge, ambas base de dados do Microsoft Access. Geralmente, os comentários não são suportados pelas bases de dados do Microsoft Access.**

Co-funded by the Erasmus+ Programme of the European Union

Comentários em SQL

Comentários com linhas múltiplas começam por /* e terminam em */. Qualquer texto escrito entre estes dois símbolos será ignorado.

Exemplo 1:

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```

Para ignorar parte de uma instrução também podemos usar /**/

Exemplo 2:

```
SELECT CustomerName, /*City,*/ Country FROM Customers;
```

Co-funded by the
Erasmus+ Programme
of the European Union

Vamos praticar!

Aprendemos muitas coisas novas. É tempo de as colocar em prática!

Clica aqui.

Co-funded by the Erasmus+ Programme of the European Union

OBRIGADO!

PRÓXIMO CAPÍTULO: SQL Database

Co-funded by the
Erasmus+ Programme
of the European Union