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

code4sp
coding for social promotion

SQL Trainer Materials
Subchapter 1 – The basics of SQL

WP3: Code4SP Training Materials

Prepared by: CITIZENS IN POWER

Subchapter 1: The basics of SQL

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

SQL Introduction

- SQL (Structured Query Language) is a computer language for storing, **manipulating and retrieving data stored in a relational database.**

A relational database is a collection of data items with pre-defined relationships between them.

Some of its many applications are:

- Allowing users to **access data** in the relational database management systems.
- Allowing users to **describe the data**.
- Allowing users to **define the data** in a database and manipulate that data.

Shall we give it a try?
https://www.w3schools.com/html/tryit.asp?filename=tryhtml_intro

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

SQL Syntax

Most of the actions you need to perform on a database are done with SQL statements

An SQL statement is composed of a sequence of keywords, identifiers etc. terminated by a semicolon (;)

Example:

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

For better readability, you can also write the same statement, as follows:

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

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

Case Sensitivity in SQL

SQL keywords are **case-insensitive**, which means SELECT is the same as select

- Consider another SQL statement that retrieves records from the Employees table:

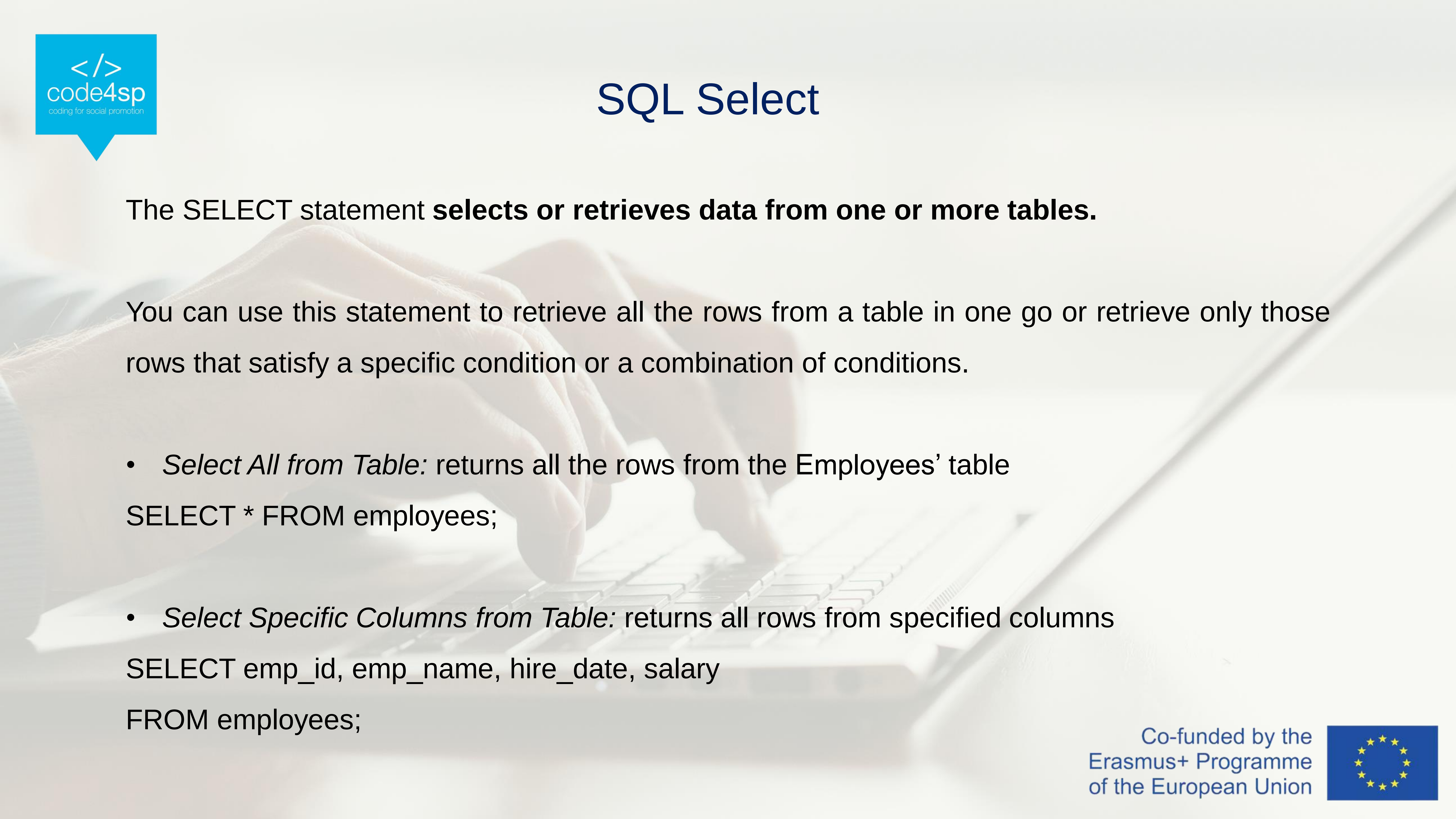
```
SELECT emp_name, hire_date, salary FROM employees;
```

- The same statement can also be written as follows:

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

*** Note that this depends on the operating system, i.e. Unix or Linux platforms are case-sensitive whereas Windows platforms aren’t.**

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



code4sp coding for social promotion

SQL Select

The SELECT statement **selects or retrieves data from one or more tables.**

You can use this statement to retrieve all the rows from a table in one go or retrieve only those rows that satisfy a specific condition or a combination of conditions.

- *Select All from Table:* returns all the rows from the Employees' table

```
SELECT * FROM employees;
```

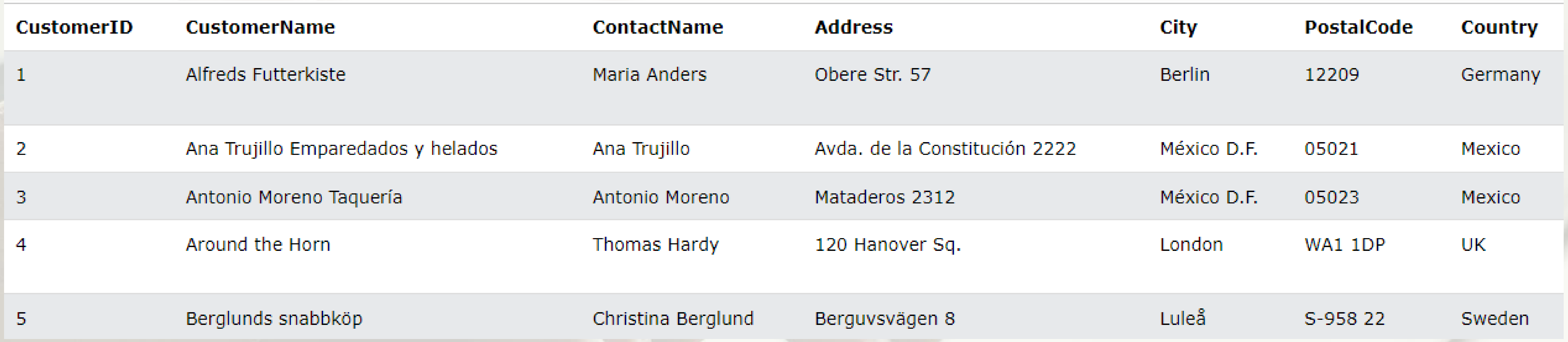
- *Select Specific Columns from Table:* returns all rows from specified columns

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

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

SQL Select Distinct

In the previous slide, we have seen how to select all values from a table or from specific columns.



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

Customers table - SELECT DISTINCT example

(Source: https://www.w3schools.com/sql/sql_distinct.asp)

- *Write a statement to select all values from the Country column in the Customers table*

  Can you notice anything? Are the values different or the same?

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

SQL Select Distinct

SELECT DISTINCT **omits duplicated values** when used in a query.

Syntax:

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

Example 1: Select distinct countries from the Customers table

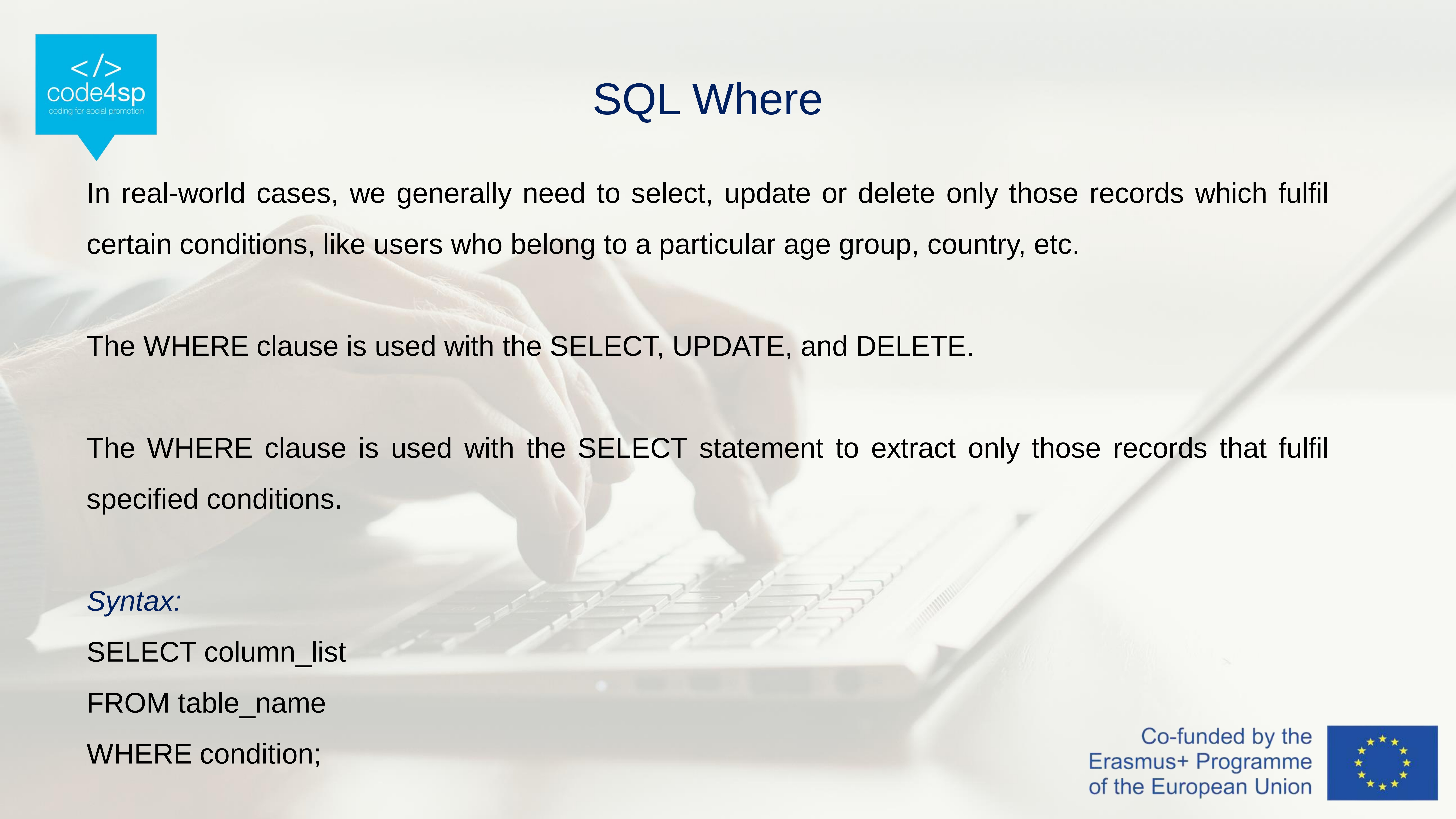
```
SELECT DISTINCT Country FROM Customers;
```

Example 2: List the number of different Customer countries

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

***Note that this example will not work in Firefox since COUNT(DISTINCT column_name) is not supported in MS Access**

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



SQL Where

In real-world cases, we generally need to select, update or delete only those records which fulfil certain conditions, like users who belong to a particular age group, country, etc.

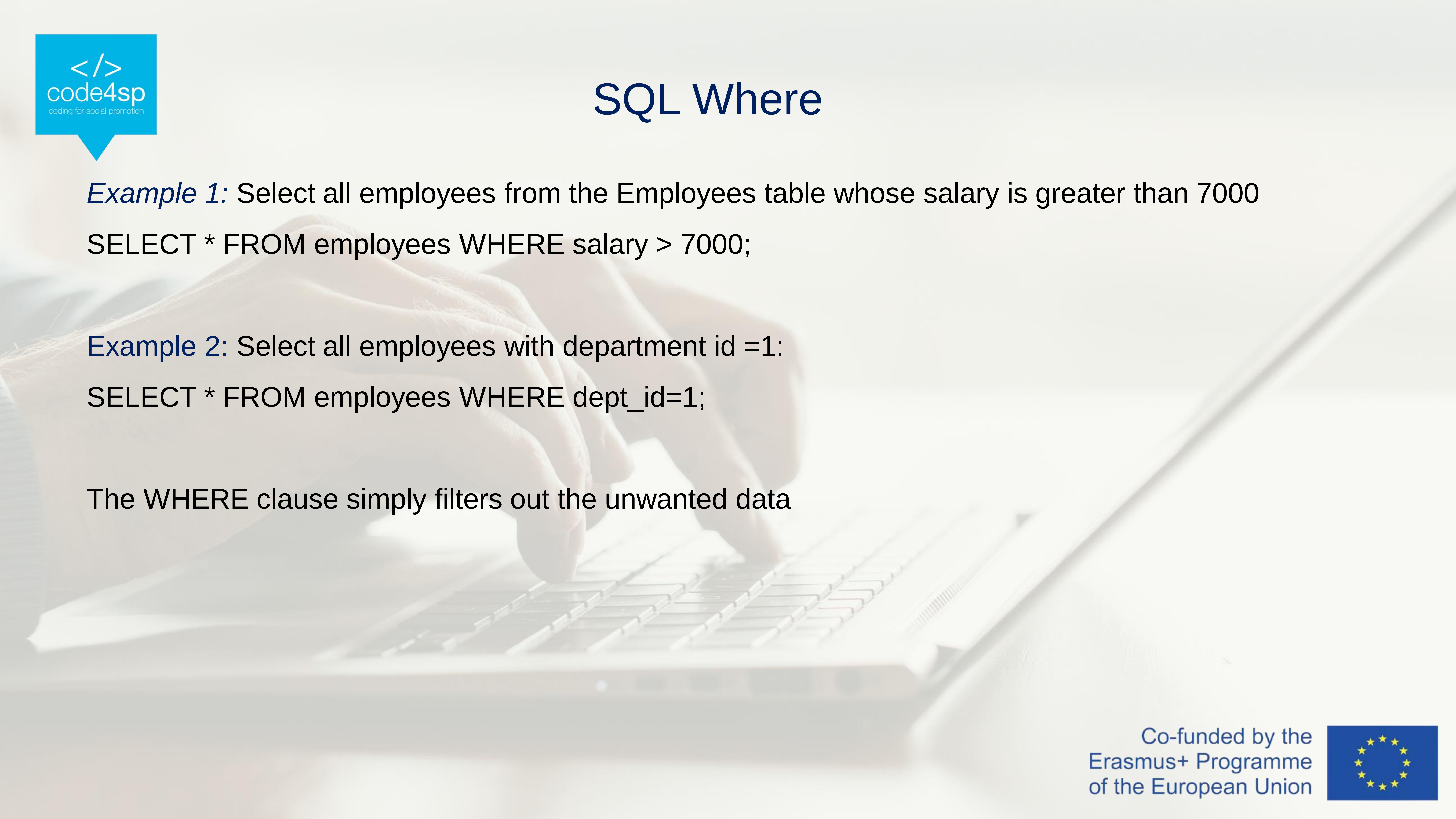
The WHERE clause is used with the SELECT, UPDATE, and DELETE.

The WHERE clause is used with the SELECT statement to extract only those records that fulfil specified conditions.

Syntax:

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

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



SQL Where

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

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

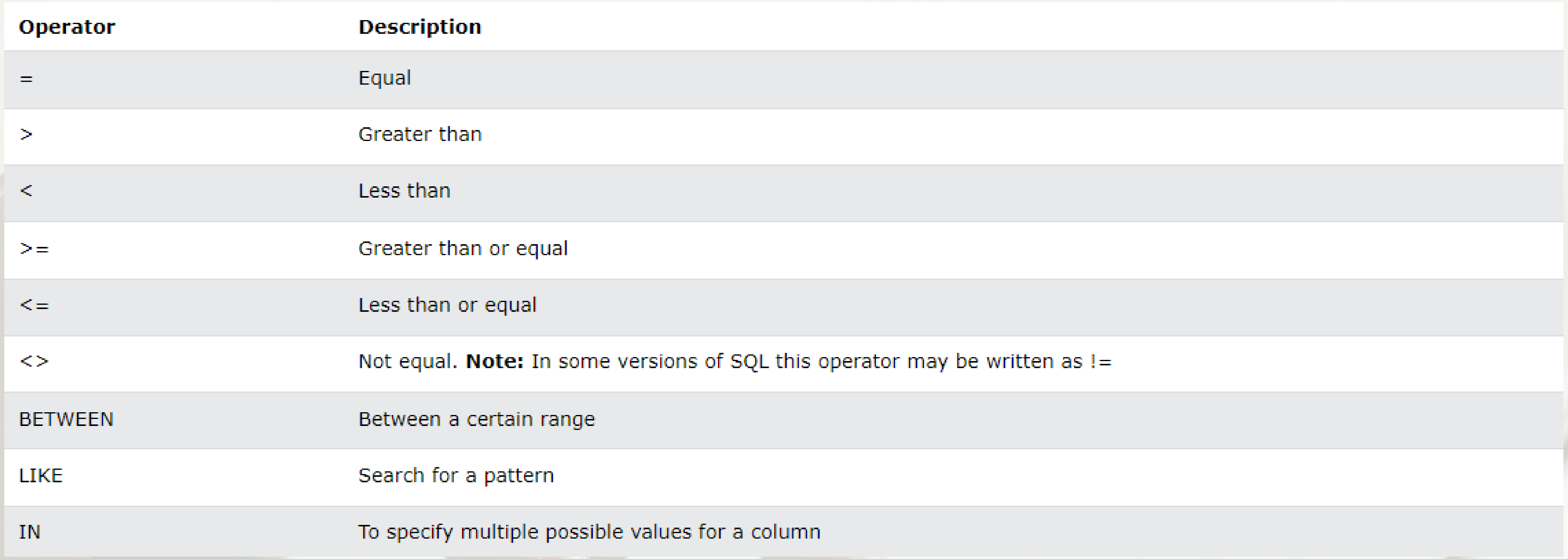
Example 2: Select all employees with department id =1:

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

The WHERE clause simply filters out the unwanted data

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

SQL Where



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

Table of Operators used in WHERE clause

(Source: https://www.w3schools.com/sql/sql_where.asp)

Practice:

- Select all employees from the employees table whose salary is less than 7000
- Select all employees from the employees table whose dept_id equals 5

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

SQL And, Or, Not

The WHERE clause can be combined with AND, OR, and NOT operators.

The **AND operator** displays a record if all the conditions that use AND are TRUE.

Syntax:

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

Example: Select all fields from Customers table where country is “Germany” AND City is “Berlin”

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

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

SQL And, Or, Not

The **OR operator** displays a record if any of the conditions that use OR are TRUE.

Syntax:

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

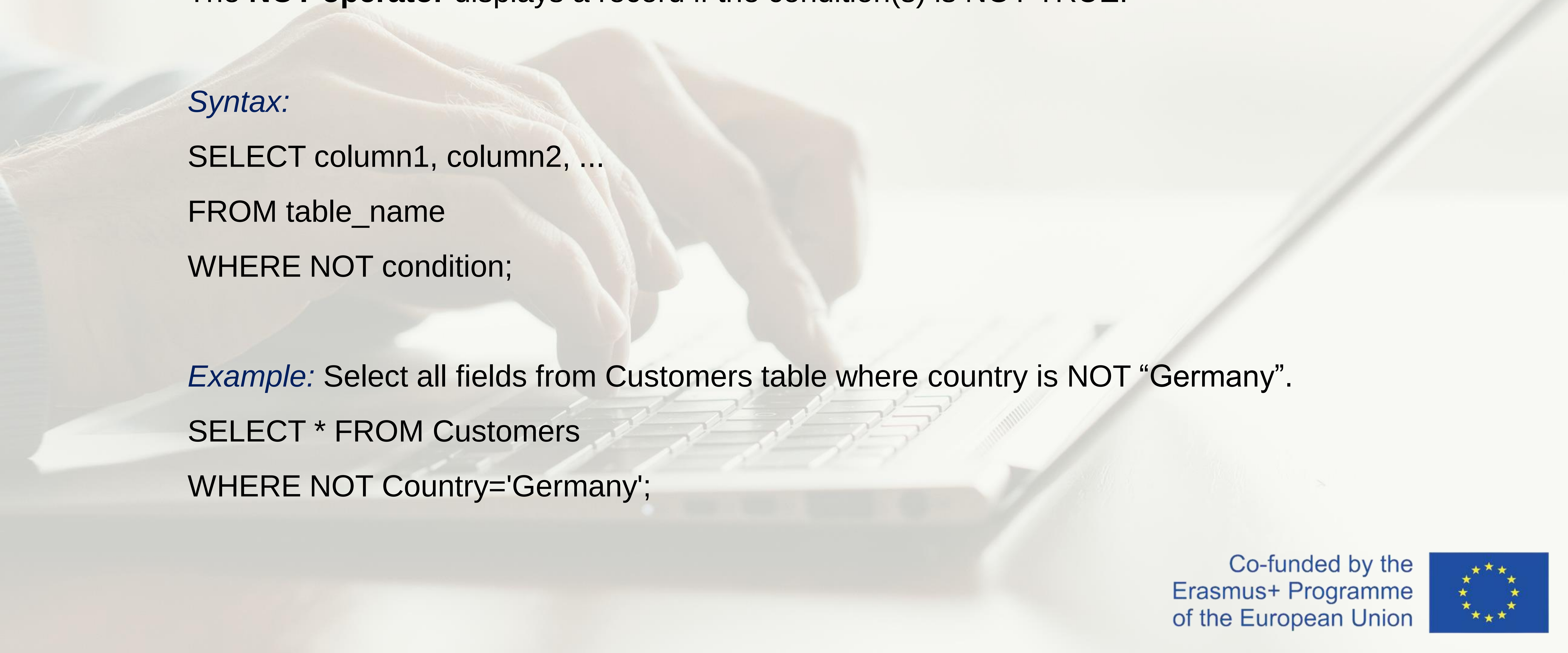
Example 1: Select all fields from Customers table where city is “Berlin” or “München”

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

Example 2: Select all fields from Customers table where country is “Germany” or “Spain”.

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

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



SQL And, Or, Not

The **NOT operator** displays a record if the condition(s) is NOT TRUE.

Syntax:

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

Example: Select all fields from Customers table where country is NOT “Germany”.

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

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

SQL And, Or, Not

Combining AND, OR and NOT

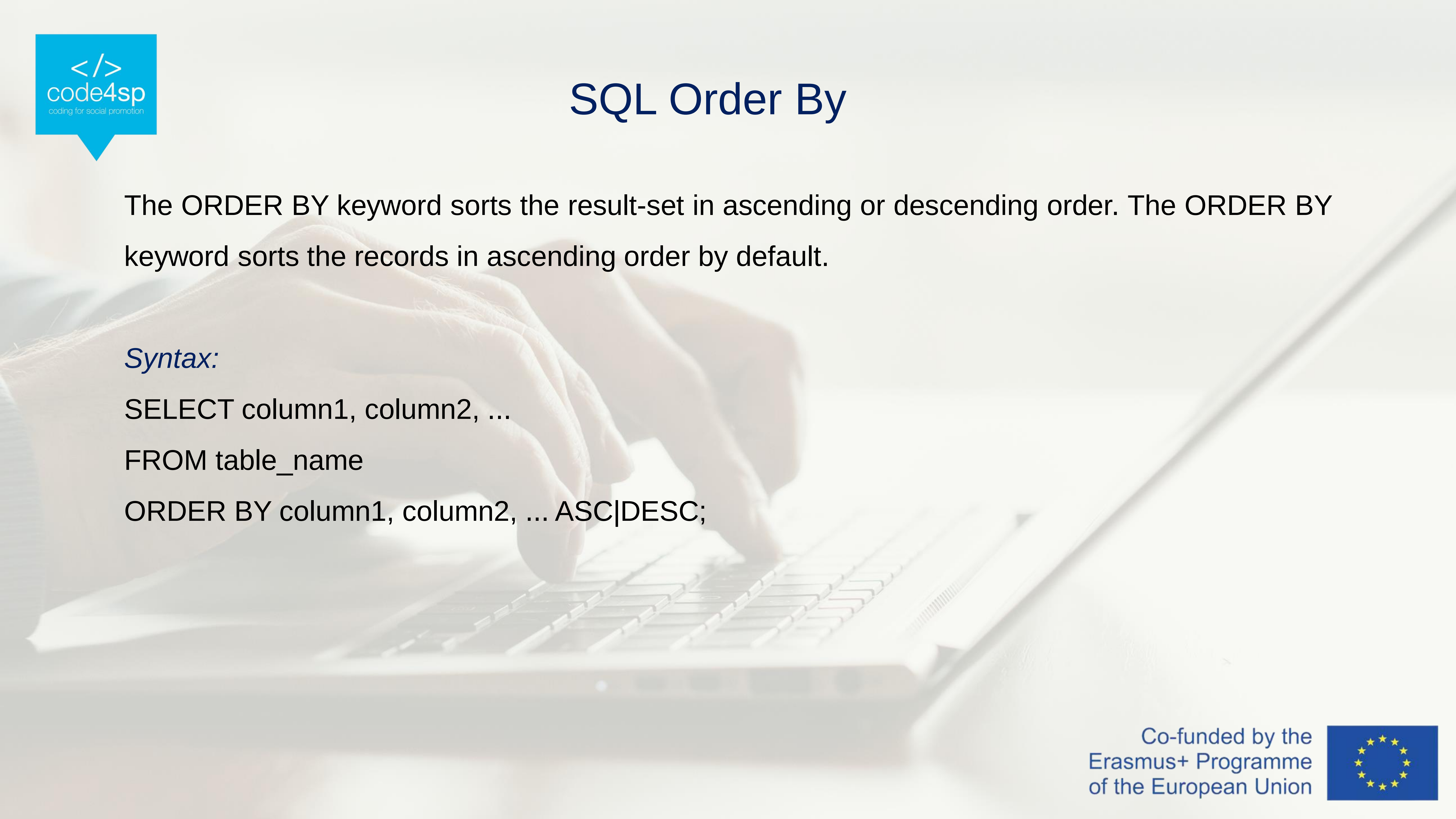
Example 1: Select all rows from the Customers table where country is Germany and city must be either Berlin or München

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

Example 2: Select all rows from the Customers table where country is NOT Germany and NOT USA

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

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



SQL Order By

The ORDER BY keyword sorts the result-set in ascending or descending order. The ORDER BY keyword sorts the records in ascending order by default.

Syntax:

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

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

SQL Order By

For our examples, we will use the Customers table that can be found here

Example 1: Selects all customers from the Customers table and sorts them by the Country column

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

Example 2: Selects all customers from the same table and sorts them in Descending order by the Country column

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

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

SQL Order By

Example 3: Selects all customers from the same table and sorts them by Country and Customer Name.

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

* Here, the order is initially sorted by Country. However, if there are some rows that have the same country, then they are sorted by Customer Name.

Example 4: Selects all customers from the same table and sorts them in ascending order by Country and descending order by Customer Name

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

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

SQL Order By

Example 3: Selects all customers from the same table and sorts them by Country and Customer Name.

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

* Here, the order is initially sorted by Country. However, if there are some rows that have the same country, then they are sorted by Customer Name.

Example 4: Selects all customers from the same table and sorts them in ascending order by Country and descending order by Customer Name

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

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

SQL Order By

Practice:

- Select all customers from the Customers table and sort them by the Country column in ascending order
- Select all customers from the Customers table and sort them in descending order by Customer Name
- Select all customers from the Customers table and sort them in descending order by City and ascending order by Customer Name

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

SQL Insert Into

The INSERT INTO statement inserts new records in a table. For your code to run correctly, specify both the column names and the values that will be inserted.

Syntax:

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

Example: Add a new record in your “Customers” table

```
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

SQL Null Values

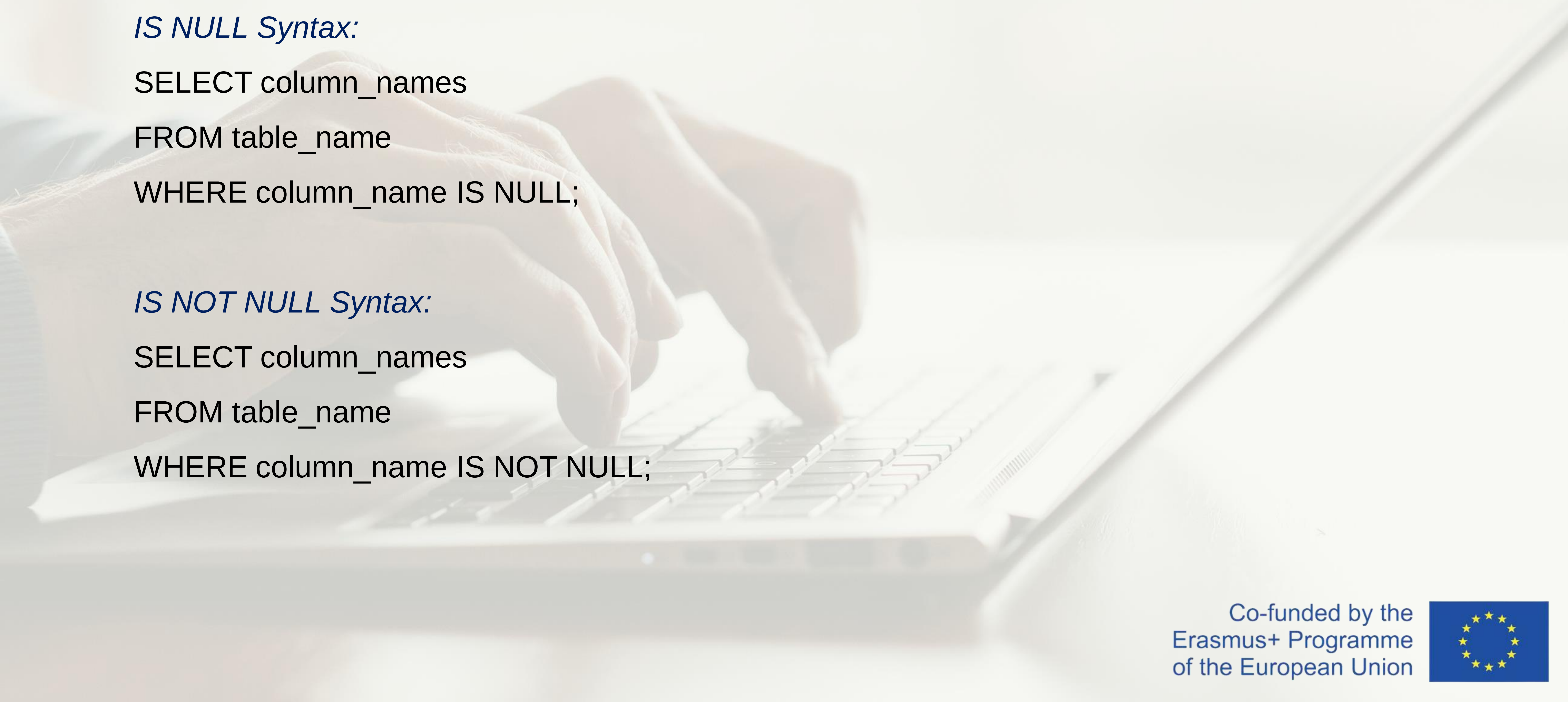
A field with a NULL value is **a field with no value**. If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field.

*** A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value has been left blank during record creation!**

To test for NULL values, you cannot use comparison operators such as =, <>.

You can use either the **IS NULL** or **IS NOT NULL** operators.

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



SQL Null Values

IS NULL Syntax:

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

IS NOT NULL Syntax:

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

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

SQL Null Values

Example of IS NULL: Selects all customers with Null values in the Address column

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

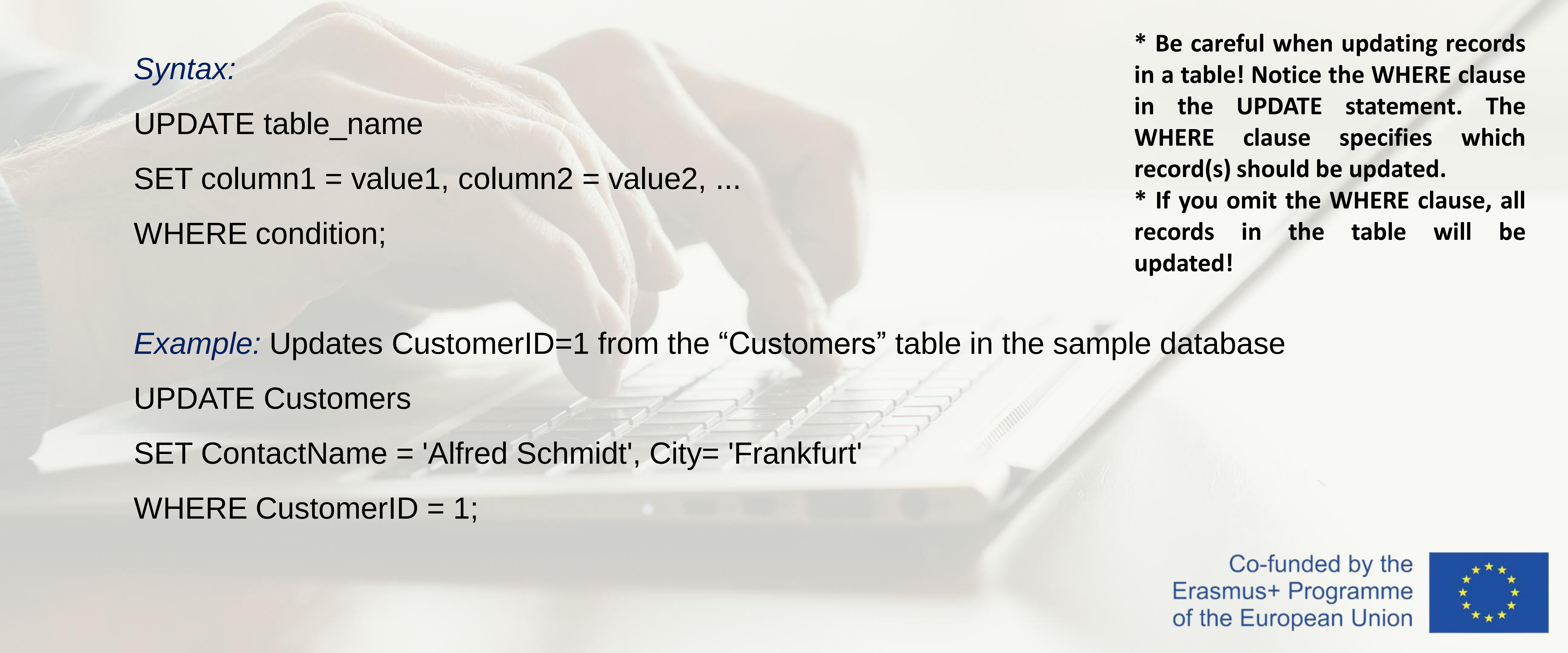
Example of IS NULL: Selects all customers with NOT Null values in the Address column

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

Practice:

- Look for null values in the PostalCode and City columns
- Look for non-empty values in the ContactName column

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



SQL Update

The UPDATE statement **modifies the existing records** of a table.

Syntax:

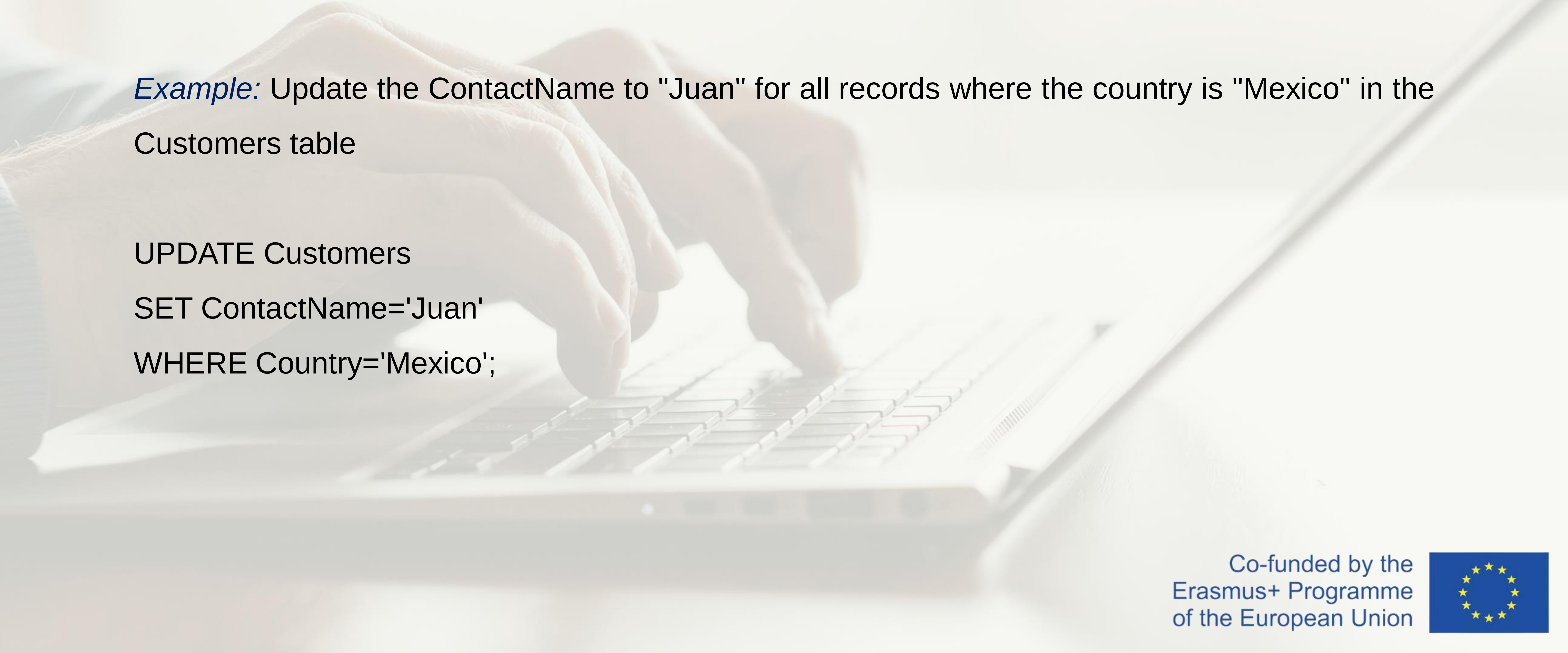
```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

*** Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) should be updated.**
*** If you omit the WHERE clause, all records in the table will be updated!**

Example: Updates CustomerID=1 from the “Customers” table in the sample database

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

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



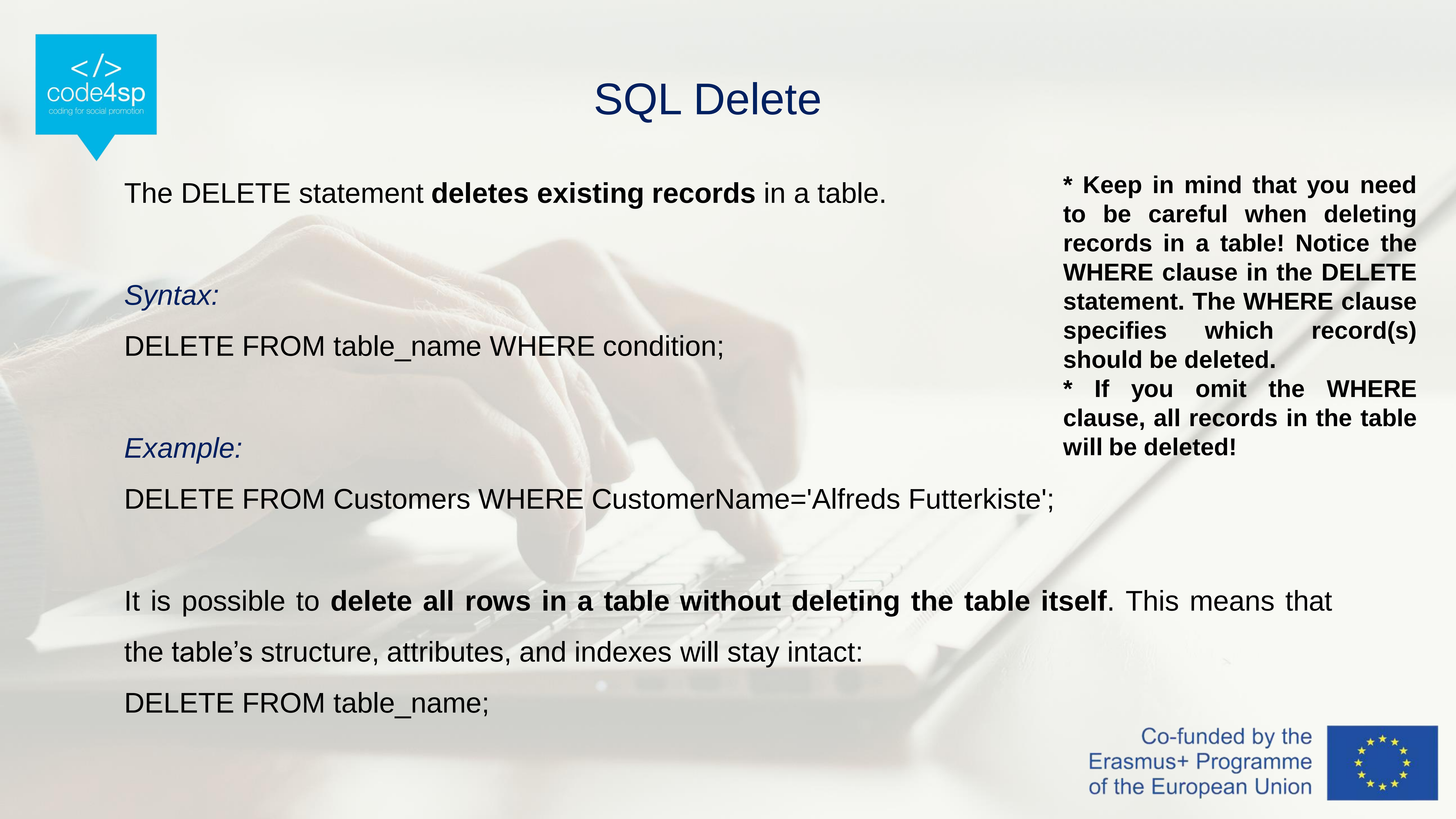
SQL Update

The WHERE clause **determines how many records** will be updated.

Example: Update the ContactName to "Juan" for all records where the country is "Mexico" in the Customers table

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

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



SQL Delete

The DELETE statement **deletes existing records** in a table.

Syntax:

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

Example:

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

*** Keep in mind that you need to be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted.**

*** If you omit the WHERE clause, all records in the table will be deleted!**

It is possible to **delete all rows in a table without deleting the table itself**. This means that the table's structure, attributes, and indexes will stay intact:

```
DELETE FROM table_name;
```

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

SQL Delete

The DELETE statement **deletes existing records** in a table.

Syntax:

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

Example:

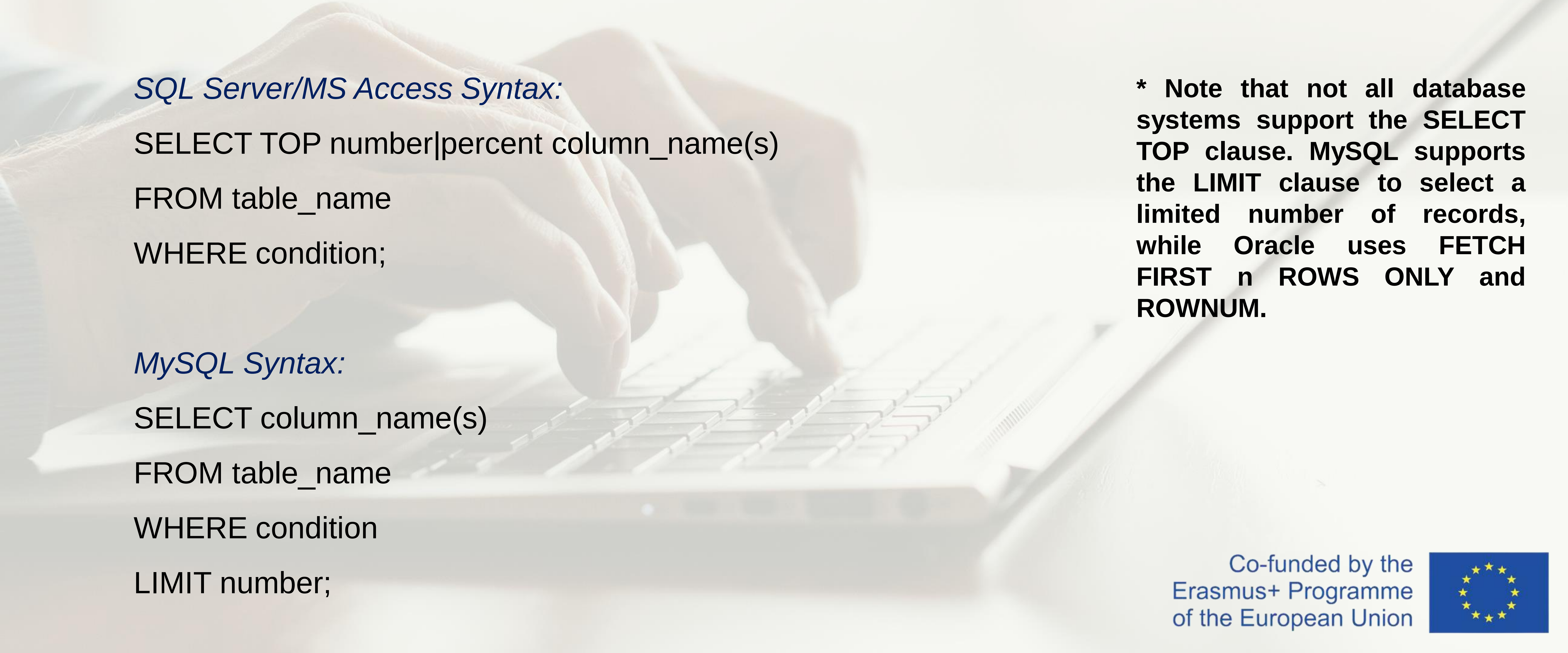
```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

It is possible to **delete all rows in a table without deleting the table itself**. This means that the table’s structure, attributes, and indexes will stay intact:

```
DELETE FROM table_name;
```

*** Keep in mind that you need to be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted.**
*** If you omit the WHERE clause, all records in the table will be deleted!**

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



SQL Select Top

The SELECT TOP clause **specifies the number of records that will be returned.**

SQL Server/MS Access Syntax:

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

MySQL Syntax:

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

*** Note that not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses FETCH FIRST n ROWS ONLY and ROWNUM.**

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

SQL Select Top

Oracle 12 Syntax:

SELECT column_name(s)

FROM table_name

ORDER BY column_name(s)

FETCH FIRST number ROWS ONLY;

For Older Oracle Syntax and Older Oracle Syntax with ORDER BY, click [here]

Example: Select the first three records from the Customers table

SQL Server/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

SQL Select Top

TOP PERCENT

To select the first 50% of records found in the Customers table:

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

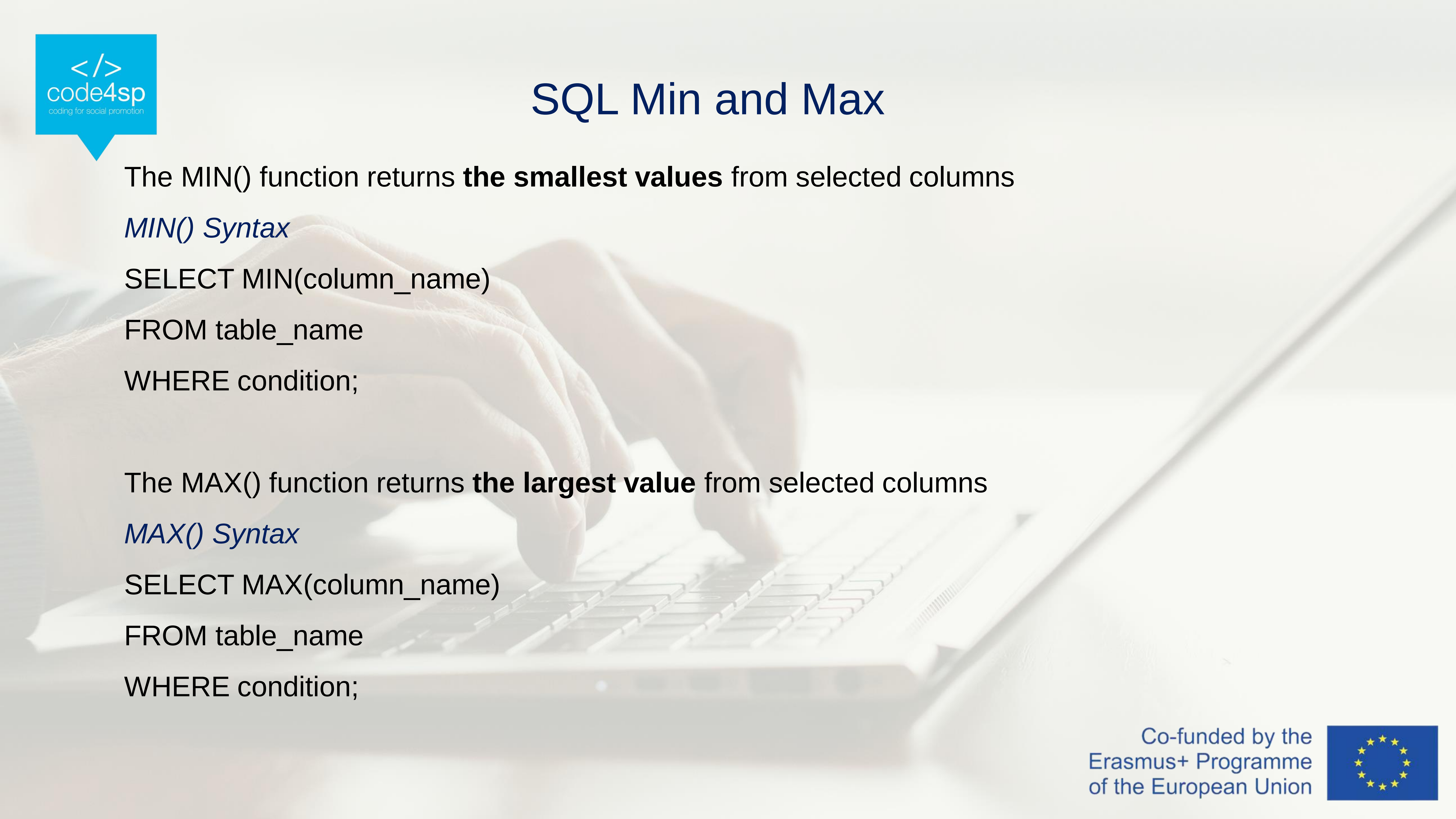
Oracle: SELECT * FROM Customers FETCH FIRST 50 PERCENT ROWS ONLY;

ADD a WHERE Clause

Select the first three records from the Customers table, where the Country is “Germany”:

- 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



SQL Min and Max

The MIN() function returns **the smallest values** from selected columns

MIN() Syntax

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

The MAX() function returns **the largest value** from selected columns

MAX() Syntax

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

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

SQL Min and Max

In the following examples, we are using the Products table that can be found here.

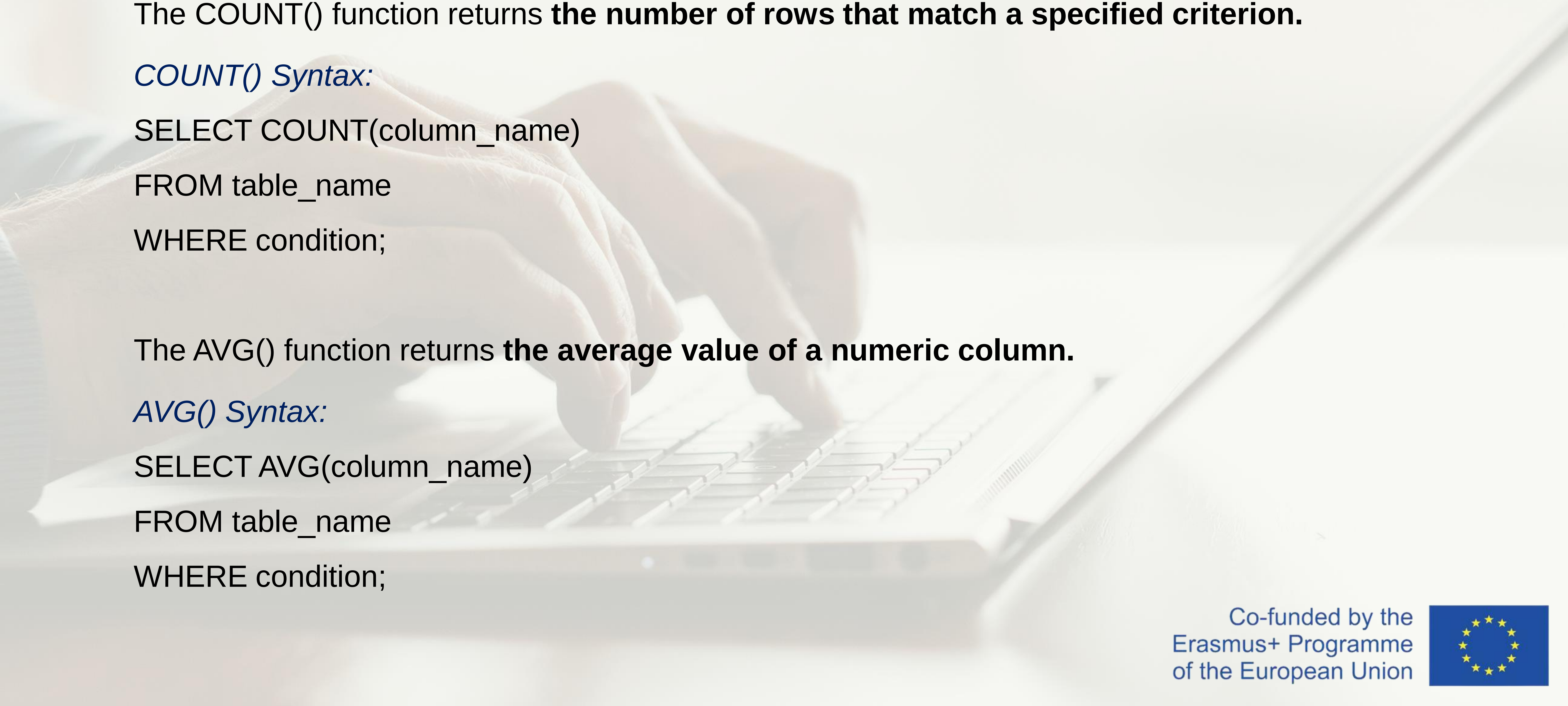
Example 1: Finds the price of the cheapest product

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

Example 2: Finds the price of the most expensive product

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

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



SQL Count, Avg, Sum

The COUNT() function returns **the number of rows that match a specified criterion.**

COUNT() Syntax:

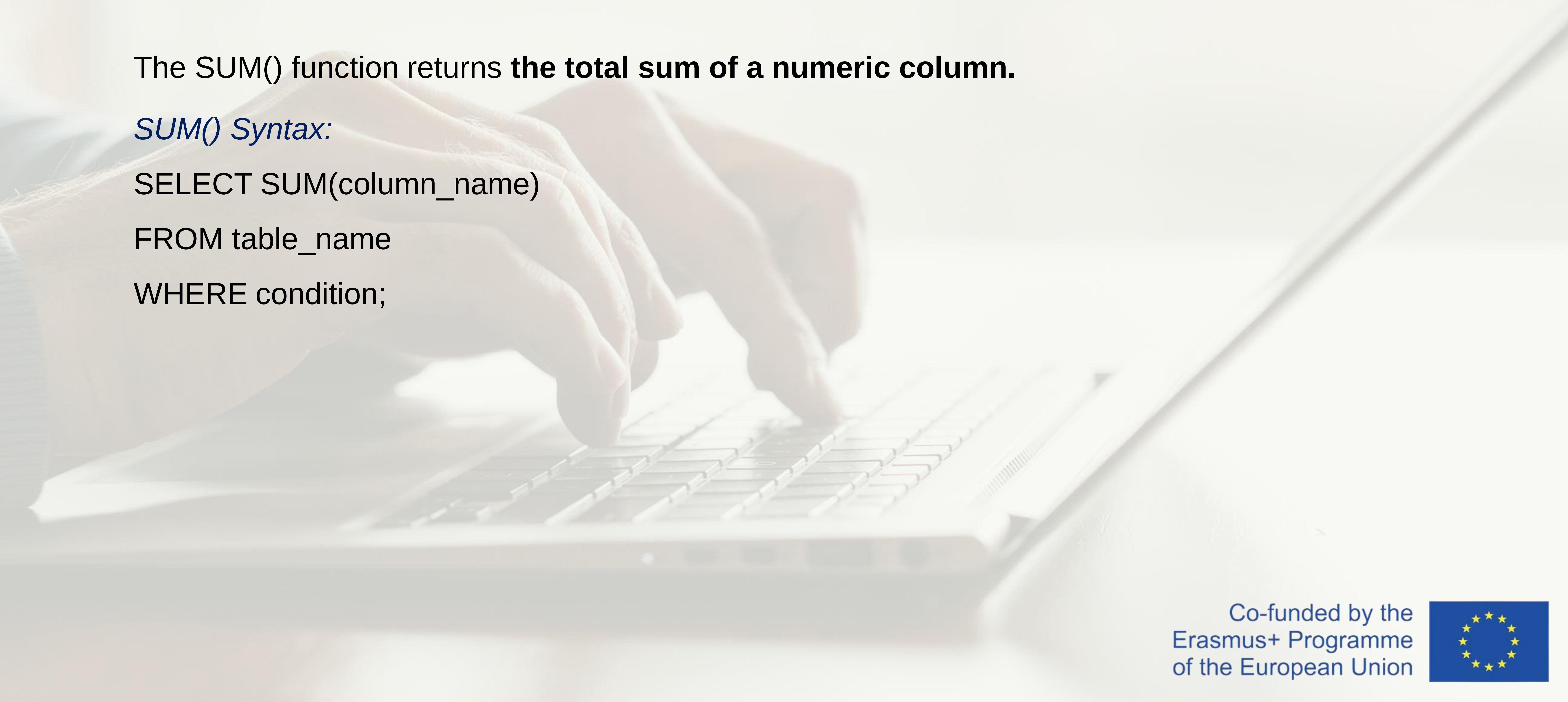
```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

The AVG() function returns **the average value of a numeric column.**

AVG() Syntax:

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

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



SQL Count, Avg, Sum

The SUM() function returns **the total sum of a numeric column.**

SUM() Syntax:

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

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

SQL Count, Avg, Sum

For the examples, we are using the same table in the SQL Min and Max, the Products table:

Example of COUNT(): Execute a query to find the number of products

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

Example of AVG(): Execute a query to find the average price of all products

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

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

code4sp
coding for social promotion

SQL Count, Avg, Sum

In the following example, we will use the OrdersDetails table, which can be found here, to find the sum of “Quantity”.

Example of SUM():

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

Practice:

- Find the average Quantity from the OrdersDetails table
- Find the number of Orders from the OrdersDetails table

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

SQL Like

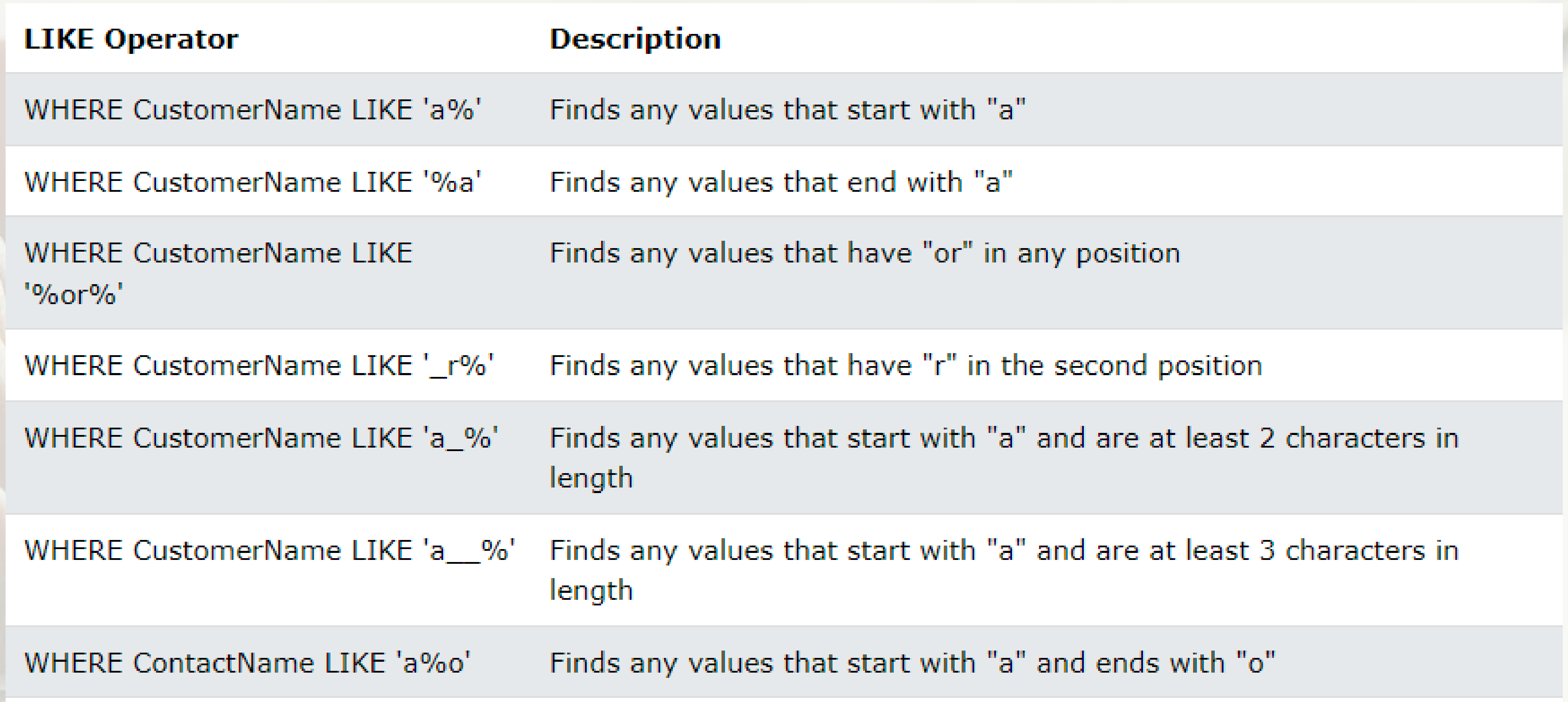
The LIKE operator is used in a WHERE clause to **search for a specified pattern in a column.**

LIKE Syntax:

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"

LIKE operators with '%' and '_' Wildcards

(Source: https://www.w3schools.com/sql/sql_like.asp)

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

SQL Like

Example 1: Selects all customers with a Customer Name that starts with ‘a’:

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

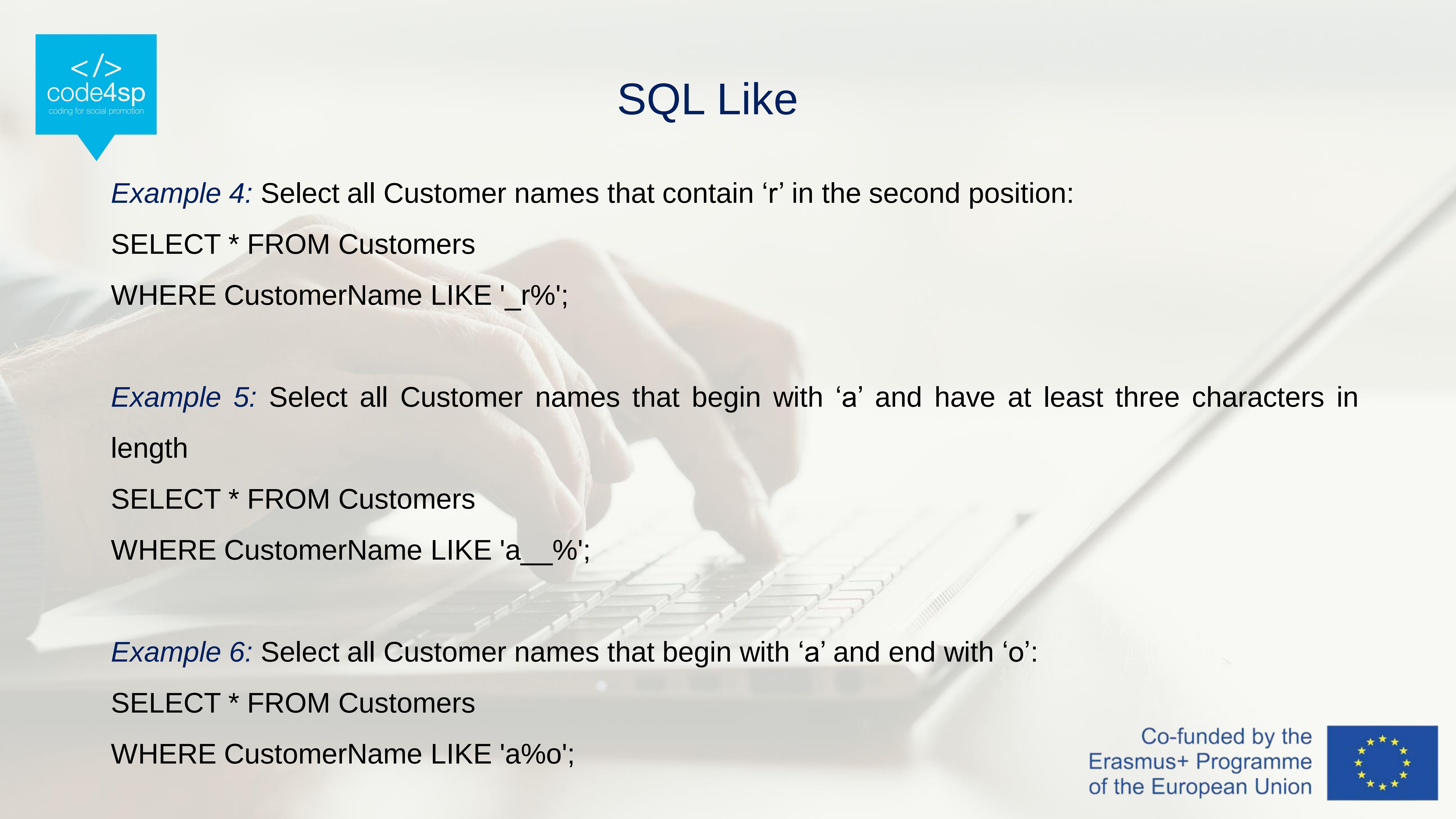
Example 2: Selects all Customer names ending with ‘a’:

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

Example 3: Selects all Customer names that contain ‘or’ in their name in any position:

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

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



SQL Like

Example 4: Select all Customer names that contain ‘r’ in the second position:

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

Example 5: Select all Customer names that begin with ‘a’ and have at least three characters in length

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

Example 6: Select all Customer names that begin with ‘a’ and end with ‘o’:

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

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

SQL Wildcards

A wildcard character **substitutes one or more characters in a stri**ng. It is used with the LIKE operator.

For more on Wildcards used in RDBMS, click here.

Examples with the % Wildcard

In this example, we are selecting all customers with a City starting with “ber”:

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

In the following example, we are selecting all customers with a City containing “es”:

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

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

SQL Wildcards

Examples with the _ Wildcard

Here, we are selecting all customers with a City starting with any character followed by “ondon”:

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

In this example, we are again selecting all customers with a City starting with ‘L’, followed by any character, followed by "n", followed by any character, followed by "on":

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

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

SQL Wildcards

Examples with the [charlist] Wildcard

Here, we are selecting all customers with a City starting with ‘b’,’ s’, or ‘p’:

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

In the following example, we are selecting all customers with City starting with ‘a’, ‘b’, or ‘c’:

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

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

SQL Wildcards

Examples with the [!charlist] Wildcard

The exclamation mark shows characters that do not contain a specified string. For example, we want to select all customers with a City that does not start with ‘b’, ‘s’, or ‘p’:

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

As an alternative, use the following:

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

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

SQL In

The IN operator **specifies multiple values in a WHERE clause**. It can be considered as satisfying various conditions.

Syntax 1:

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

Syntax 2:

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

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

SQL In

Suppose that we have a table called "Customers" containing the following columns: CustomerID, CustomerName, ContactName, Address, City, PostalCode and Country (see the table here).

Example 1: Selects all the customers that are located in Germany, France or UK

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

Example 2: Selects all the customers that are not located in Germany, France or the UK

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

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

SQL In

Example 3: Select customers that are from the same countries as the suppliers

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

Practice:

- Select customers located in the City Berlin
- Select customers located in the City London and Madrid
- Select customers that are not from the same country as the suppliers

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

SQL Between

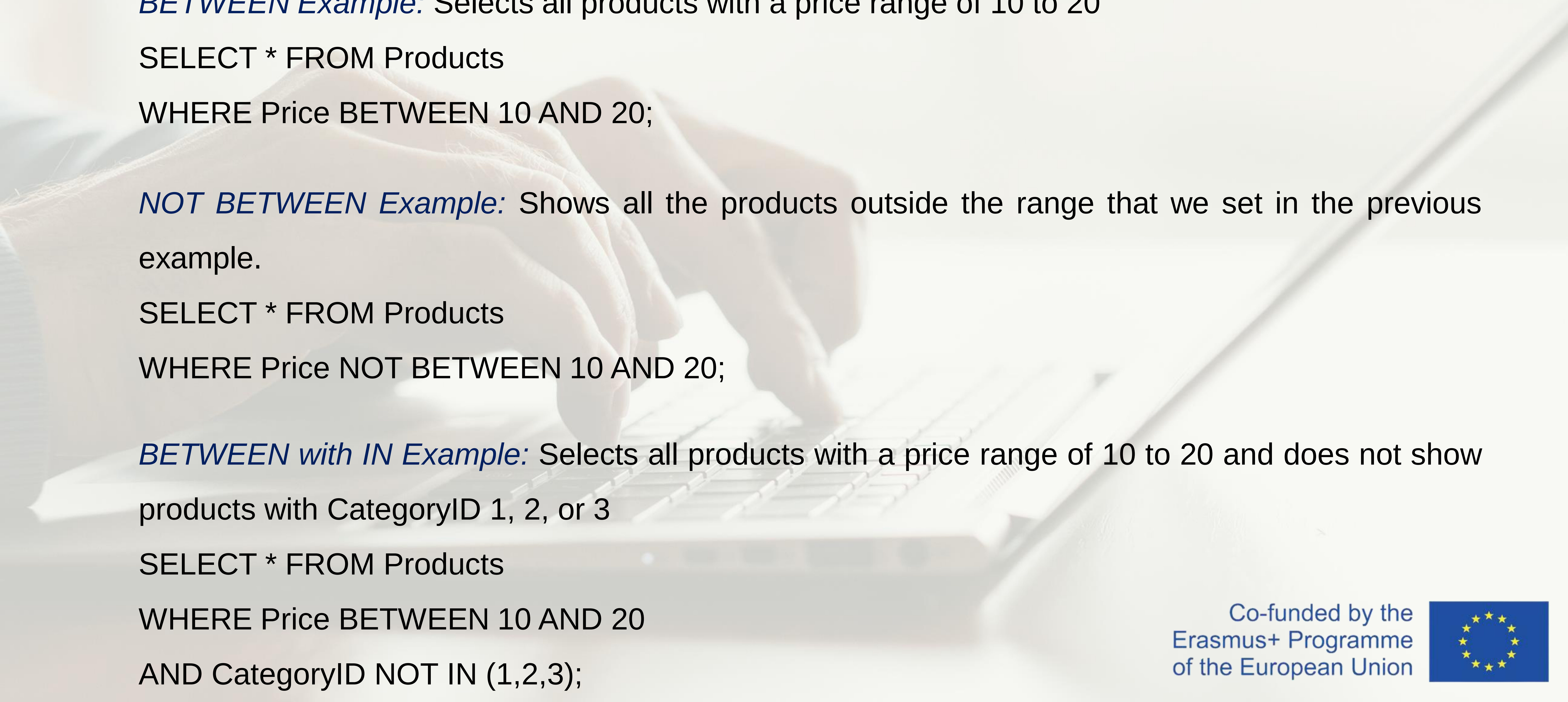
The BETWEEN operator provides a range of values to select from. The values can be text, numbers or dates. The BETWEEN operator **includes the starting and end values.**

Syntax:

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

There will be a series of examples with the following operators: BETWEEN, NOT BETWEEN, BETWEEN with IN, BETWEEN and NOT BETWEEN with text values and BETWEEN dates.

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



SQL Between

BETWEEN Example: Selects all products with a price range of 10 to 20

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

NOT BETWEEN Example: Shows all the products outside the range that we set in the previous example.

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

BETWEEN with IN Example: Selects all products with a price range of 10 to 20 and does not show products with CategoryID 1, 2, or 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

BETWEEN with text values Example: Selects all products with a ProductName between Carnarvon Tigers and Mozzarella di Giovanni.

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

NOT BETWEEN with text values Example: Selects all products with a ProductName NOT between Carnarvon Tigers and 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

BETWEEN Dates Example: Selects all orders with an OrderDate between '01-July-1996' and '31-July-1996' (to find the table used in this example, go here)

There are two ways that this can be done, by either using a hashtag (#) or quote marks (''):

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

OR

```
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

Aliases assign a **temporary name to a table or a column within a table.** An alias exists only for the duration of a query, and it is often used to make column names more readable. An alias is created by using the keyword AS.

Syntax for column alias:

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

Syntax for table alias:

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

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

SQL Aliases

Column Aliases

Let’s see an example that creates two aliases, one for each column:

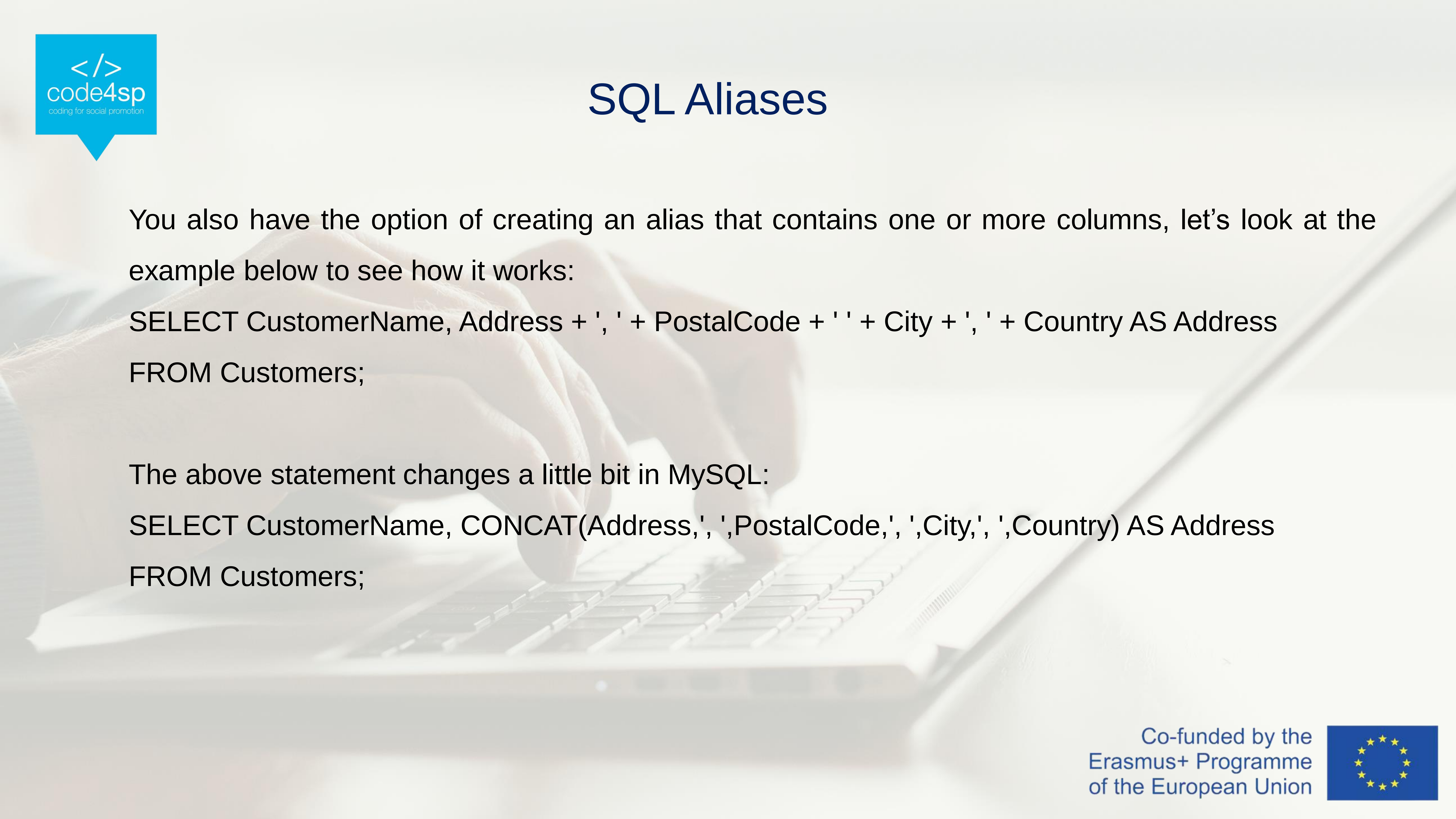
```
SELECT CustomerID AS ID, CustomerName AS Customer
FROM Customers;
```

Another example creates two aliases again:

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

*** Note that it is put in square brackets ([ ]) because the alias contains spaces. Quotation marks can be used as an alternative to square brackets.**

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



SQL Aliases

You also have the option of creating an alias that contains one or more columns, let’s look at the example below to see how it works:

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

The above statement changes a little bit in MySQL:

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

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

SQL Aliases

Table Aliases: The following example selects all the orders from the customer table with CustomerID=4 (Around the Horn).

Here aliases are used to shorten the query:

```
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;
```

A query without aliases would look something like this:

```
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

SQL Join

A JOIN clause **combines rows from two or more tables based on a related column found in both tables.** There are four different joins in SQL:

1. *(INNER) JOIN:* Returns records that have matching values in both tables;
2. *LEFT (OUTER) JOIN:* Returns all records from the left table and the corresponding matched records from the right table;
3. *RIGHT (OUTER) JOIN:* Returns all records from the right table and the corresponding matched records from the left table;
4. *FULL (OUTER) JOIN:* Returns all records when there is a match in either the left or the right table.

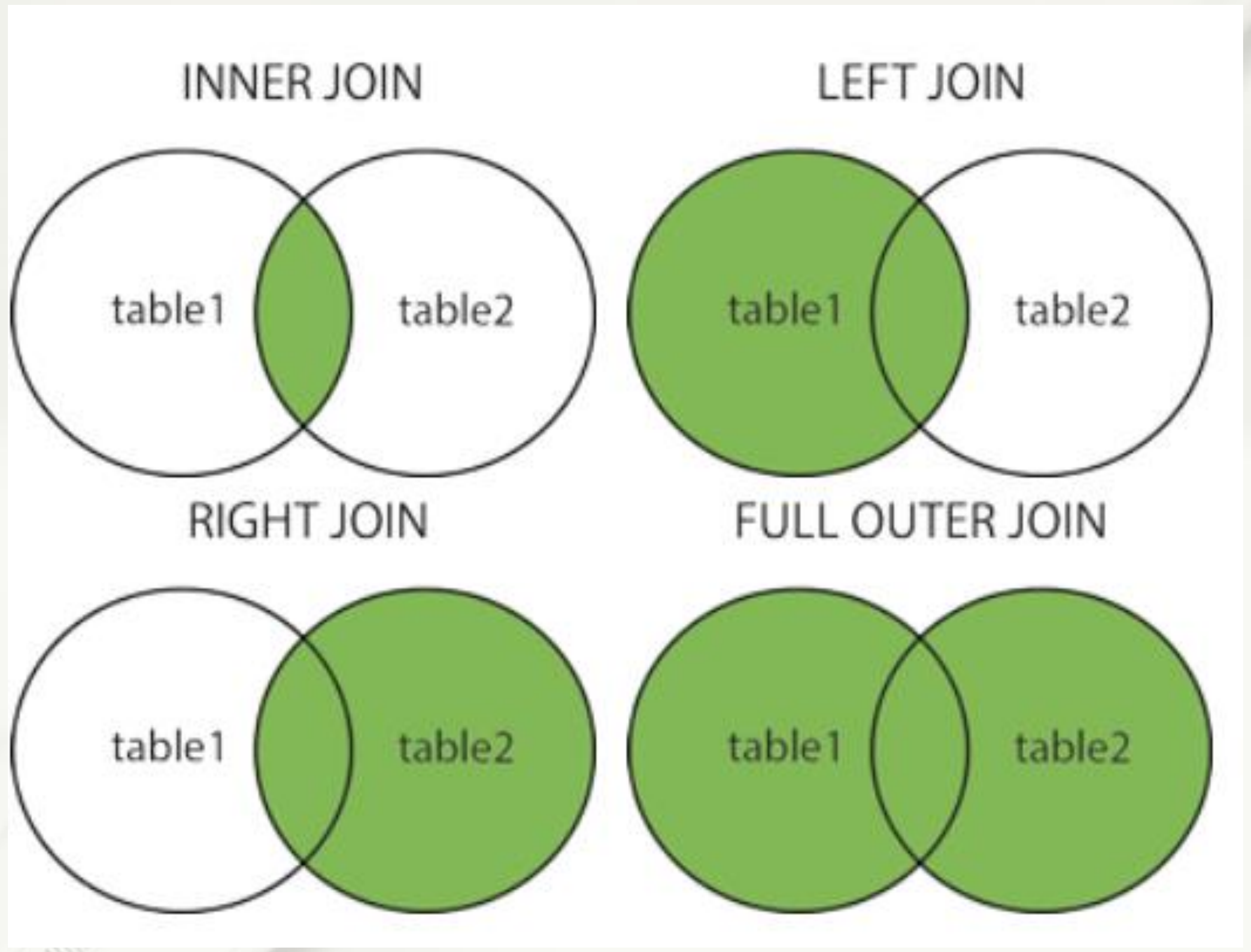


Figure - Different types of JOINS
(Source: https://www.w3schools.com/sql/sql_join.asp)

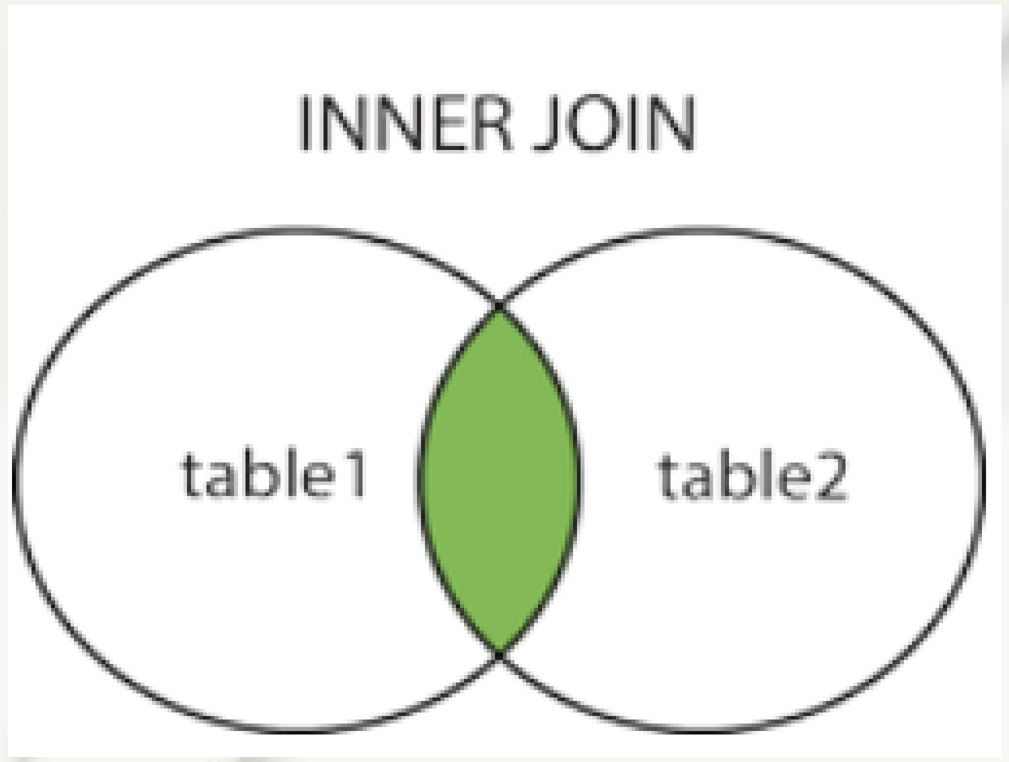
Co-funded by the Erasmus+ Programme of the European Union

SQL Inner Join

The INNER JOIN keyword selects records that have matching values in both tables.

Syntax:

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



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

To retrieve the names of customers and their corresponding Order IDs:

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

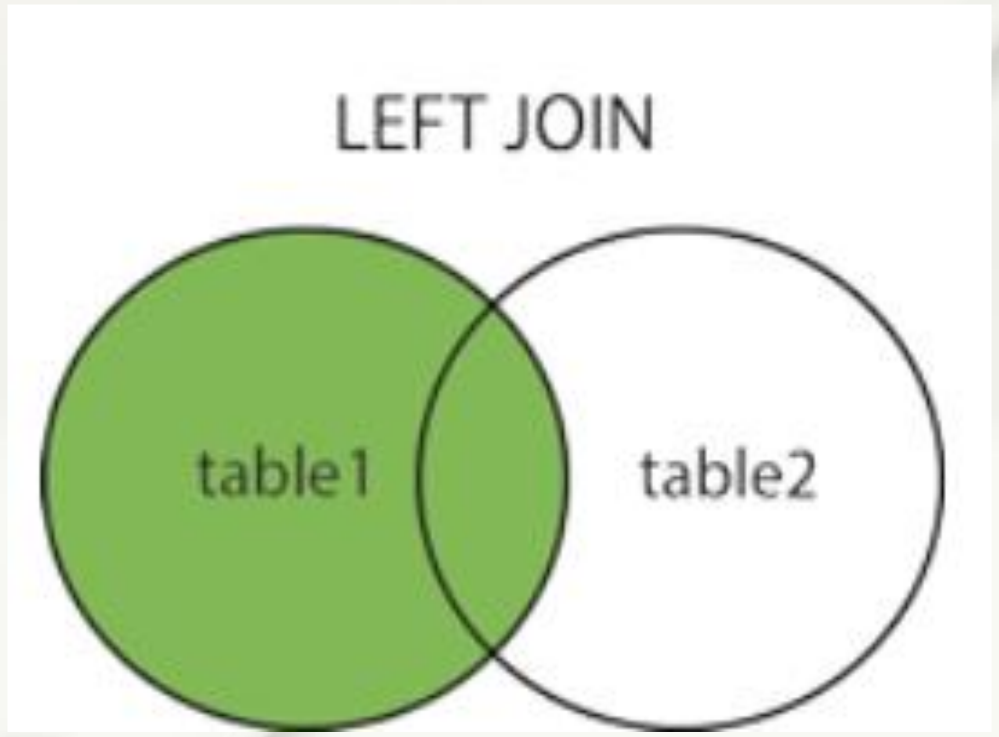
Co-funded by the Erasmus+ Programme of the European Union

SQL Left Join

The LEFT JOIN keyword returns all records from the left table and the matching records from the right table.

Syntax:

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



Left Join
(Source: https://www.w3schools.com/sql/sql_join.asp)

Select all customers and any orders that they might have:

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

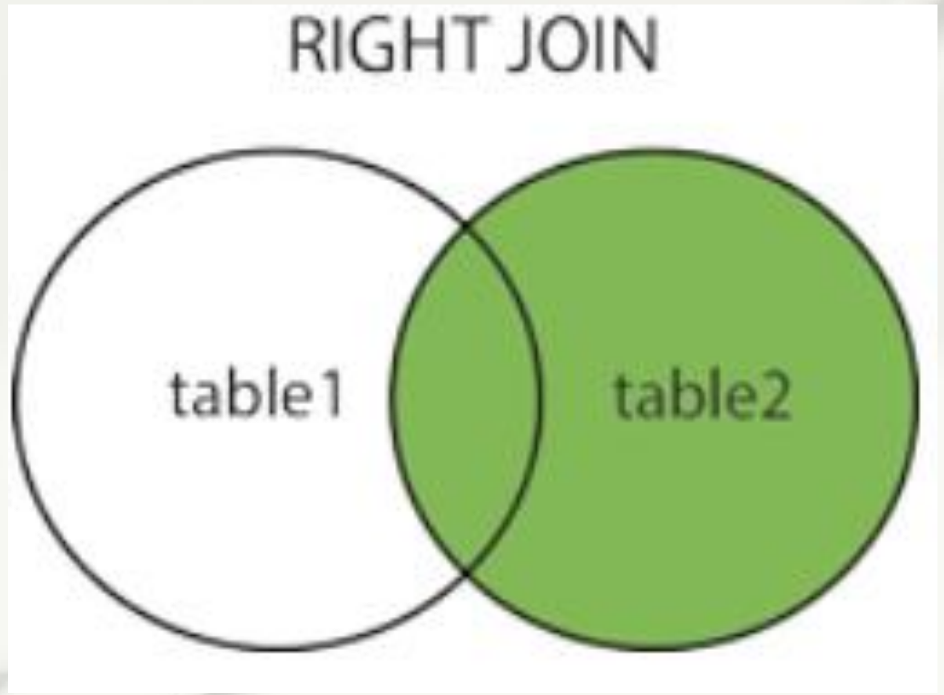
Co-funded by the Erasmus+ Programme of the European Union

SQL Right Join

The RIGHT JOIN keyword essentially follows the same logic from the right side instead of the left one as described in the previous subsection.

Syntax:

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



Right Join
(Source: https://www.w3schools.com/sql/sql_join.asp)

Selects all employees and any orders that they might have placed:

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

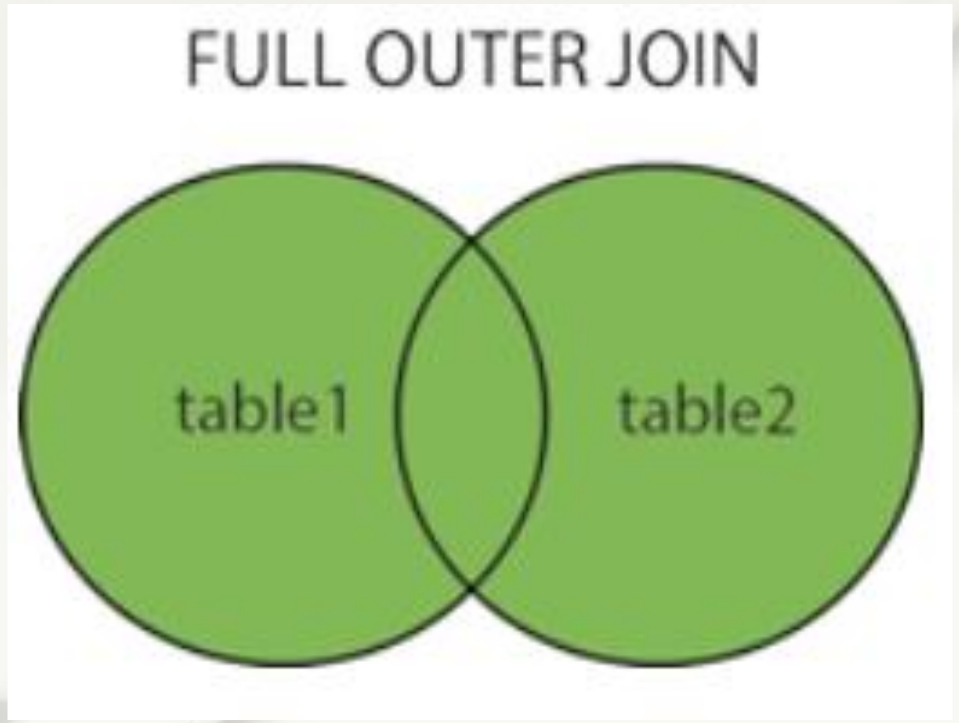
Co-funded by the Erasmus+ Programme of the European Union

SQL Full Join

The FULL JOIN keyword returns all records when matching records are found in either the right or the left table.

Syntax:

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



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

* Note that it returns all matching records from both tables even if no common matches are found between the two tables. In the case of no common matches, a null value is assigned.

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

SQL Self Join

A self-join is considered a regular join, but the table is joined within.

Syntax:

SELECT column_name(s)

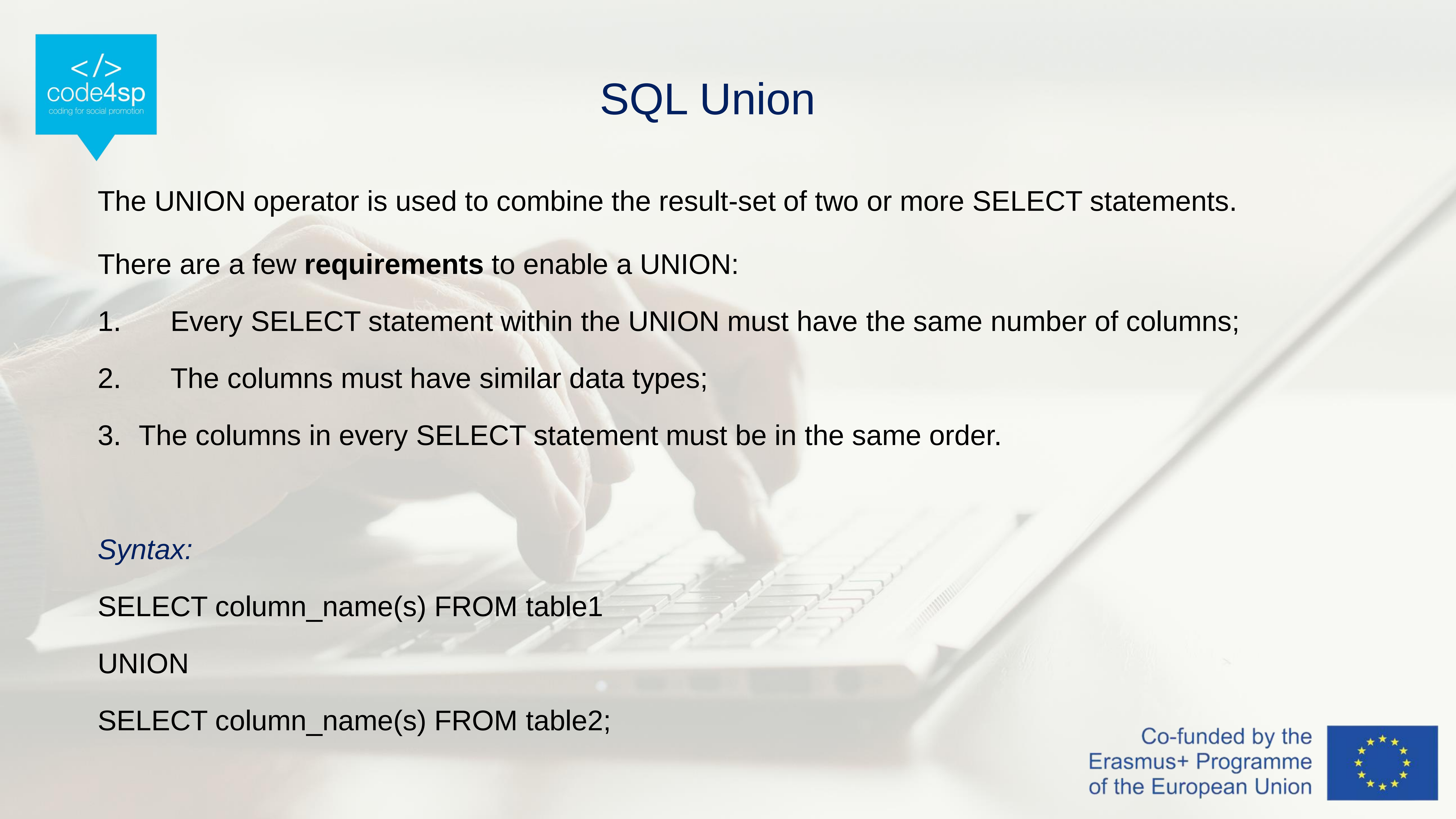
FROM table1 T1, table1 T2   *(T1 and T2 are aliases used for the same table)*

WHERE condition;

Example: Selects customers that are from the same city

```
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



SQL Union

The UNION operator is used to combine the result-set of two or more SELECT statements.

There are a few **requirements** to enable a UNION:

1. Every SELECT statement within the UNION must have the same number of columns;
2. The columns must have similar data types;
3. The columns in every SELECT statement must be in the same order.

Syntax:

```
SELECT column_name(s) FROM table1

UNION

SELECT column_name(s) FROM table2;
```

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

SQL Union

The UNION operator **selects only distinct values by default.**

To allow duplicate values, use UNION ALL:

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

* Note that the column names in the two SELECT statements are usually equal.

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

SQL Union

Example 1: Returns distinct cities

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

Example 2: Return duplicate values

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

Example 3: Returns the distinct German cities from both the “Customers” and “Suppliers” tables with the use of WHERE:

```
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

SQL Group By

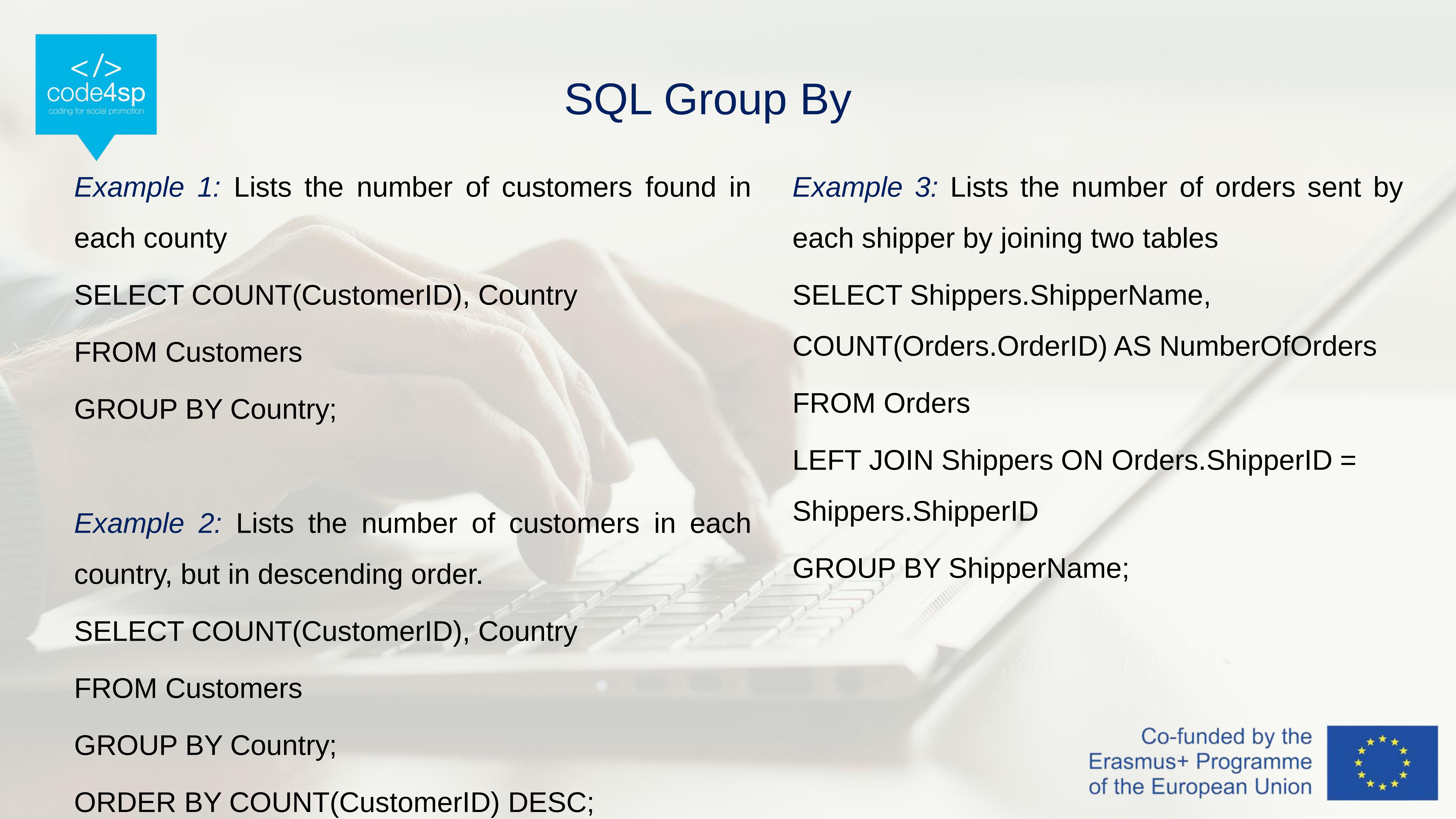
The GROUP BY statement groups rows with the same values into summary rows. As an example, consider that you want to find the number of customers in each country.

Also, the GROUP BY statement is often used with aggregate functions such as COUNT(), MAX(), MIN(), SUM(), AVG() to group the result by one or more columns.

Syntax:

```
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



SQL Group By

Example 1: Lists the number of customers found in each county

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

Example 2: Lists the number of customers in each country, but in descending order.

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

Example 3: Lists the number of orders sent by each shipper by joining two tables

```
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

SQL Having

The HAVING clause was added to SQL because WHERE cannot be used with aggregate functions.

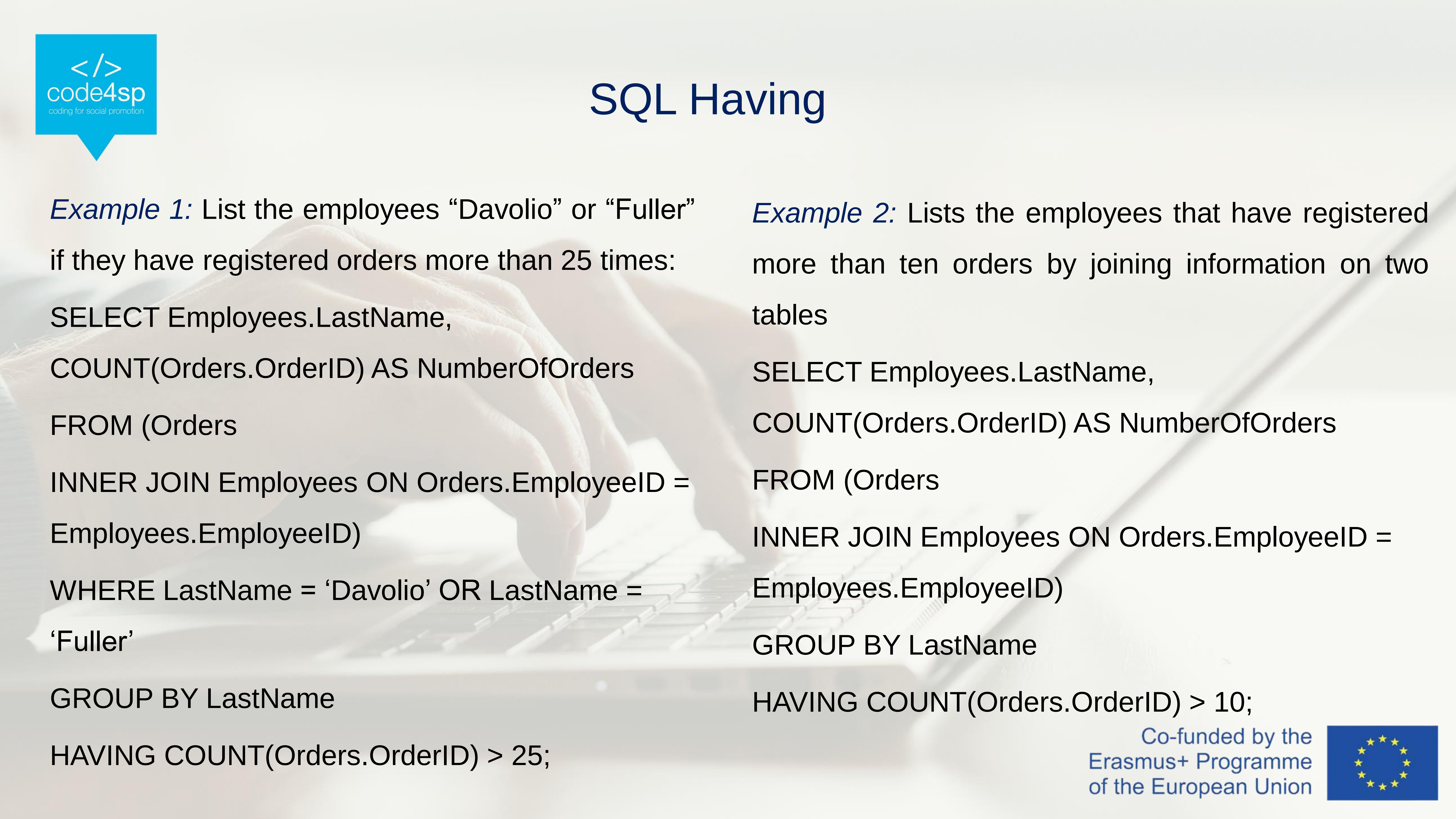
Syntax:

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

Example: Lists the number of customers found in each country and countries that have more than five customers

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

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



SQL Having

Example 1: List the employees “Davolio” or “Fuller” if they have registered orders more than 25 times:

```
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;
```

Example 2: Lists the employees that have registered more than ten orders by joining information on two tables

```
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

SQL Select Into

The SELECT INTO statement copies data from one table into a new table.

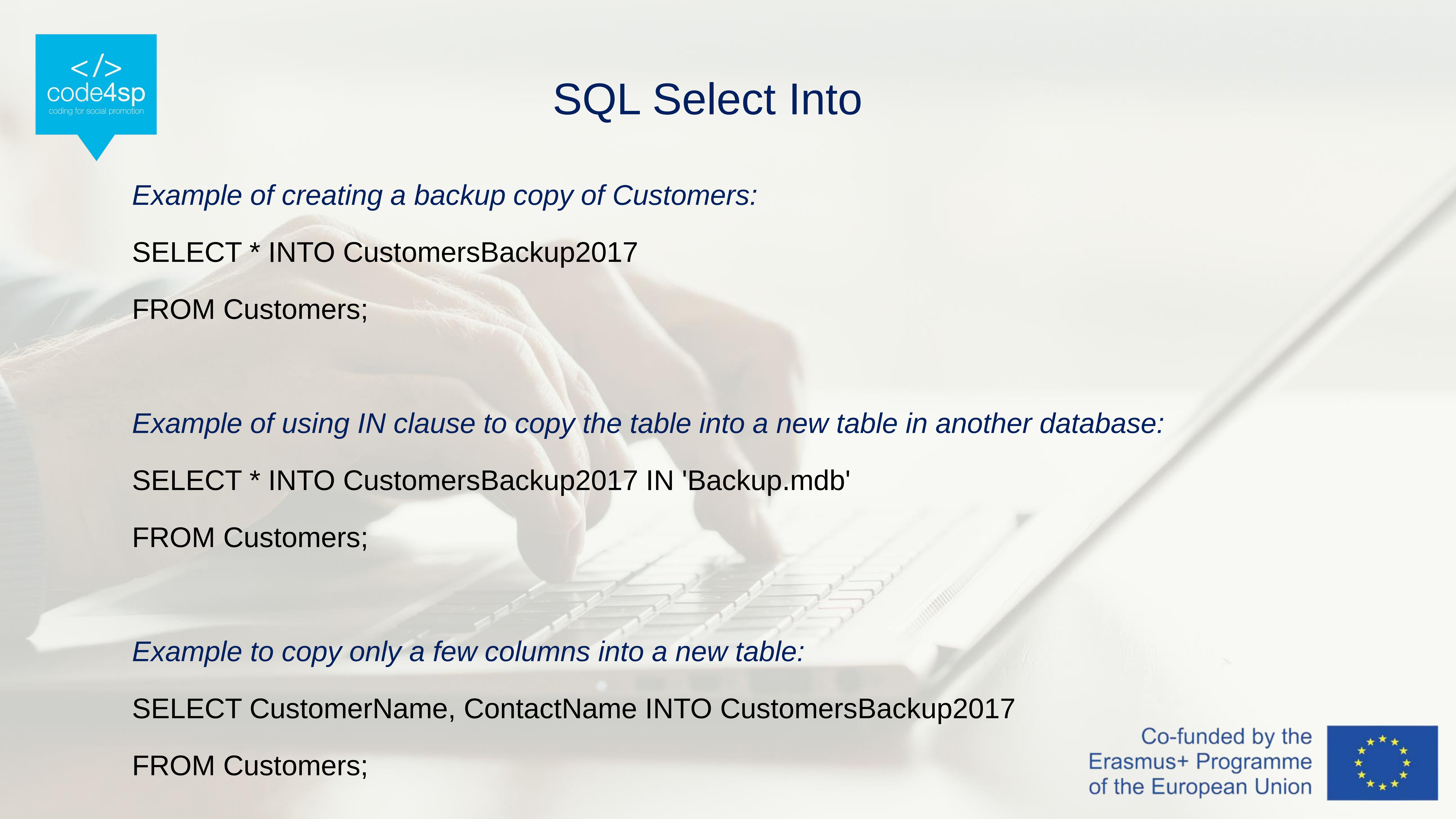
Syntax to copy all columns into a new table:

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

Syntax to copy only some columns into a new table:

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

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



SQL Select Into

Example of creating a backup copy of Customers:

```
SELECT * INTO CustomersBackup2017

FROM Customers;
```

Example of using IN clause to copy the table into a new table in another database:

```
SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'

FROM Customers;
```

Example to copy only a few columns into a new table:

```
SELECT CustomerName, ContactName INTO CustomersBackup2017

FROM Customers;
```

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

SQL Select Into

Example to copy only the German customers into a new table:

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

Example to copy data from multiple tables into a new table:

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

SELECT INTO can also be used to create a new, empty table using the schema of another.

To do that, add a WHERE clause that returns no data:

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

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

SQL Insert Into Select

The INSERT INTO SELECT statement copies data from one table and inserts it into another. It requires the data types in the source and target table to match.

Syntax to copy all columns from one table to another:

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

Syntax to copy only some columns from one table to another:

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

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

SQL Insert Into Select

Example 1: Copies Suppliers into Customers (note that the columns that are not filled with data will contain null values)

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

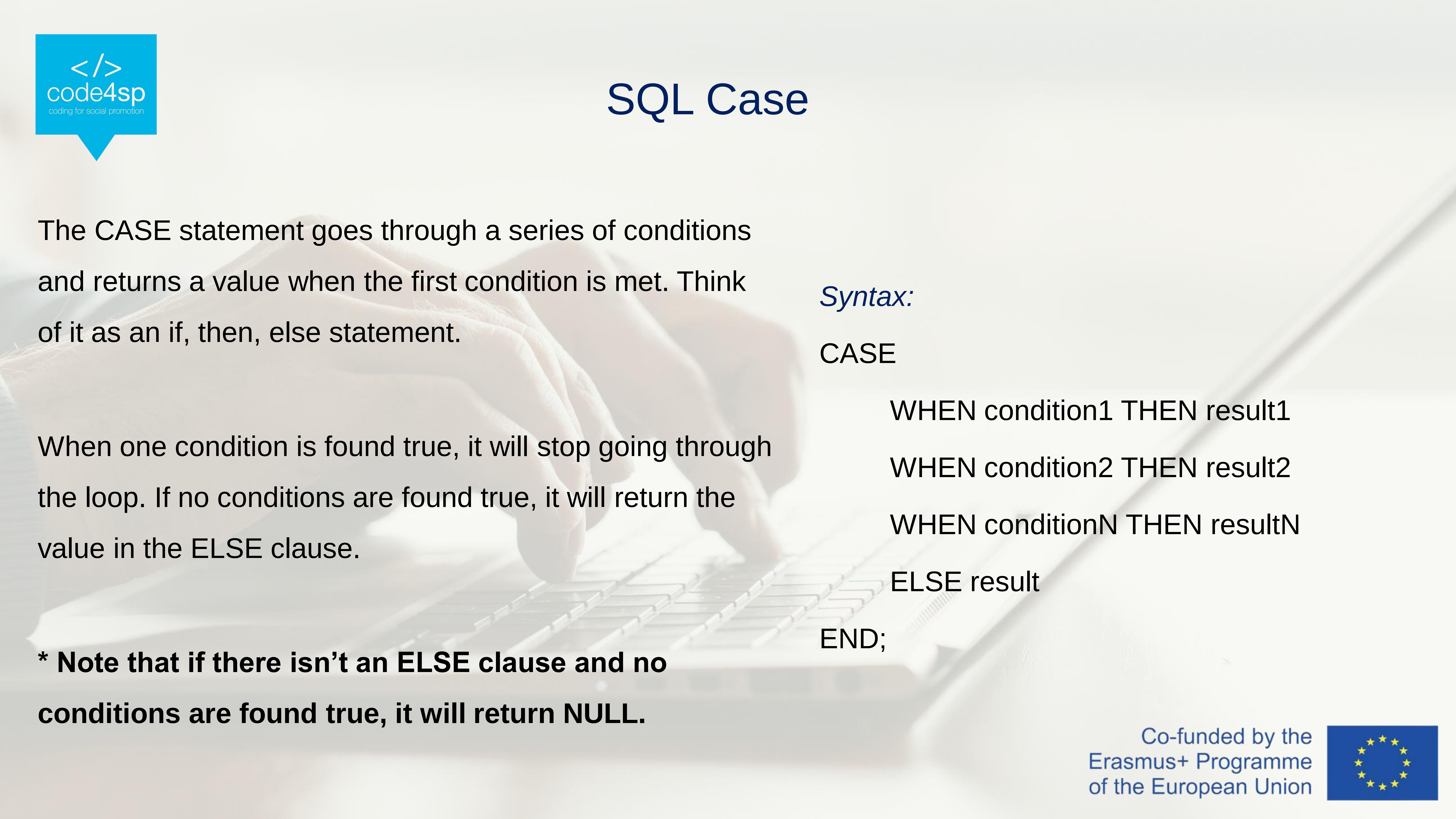
Example 2: Copies Suppliers into Customers to fill all columns

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

Example 3: Copies only the German suppliers into 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



code4sp
coding for social promotion

SQL Case

The CASE statement goes through a series of conditions and returns a value when the first condition is met. Think of it as an if, then, else statement.

When one condition is found true, it will stop going through the loop. If no conditions are found true, it will return the value in the ELSE clause.

*** Note that if there isn't an ELSE clause and no conditions are found true, it will return NULL.**

Syntax:

```
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

SQL Case

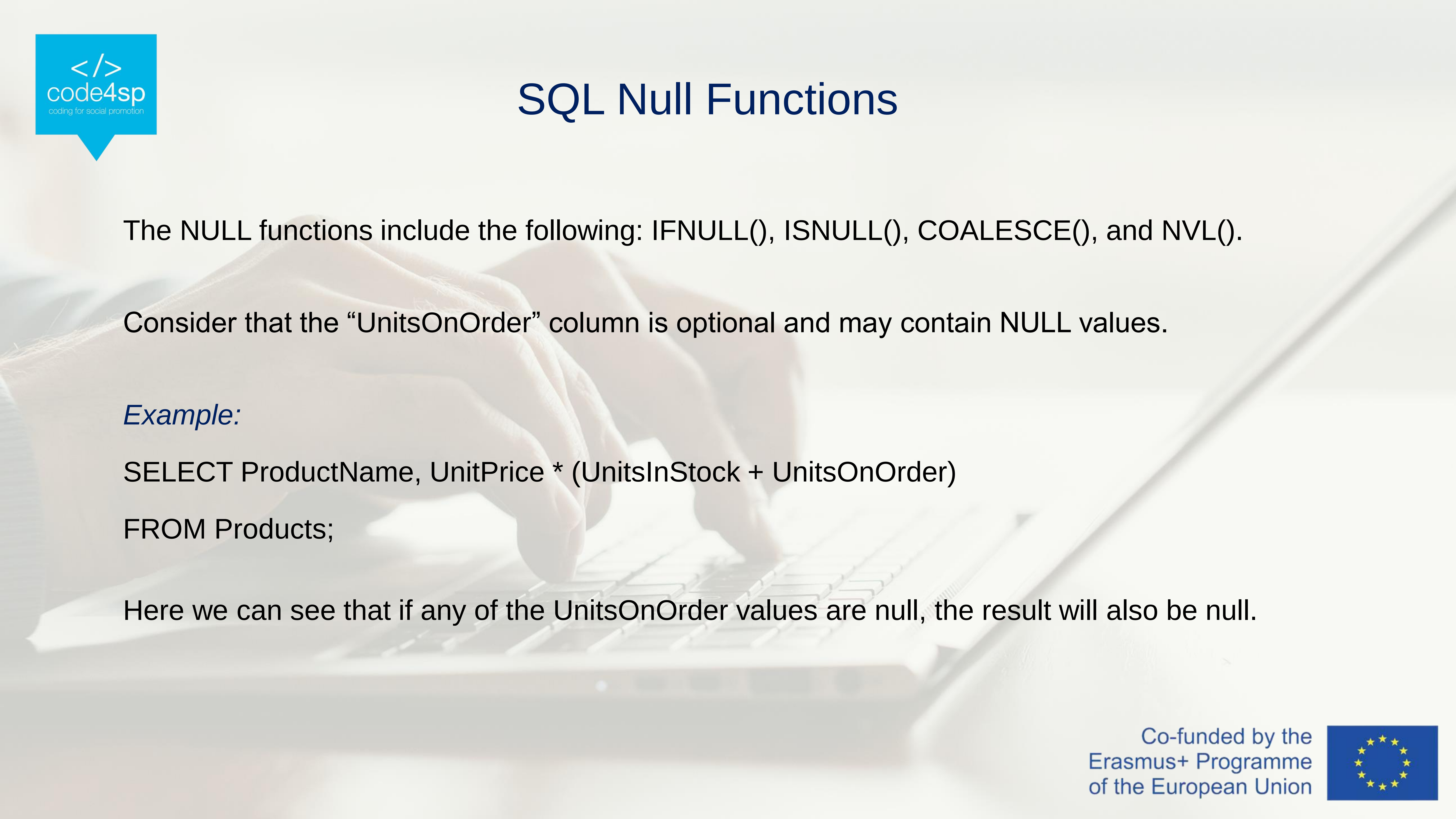
Example 1: Goes through a series of conditions and return a value when the first condition is met

```
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;
```

Example 2: Orders the customers by City. If City is NULL, it will be ordered by 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



SQL Null Functions

The NULL functions include the following: IFNULL(), ISNULL(), COALESCE(), and NVL().

Consider that the “UnitsOnOrder” column is optional and may contain NULL values.

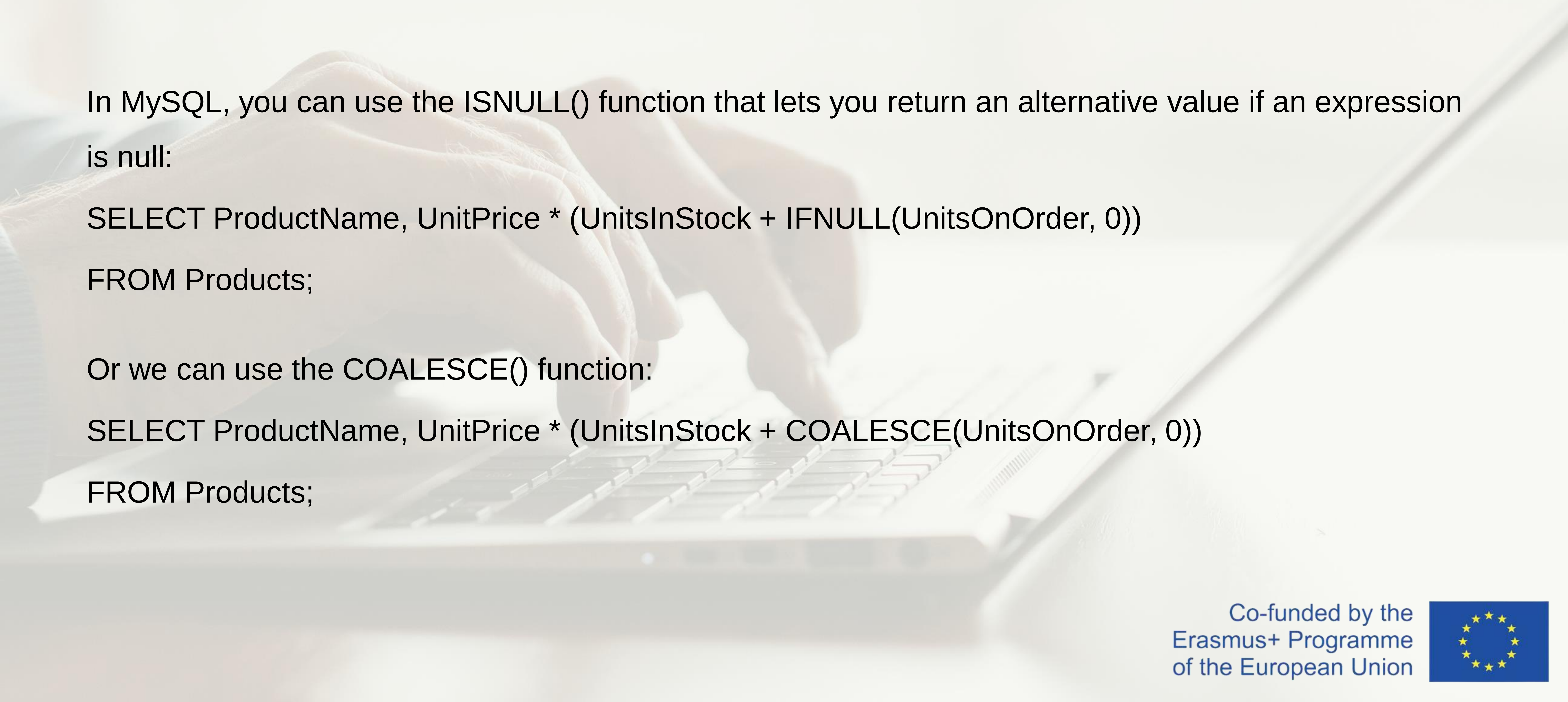
Example:

```
SELECT ProductName, UnitPrice * (UnitsInStock + UnitsOnOrder)

FROM Products;
```

Here we can see that if any of the UnitsOnOrder values are null, the result will also be null.

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



SQL Null Functions

In MySQL, you can use the ISNULL() function that lets you return an alternative value if an expression is null:

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

Or we can use the COALESCE() function:

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

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

SQL Null Functions

In SQL Server, the ISNULL() function does the same thing as in MySQL:

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

In MS Access IsNull() function returns TRUE(-1) if the expression is a null value, otherwise FALSE (0):

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

In Oracle, the NVL() function does the same thing:

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

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

SQL Comments

SQL Comments **explain SQL statement sections or prevent their execution.**

Single line comments in SQL start with - - (two dashes):

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

Or it can be used like this to ignore the end of the line:

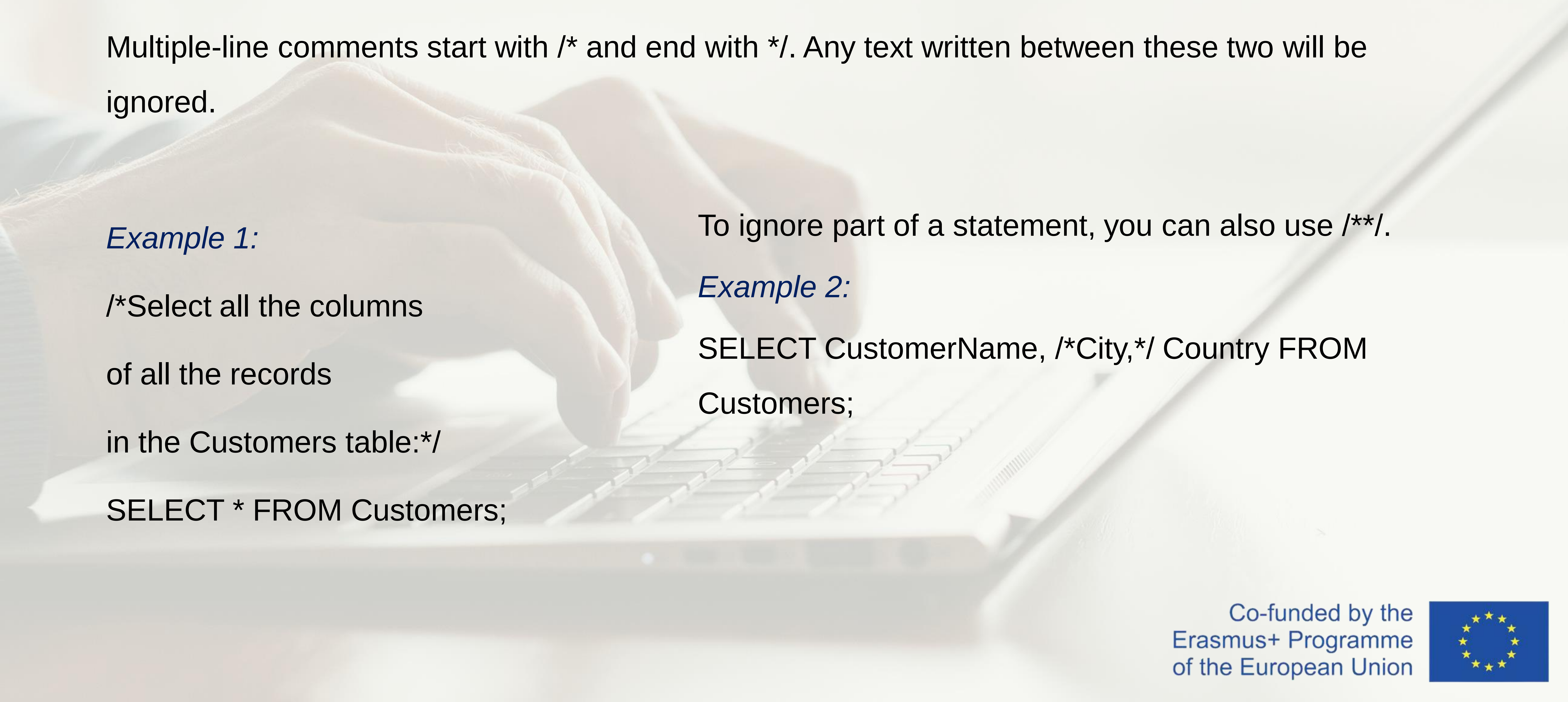
```
SELECT * FROM Customers -- WHERE City='Berlin';
```

Or to ignore a statement:

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

*** Note that the examples in this section are not supported in Firefox and Microsoft Edge, which are Microsoft Access databases. Comments are generally not supported in Microsoft Access databases.**

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



SQL Comments

Multiple-line comments start with /* and end with */. Any text written between these two will be ignored.

Example 1:

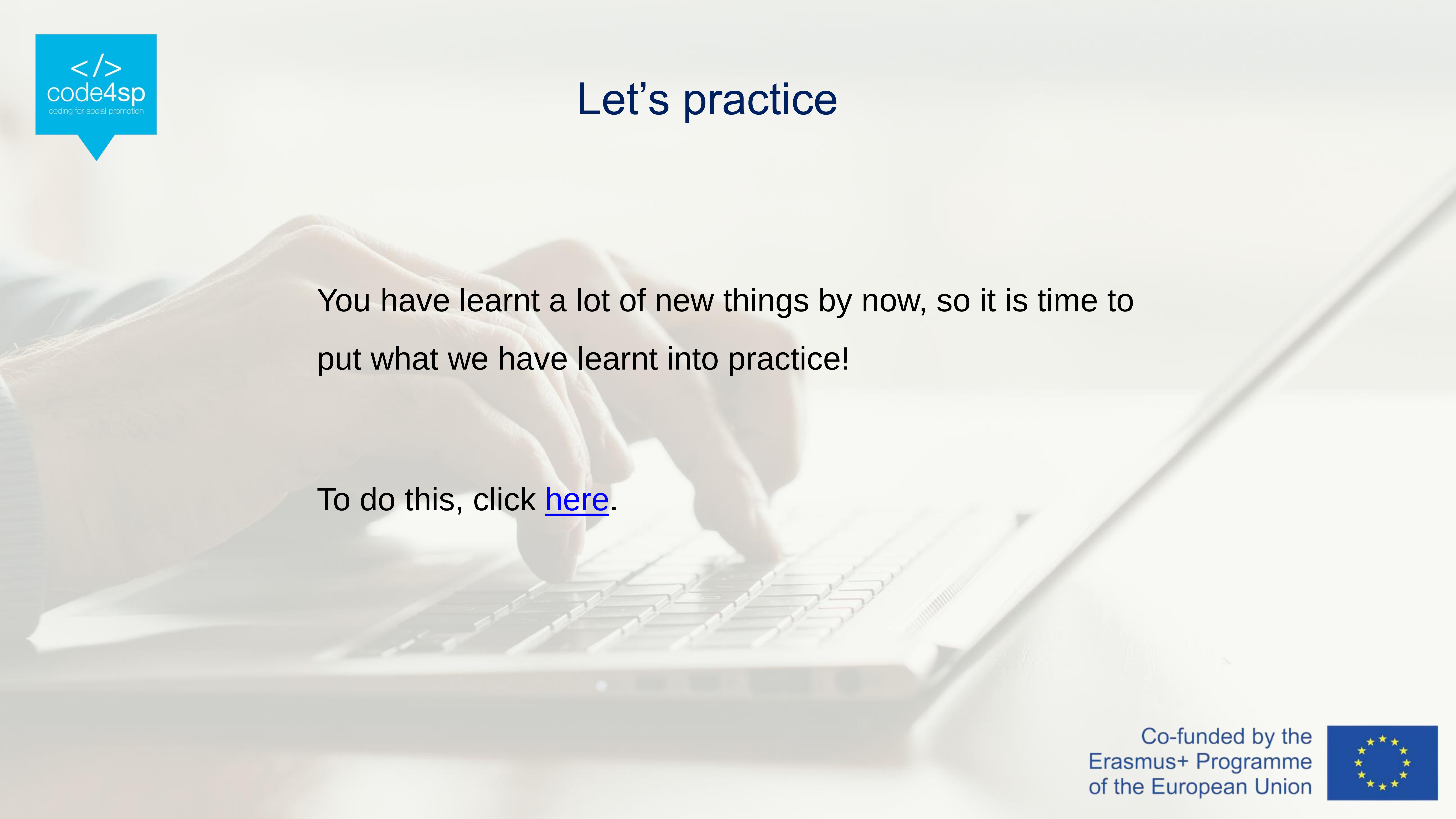
```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```

To ignore part of a statement, you can also use /**/.

Example 2:

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

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



Let’s practice

You have learnt a lot of new things by now, so it is time to put what we have learnt into practice!

To do this, click here.

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

code4sp
coding for social promotion

THANK YOU!

NEXT CHAPTER: SQL Database

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