Some of the Most Important SQL Commands

* **SELECT** - extracts data from a database
* **UPDATE** - updates data in a database
* **DELETE** - deletes data from a database
* **INSERT INTO** - inserts new data into a database
* **CREATE DATABASE** - creates a new database
* **ALTER DATABASE** - modifies a database
* **CREATE TABLE** - creates a new table
* **ALTER TABLE** - modifies a table
* **DROP TABLE** - deletes a table
* **CREATE INDEX** - creates an index (search key)
* **DROP INDEX** - deletes an index

**The SQL SELECT Statement**

The SELECT statement is used to select data from a database.

### SELECT Syntax

**1)**

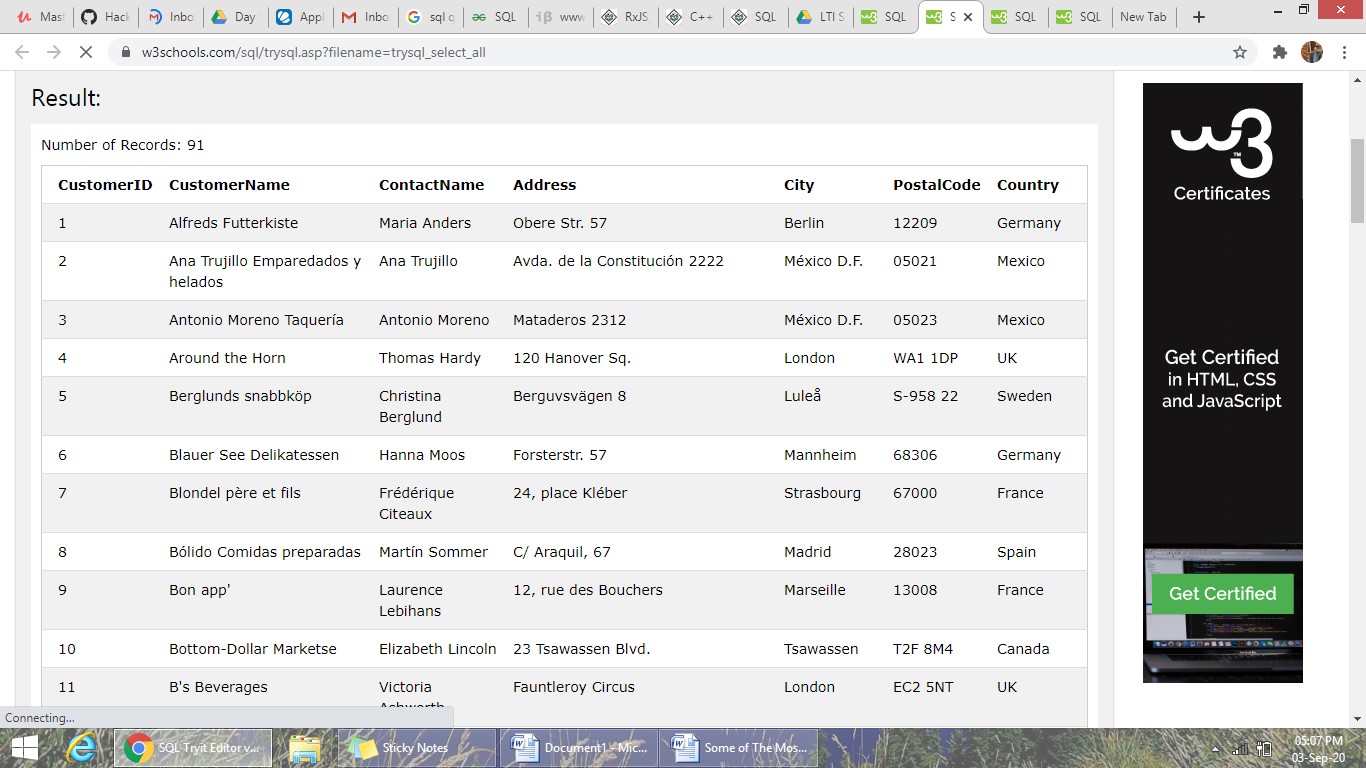
### SELECT column1, column2, ... FROM table\_name;

### Here, column1, column2, ... are the field names of the table you want to select data from.

### 2)

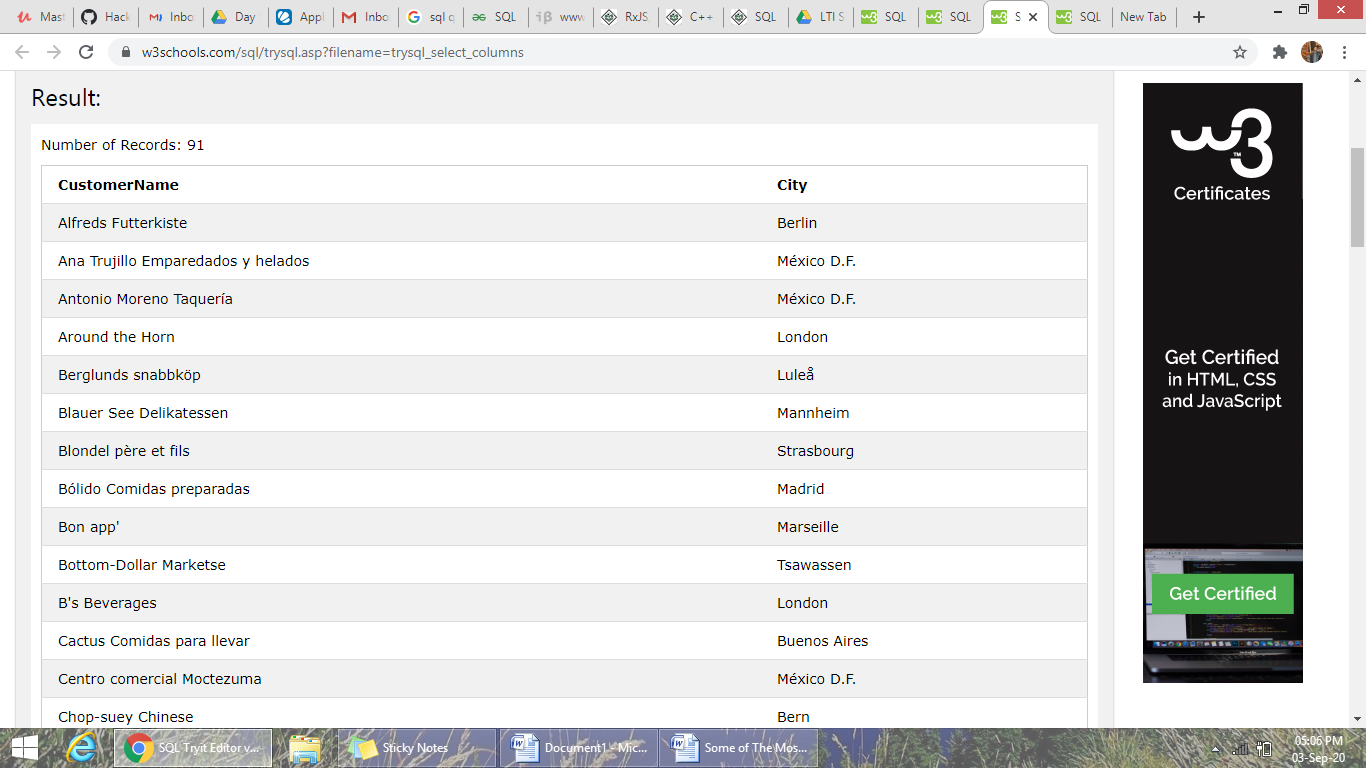
### If you want to select all the fields available in the table, use the following syntax:

### SELECT \* FROM table\_name;



**Example**

SELECT CustomerName, City FROM Customers;



**Exercise**

1)Write a statement that will select the City column from the Customers table

**SELECT City FROM** Customers;

The SQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

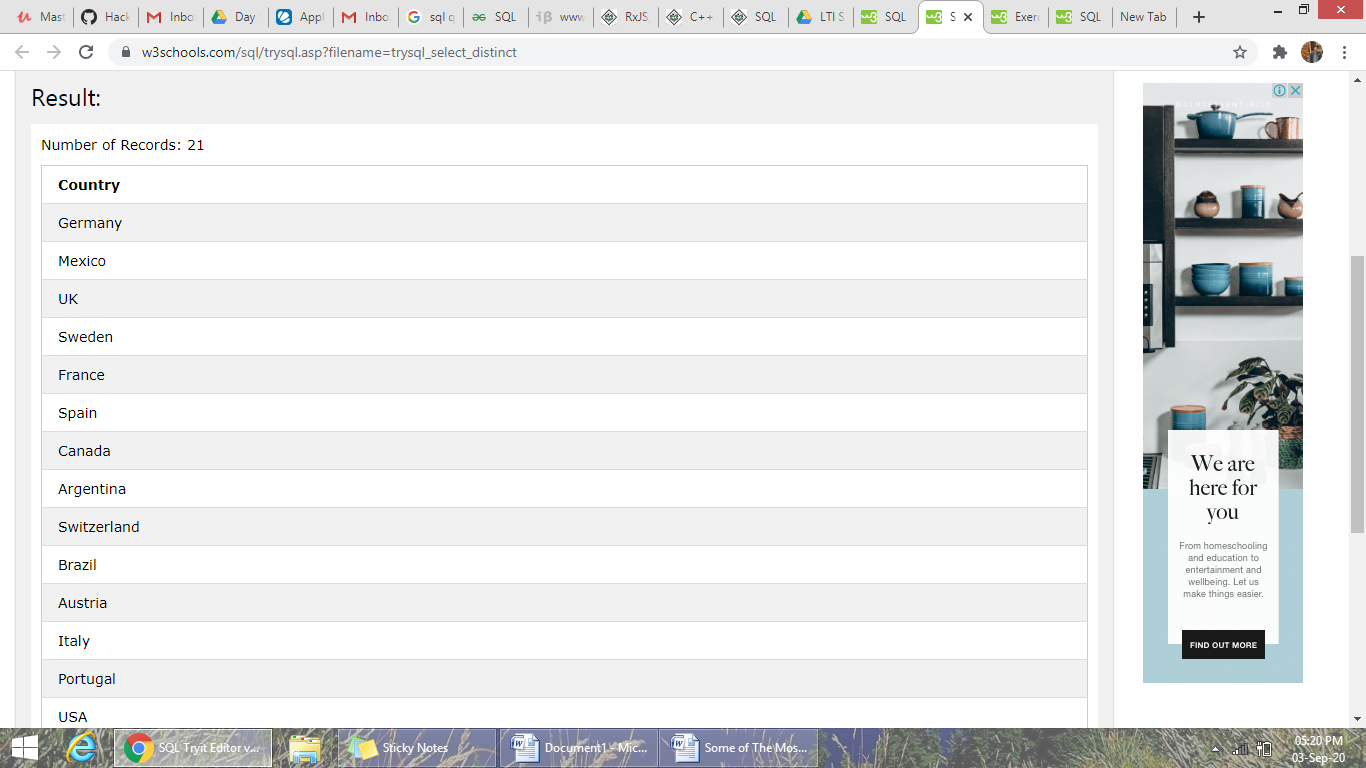
### SELECT DISTINCT Syntax

SELECT DISTINCT column1, column2,  
FROM table\_name;

**Exercise**

1. Select all the different values from the Country column in the Customers table.

**SELECT DISTINCT** Country FROM Customers;

****

SQL WHERE Clause

## The SQL WHERE Clause

The WHERE clause is used to filter records.

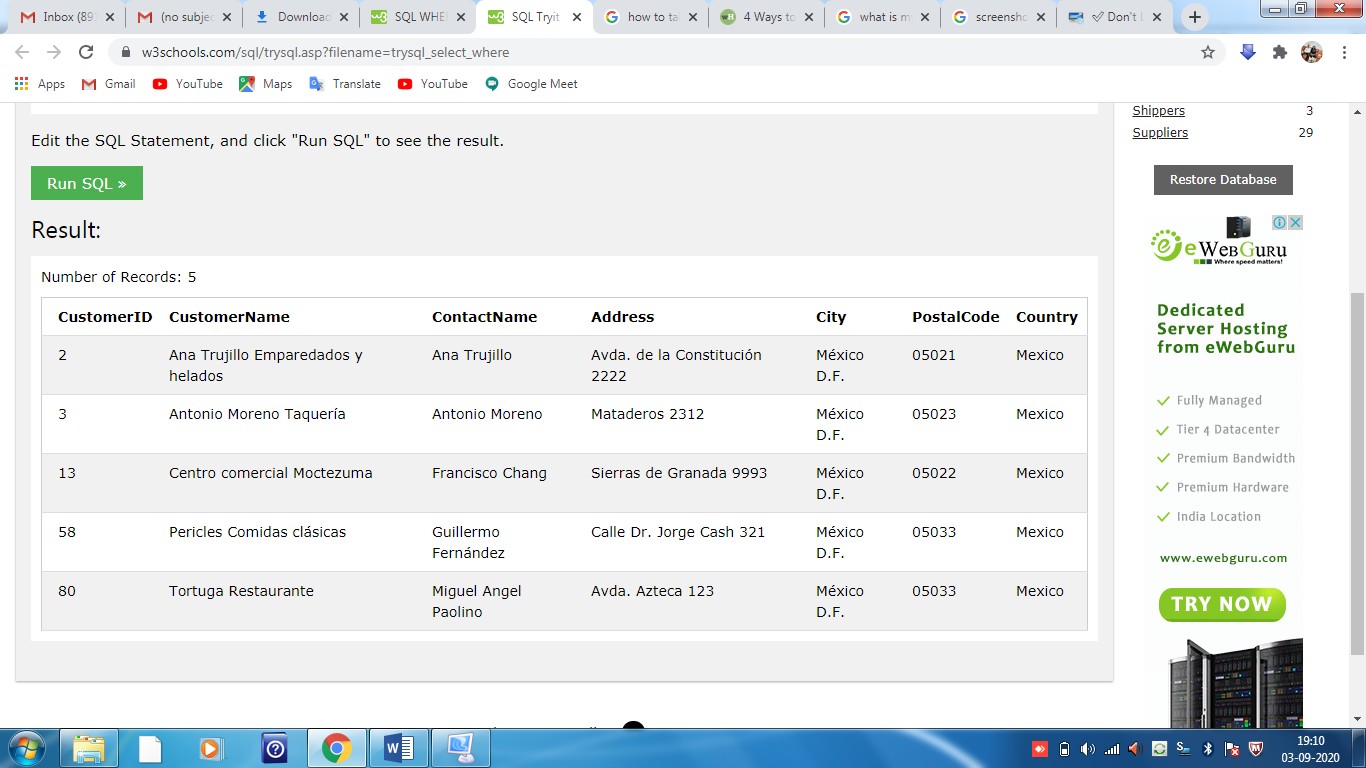
The WHERE clause is used to extract only those records that fulfill a specified condition.

### WHERE Syntax

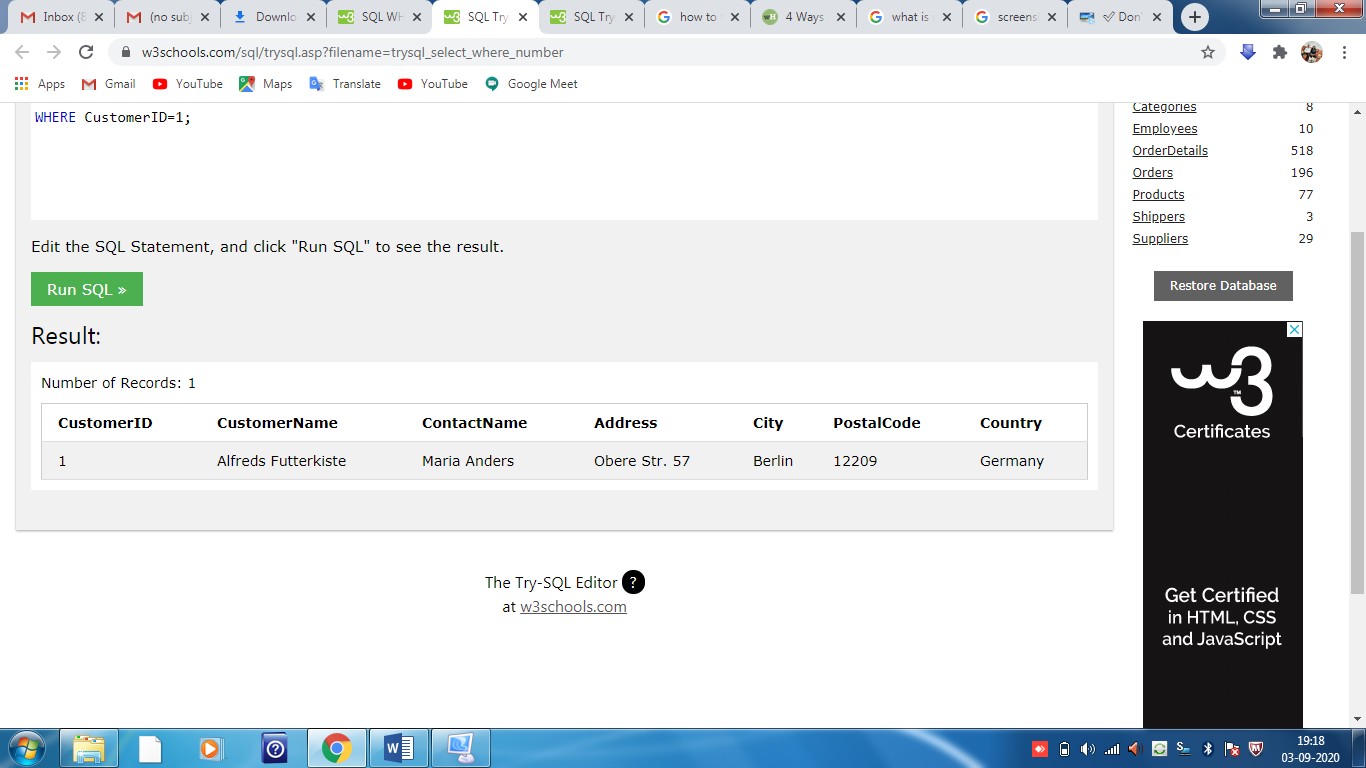
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

**Exercise**

1. SELECT \* FROM Customers  
   WHERE Country='Mexico';



1. SELECT \* FROM Customers  
   WHERE CustomerID=1;



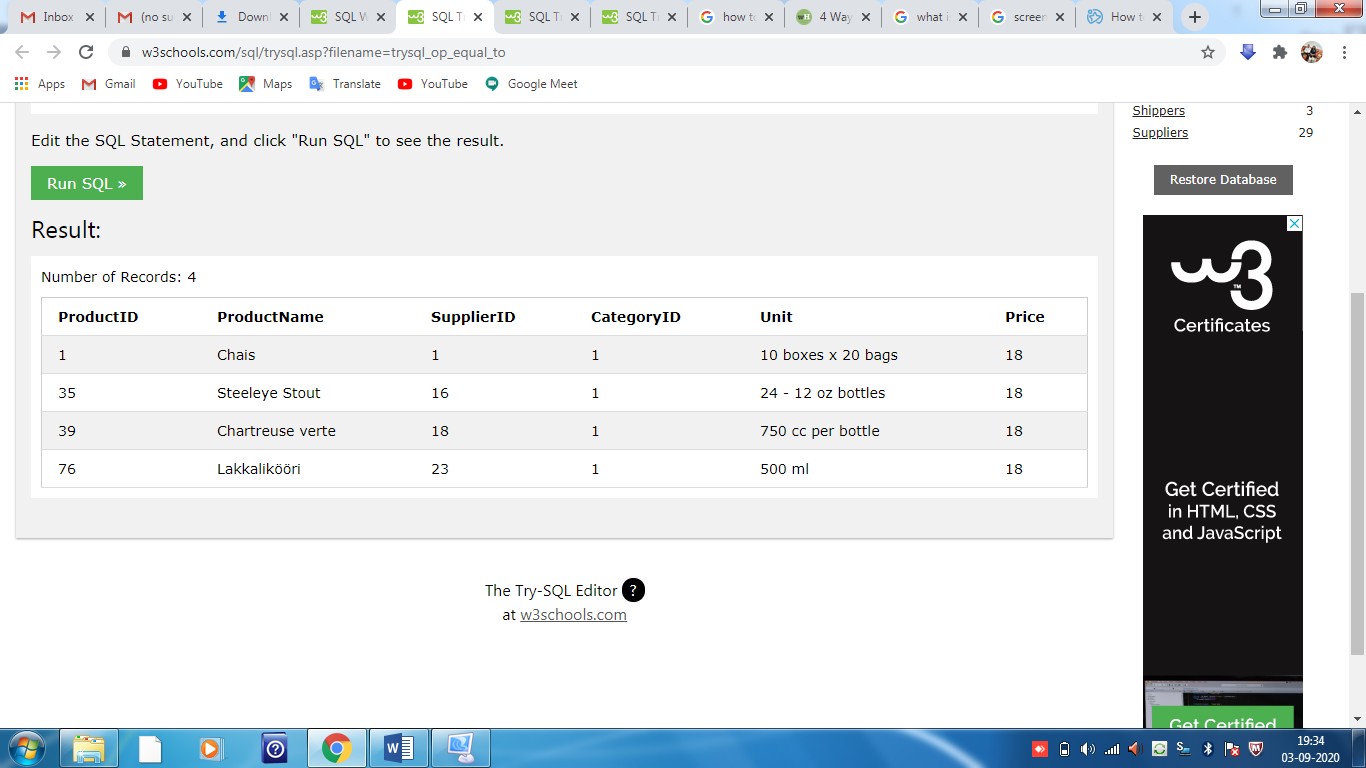
## Operators in the WHERE Clause

The following operators can be used in the WHERE clause:

|  |  |
| --- | --- |
| = | 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 |

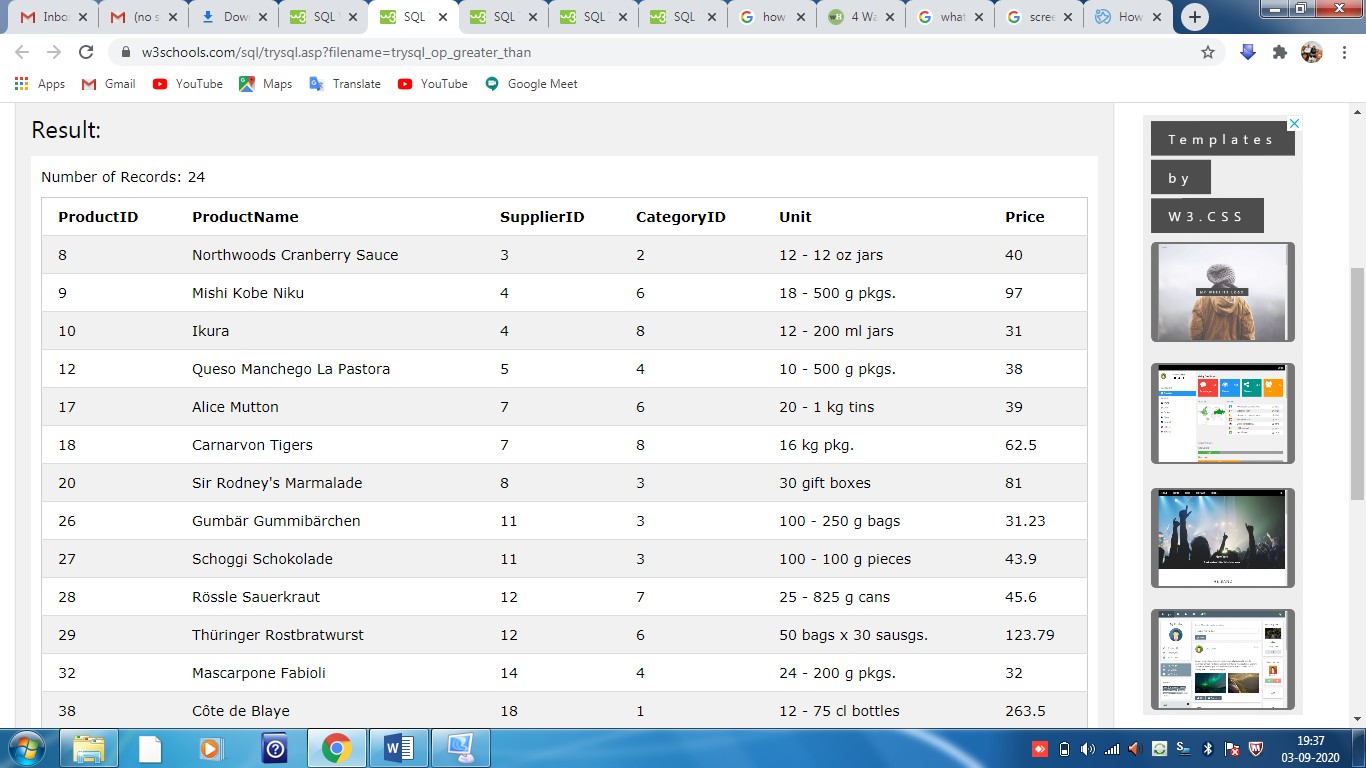
**SELECT \* FROM Products**

**WHERE Price = 18;**

****

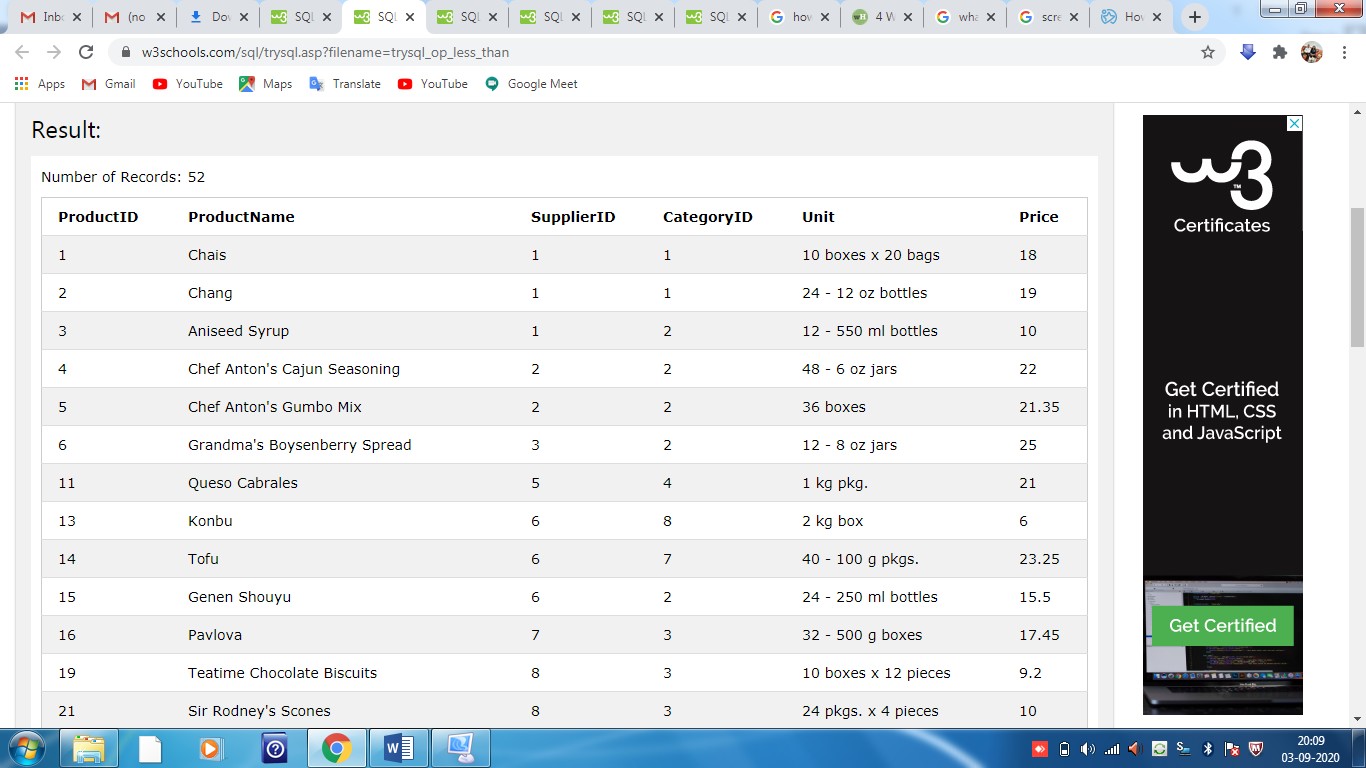
**SELECT \* FROM Products**

**WHERE Price > 30;**

****

**SELECT \* FROM Products**

**WHERE Price < 30;**

****

**SELECT \* FROM Products**

**WHERE Price >= 30;**

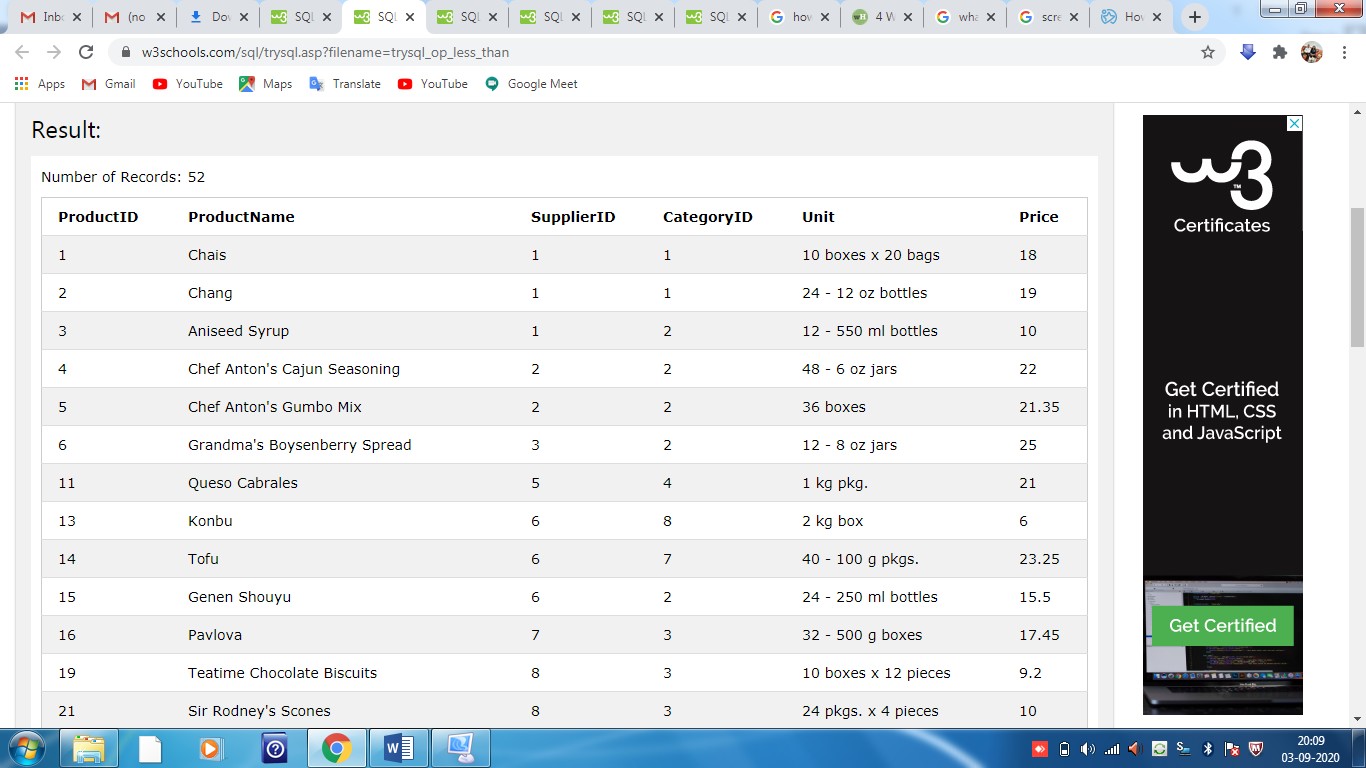
**SELECT \* FROM Products**

**WHERE Price <= 30;**

**SELECT \* FROM Products**

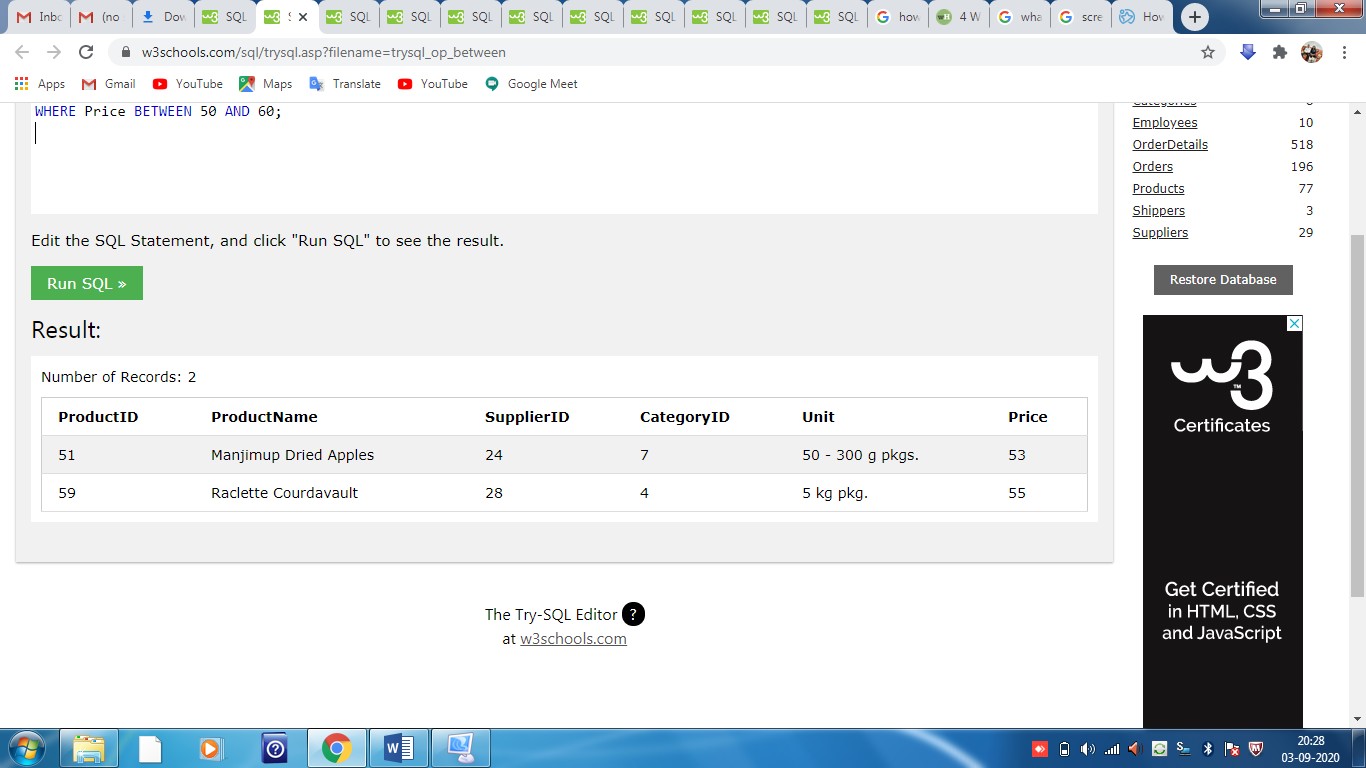
**WHERE Price <> 18;**

**Note - <> means !=(not equal)**

****

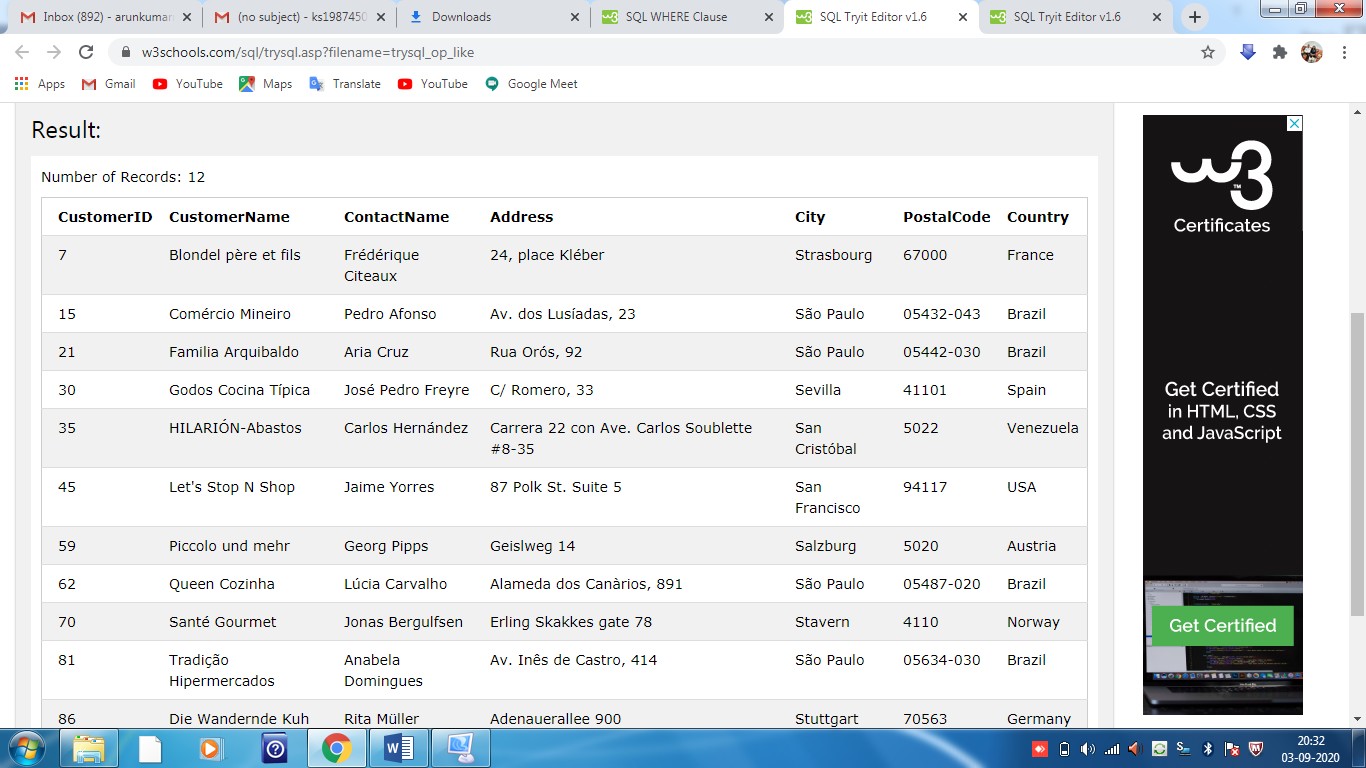
**SELECT \* FROM Products**

**WHERE Price BETWEEN 50 AND 60;**

****

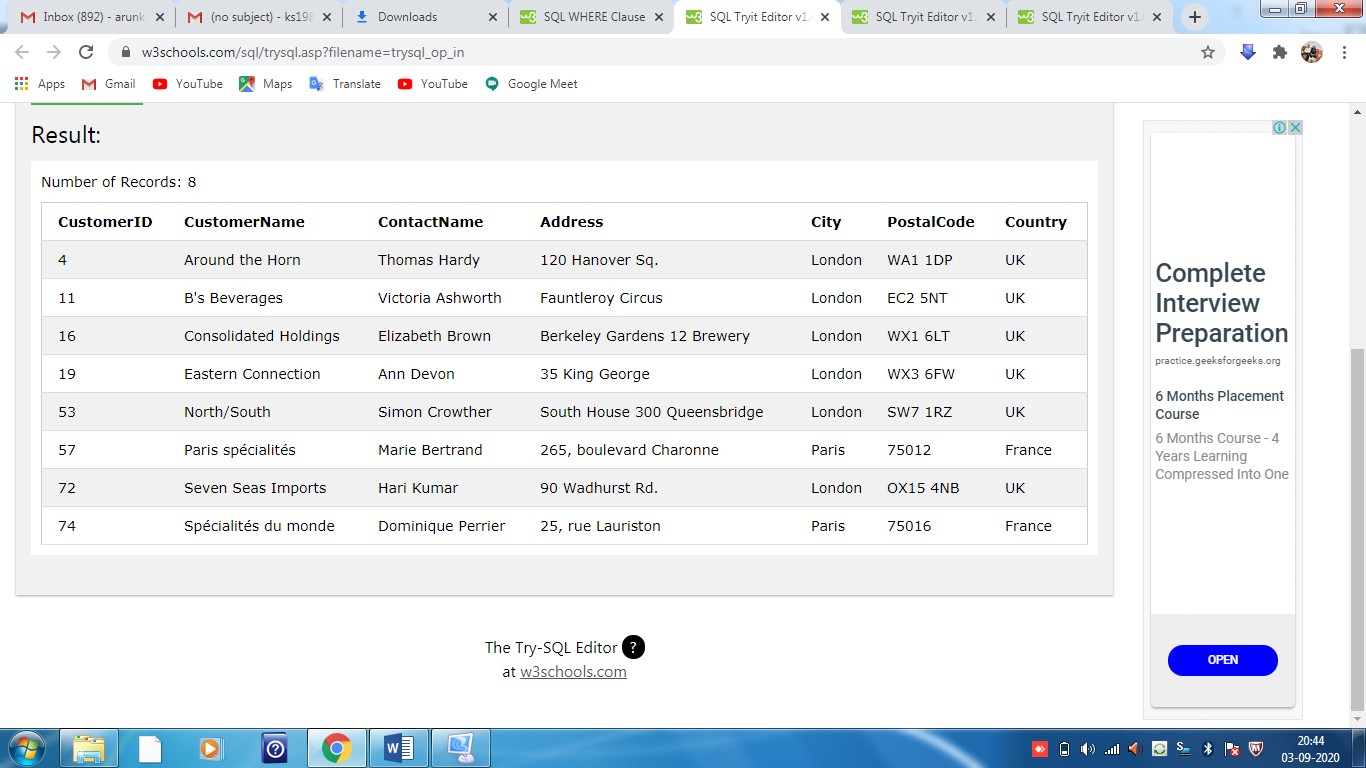
**SELECT \* FROM Customers**

**WHERE City LIKE 's%';**

****

**SELECT \* FROM Customers**

**WHERE City IN ('Paris','London');**

****

The SQL AND, OR and NOT Operators

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

The AND and OR operators are used to filter records based on more than one condition:

* The AND operator displays a record if all the conditions separated by AND are TRUE.
* The OR operator displays a record if any of the conditions separated by OR is TRUE.

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

### AND Syntax

SELECT column1, column2, ...  
FROM table\_name  
WHERE condition1 AND condition2 AND condition3 ...;

### OR Syntax

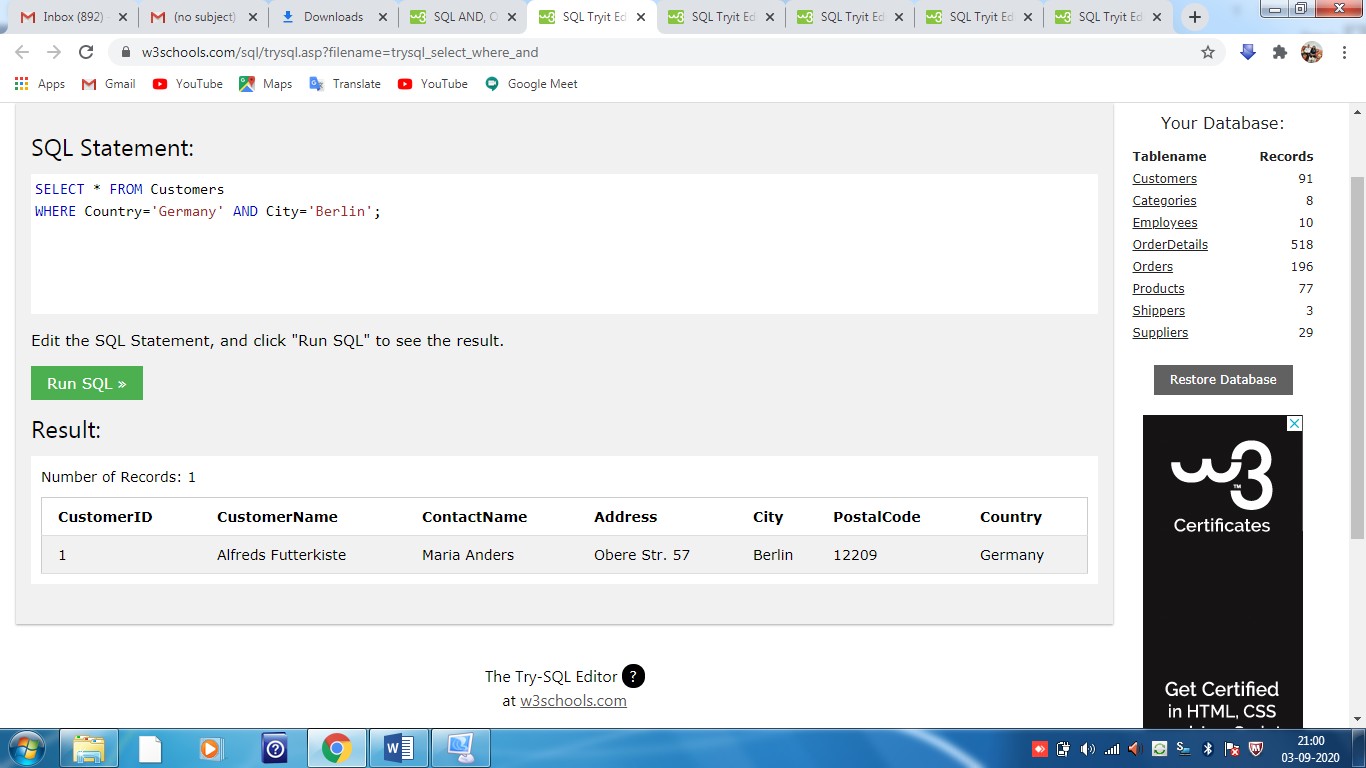
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition1 OR condition2 OR condition3 ...;

### NOT Syntax

SELECT column1, column2, ...  
FROM table\_name  
WHERE NOT condition;

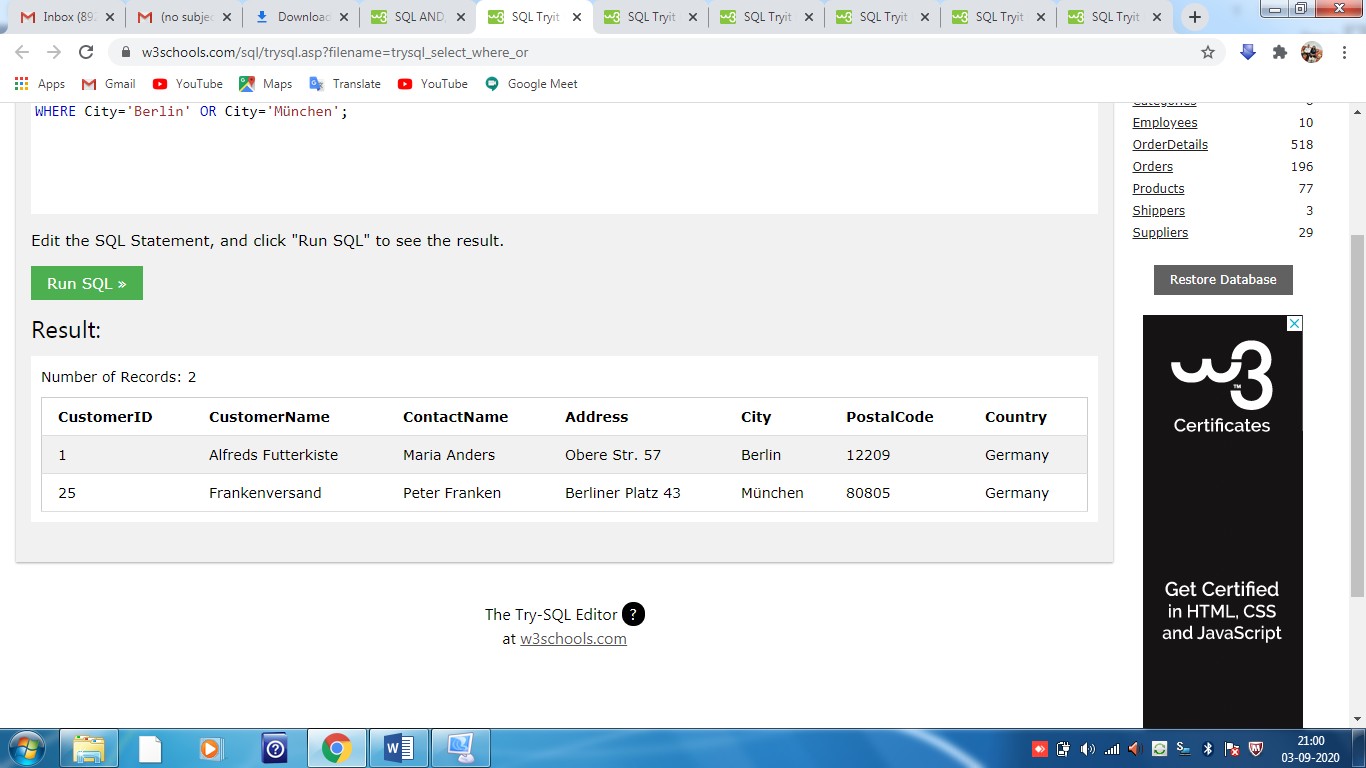
## AND Example

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



## OR Example

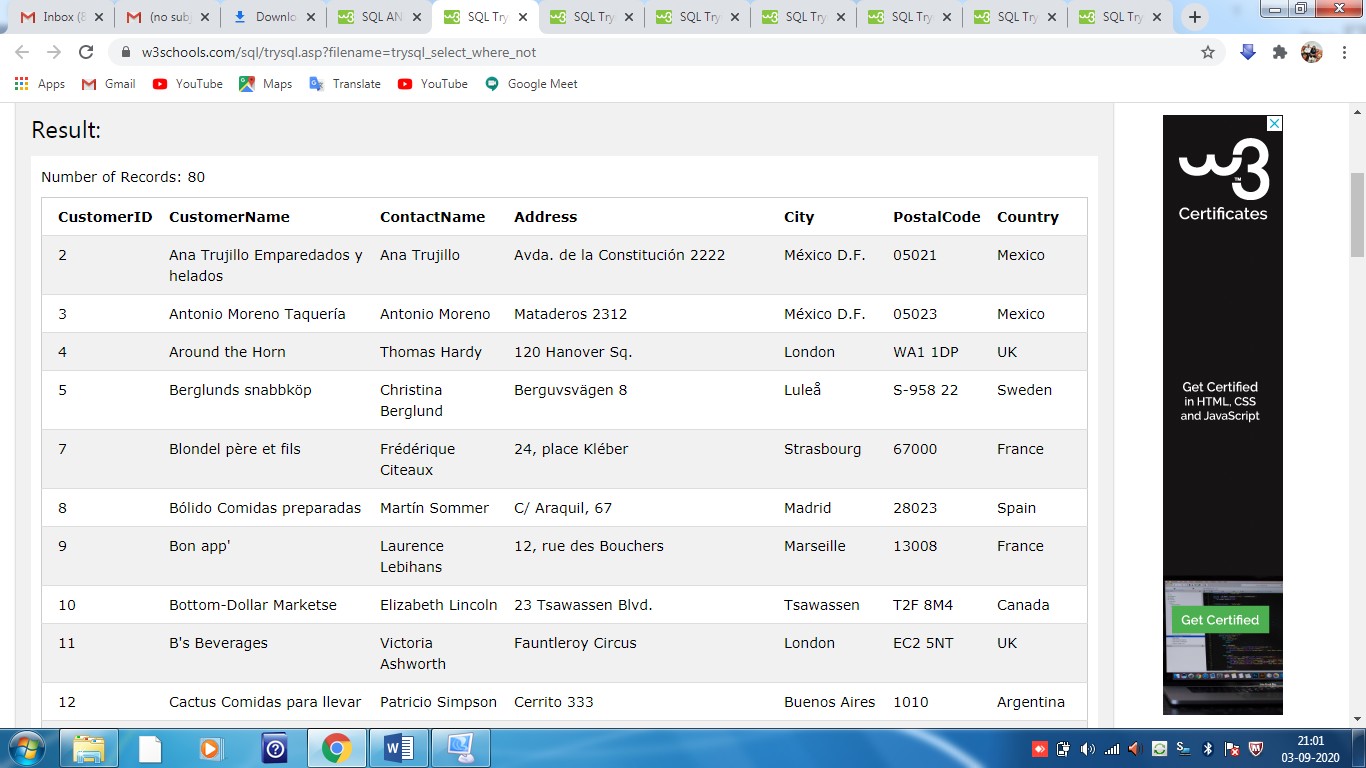
SELECT \* FROM Customers  
WHERE City='Berlin' OR City='München';

****

## NOT Example

The following SQL statement selects all fields from "Customers" where country is NOT "Germany":

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

****

SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

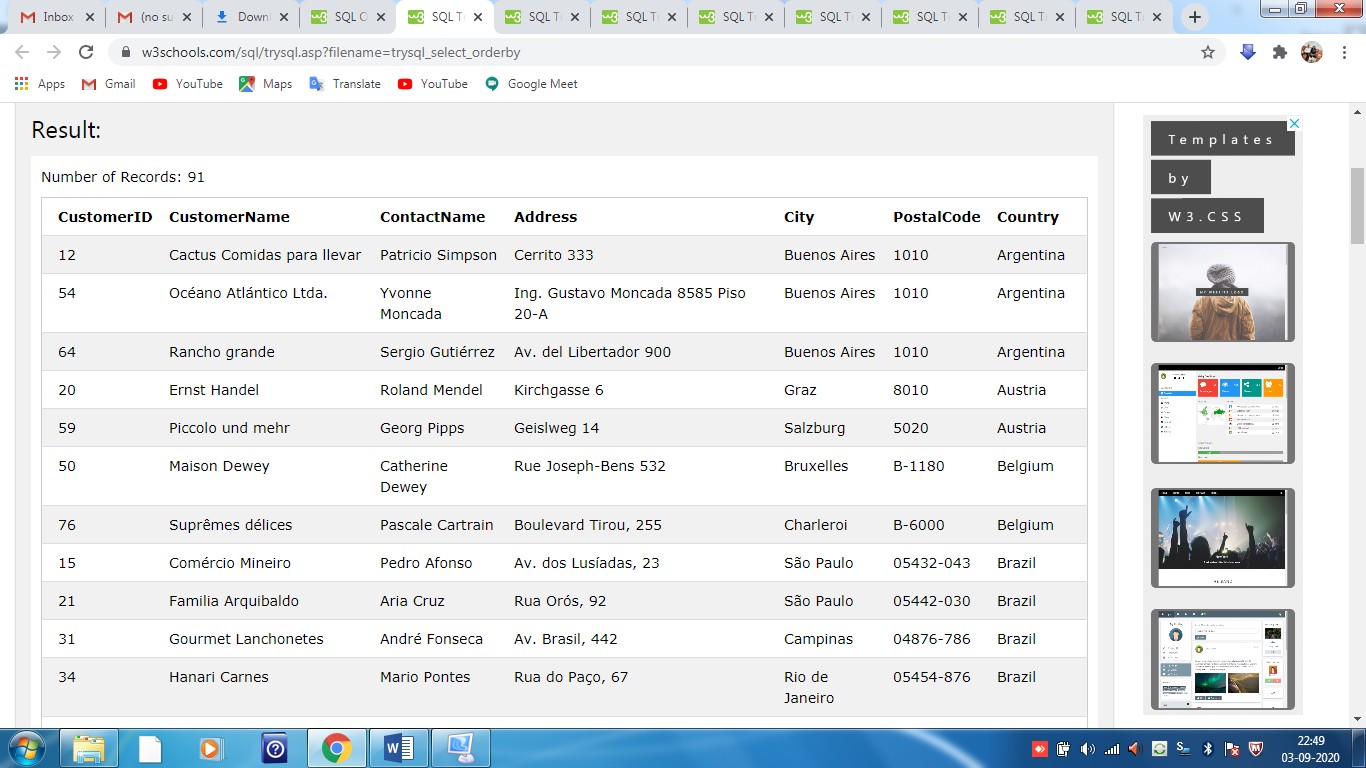
The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

### ORDER BY Syntax

SELECT column1, column2, …  
FROM table\_name  
ORDER BY column1, column2, ... ASC|DESC;

Example 1 - ORDER BY

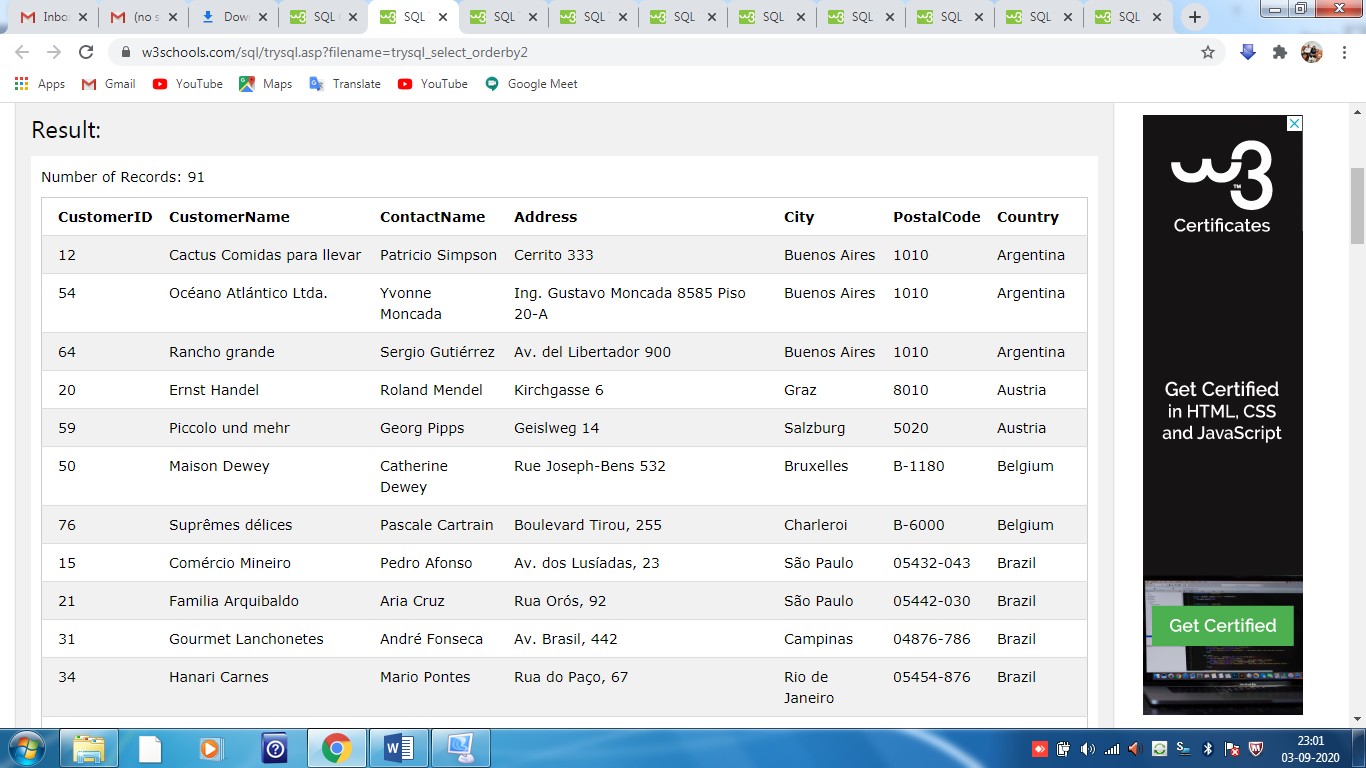
SELECT \* FROM Customers  
ORDER BY Country;



## Example 2 - ORDER BY Several Columns

## The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

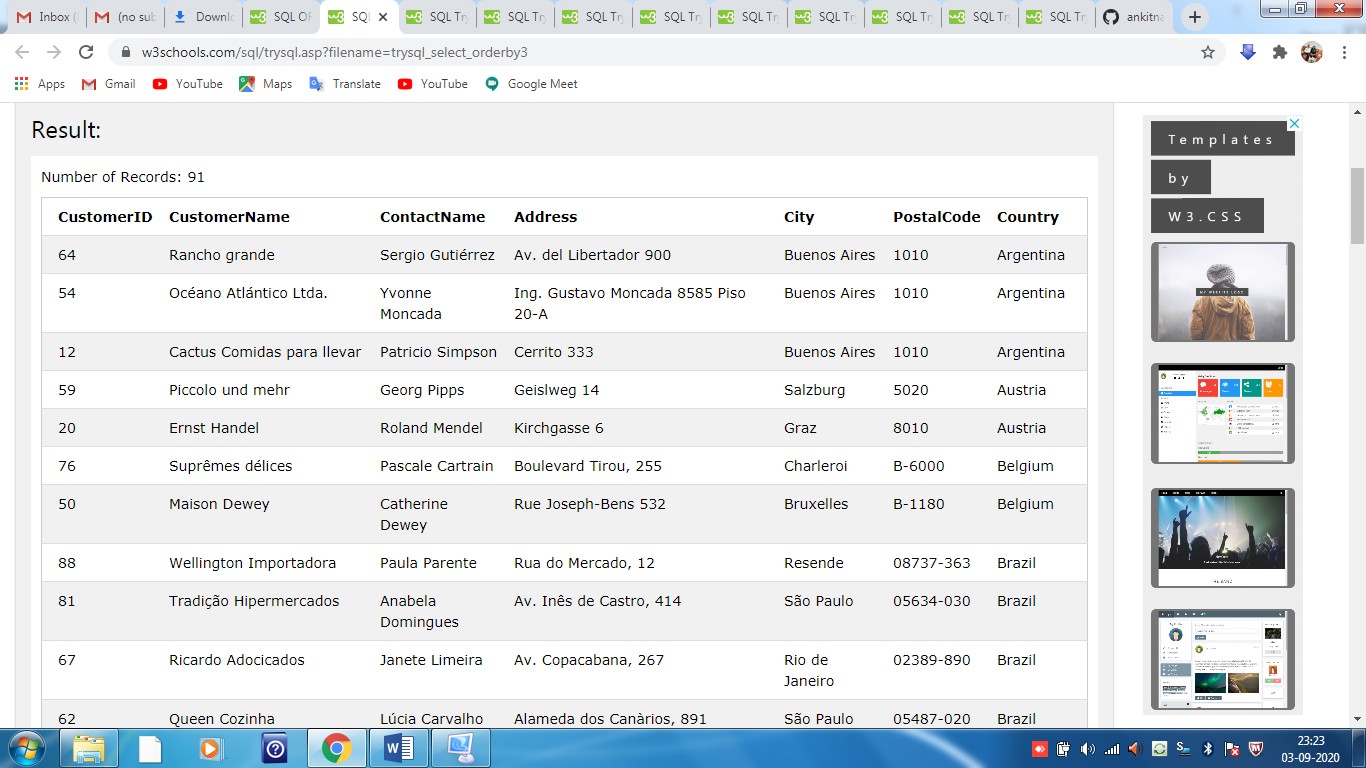
## SELECT \* FROM Customers ORDER BY Country, CustomerName;



## Example 2 - ORDER BY Several Columns

The following SQL statement selects all customers from the "Customers" table, sorted **ascending** by the **"Country"** and **descending** by the **"CustomerName"** column:

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

****

SQL INSERT INTO Statement

### The INSERT INTO statement is used to insert new records in a table.

### INSERT INTO Syntax

The INSERT INTO statement is used to insert new records in a table.

It is possible to write the INSERT INTO statement in two ways.

The first way specifies both the column names and the values to be inserted:

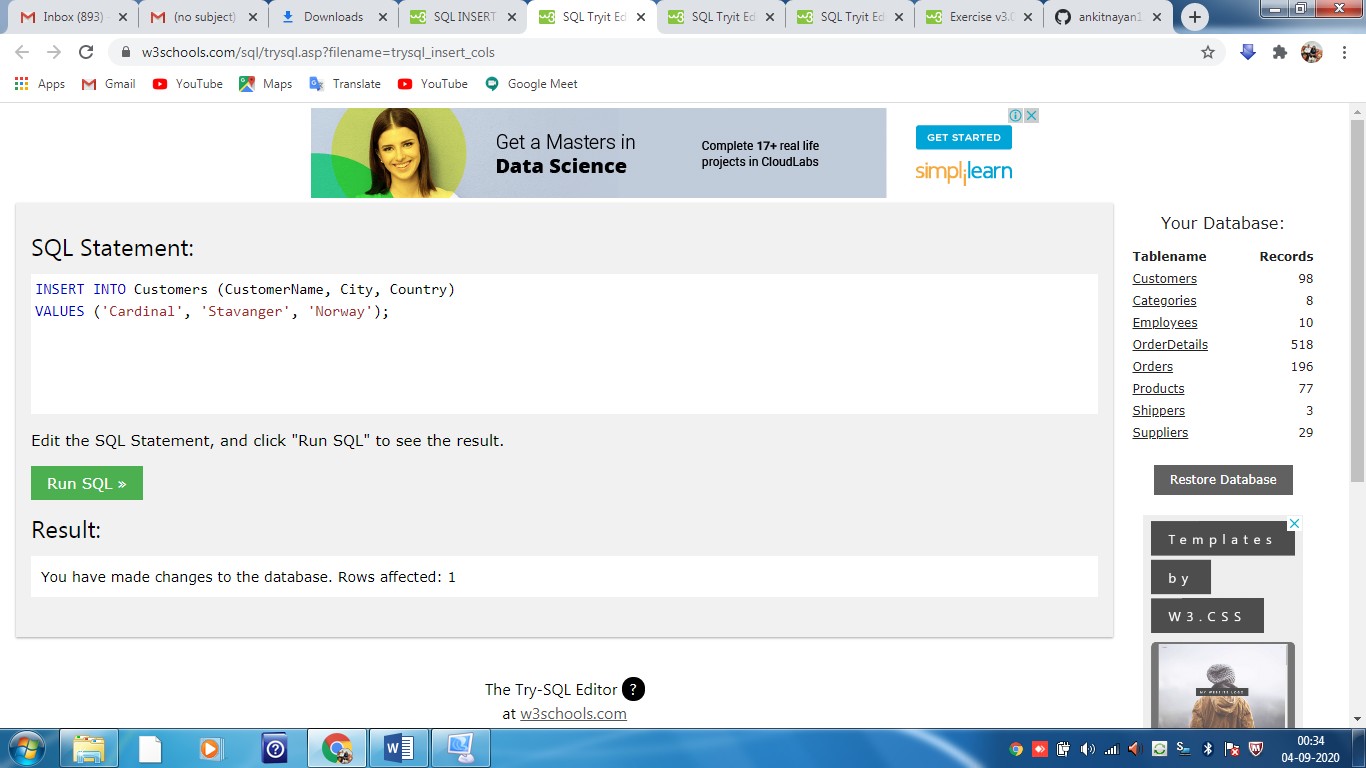
INSERT INTO table\_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. The INSERT INTO syntax would be as follows:

INSERT INTO table\_name  
VALUES (value1, value2, value3, ...);

## EXAMPLE 1-INSERT INTO Example

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



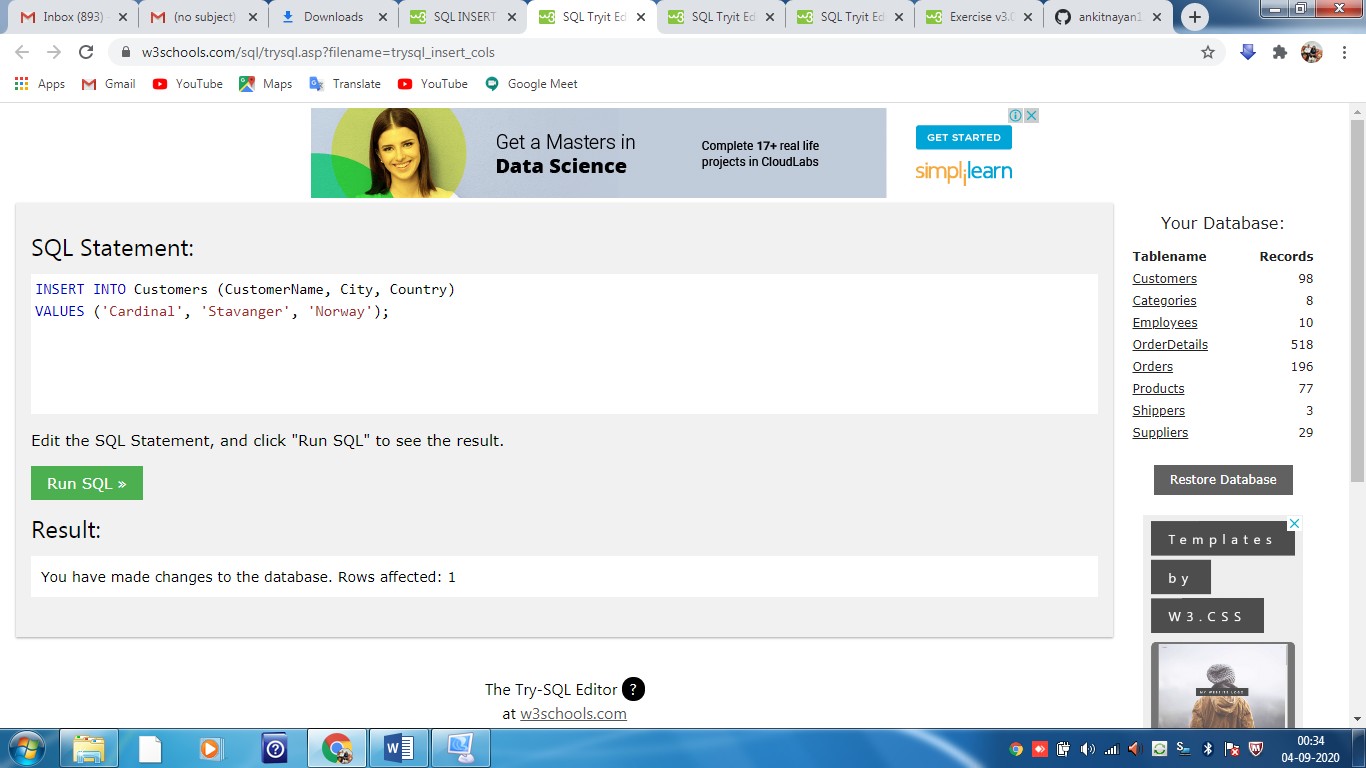
## 

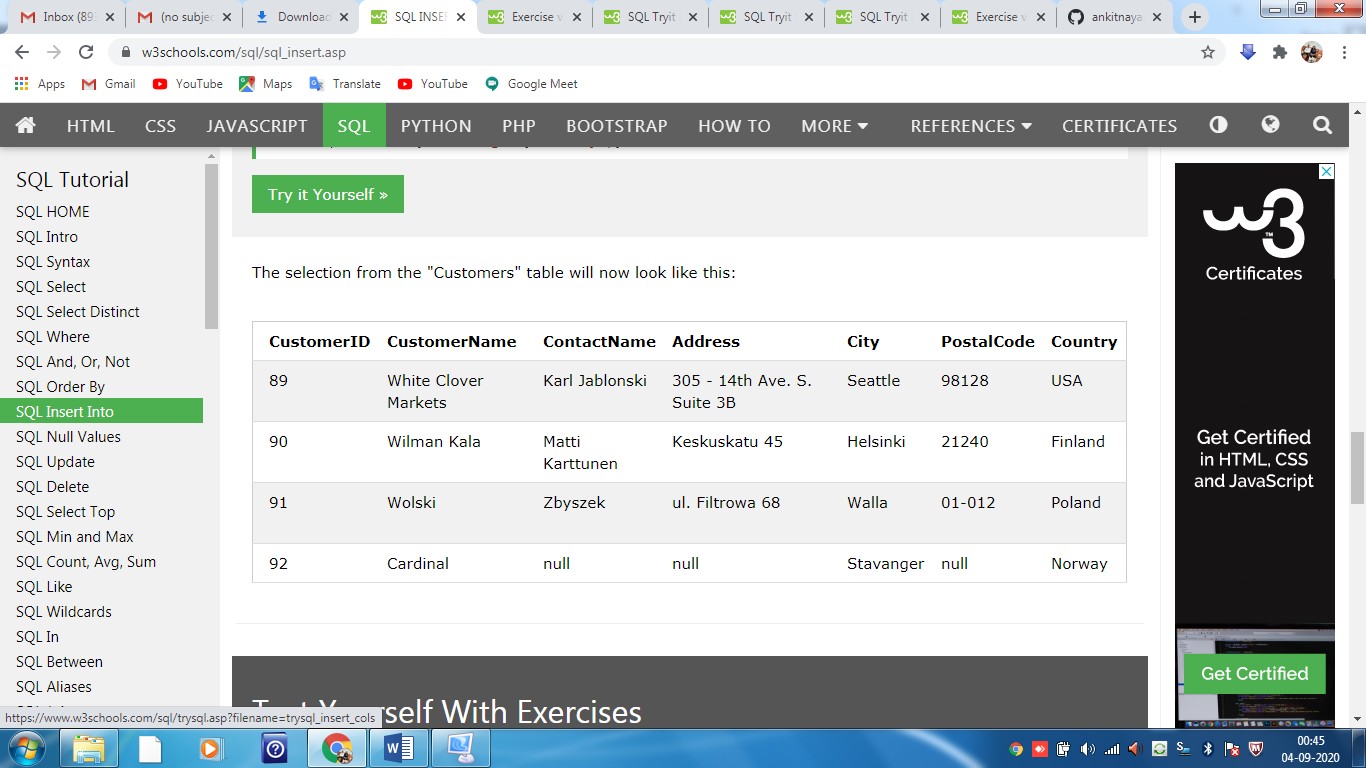
## EXAMPLE 2- Insert Data Only in Specified Columns

It is also possible to only insert data in specific columns.

The following SQL statement will insert a new record, but only insert data in the "CustomerName", "City", and "Country" columns (CustomerID will be updated automatically):

INSERT INTO Customers (CustomerName, City, Country)  
VALUES ('Cardinal', 'Stavanger', 'Norway');





SQL NULL Values

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

## How to Test for NULL Values?

## We will have to use the IS NULL and IS NOT NULL operators instead.

### IS NULL Syntax

## SELECT column\_namesFROM table\_name WHERE column\_name IS NULL;

### IS NOT NULL Syntax

## SELECT column\_namesFROM table\_name WHERE column\_name IS NOT NULL;

## The IS NULL Operator

The IS NULL operator is used to test for empty values (NULL values).

The following SQL lists all customers with a NULL value in the "Address" field:

Example

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

## 

## The IS NOT NULL Operator

The IS NOT NULL operator is used to test for non-empty values (NOT NULL values).

The following SQL lists all customers with a value in the "Address" field:

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

## 

SQL UPDATE Statement

## The SQL UPDATE Statement

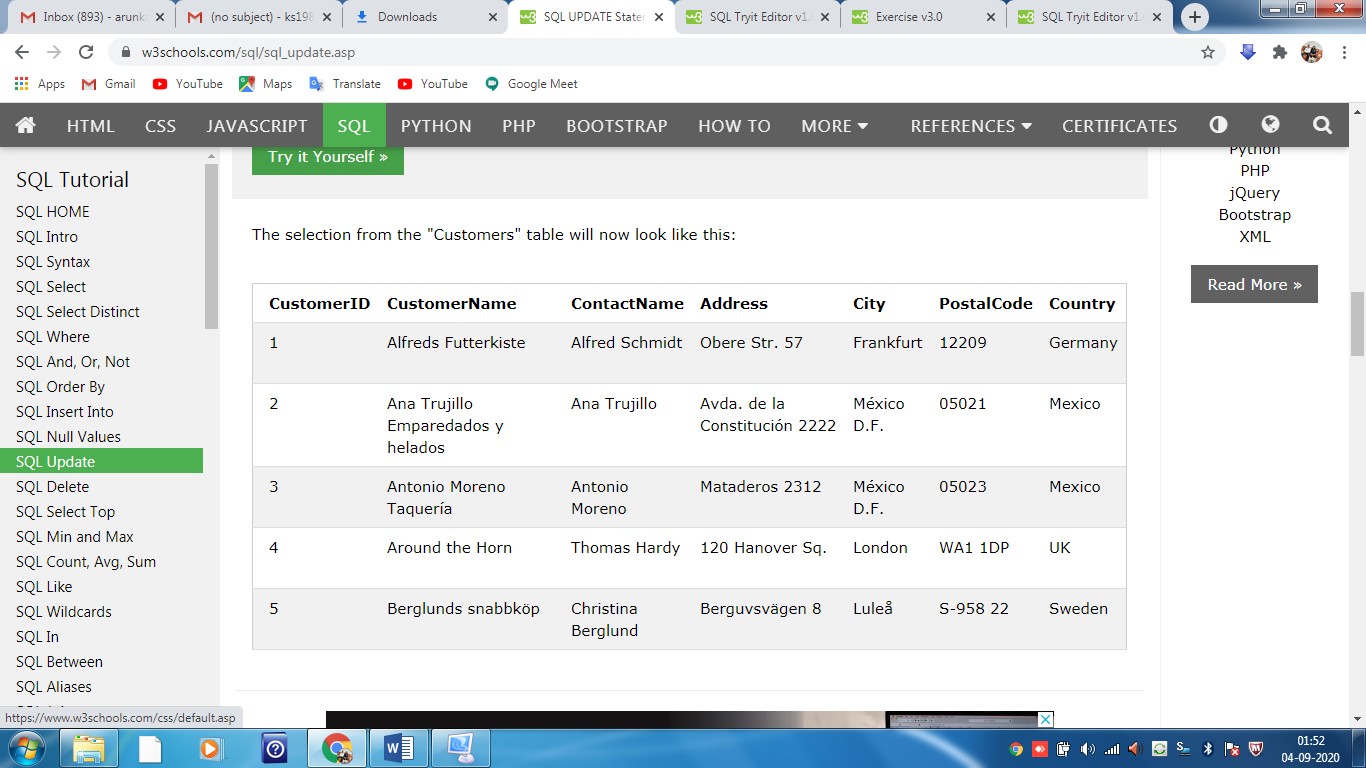
The UPDATE statement is used to modify the existing records in a table.

### UPDATE Syntax

UPDATE table\_name  
SET column1 = value1, column2 = value2, ...  
WHERE condition;

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

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

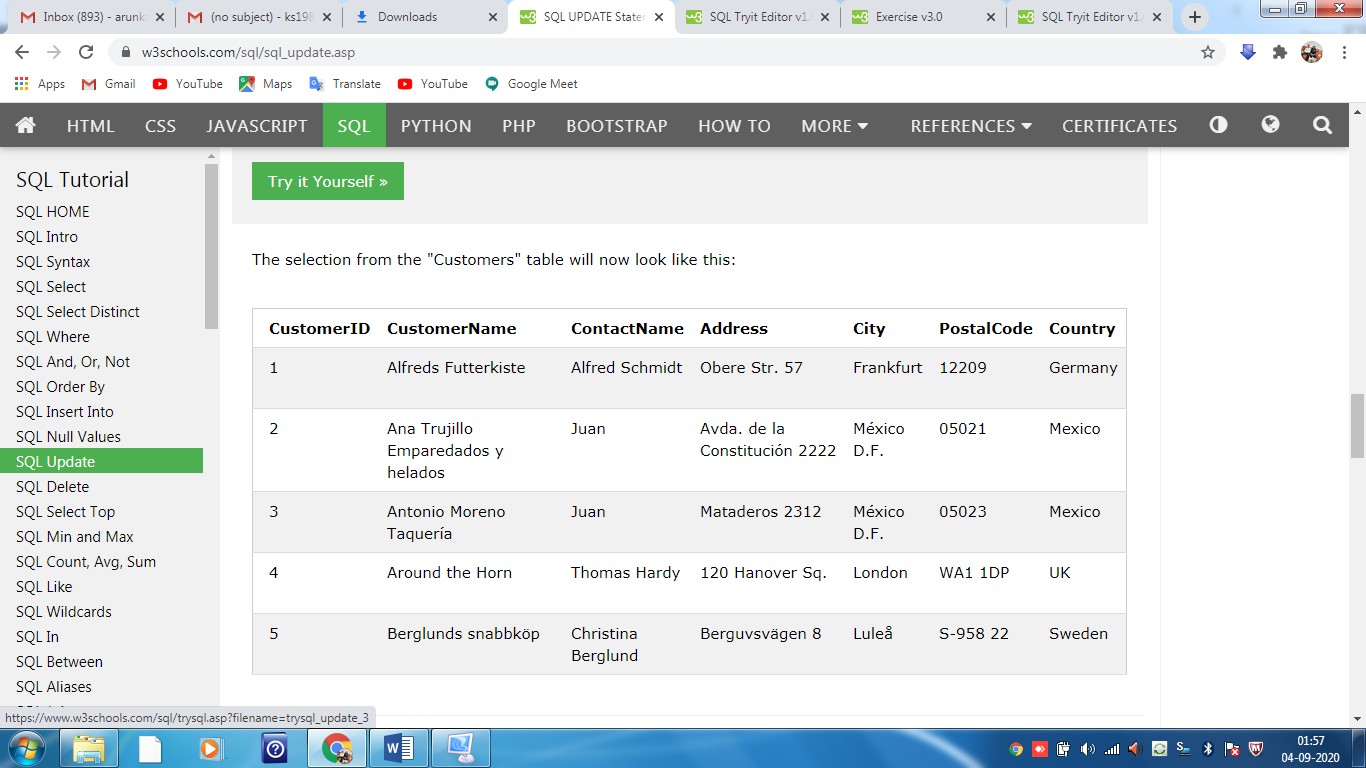


## UPDATE Multiple Records

It is the WHERE clause that determines how many records will be updated.

The following SQL statement will update the contact name to "Juan" for all records where country is "Mexico":

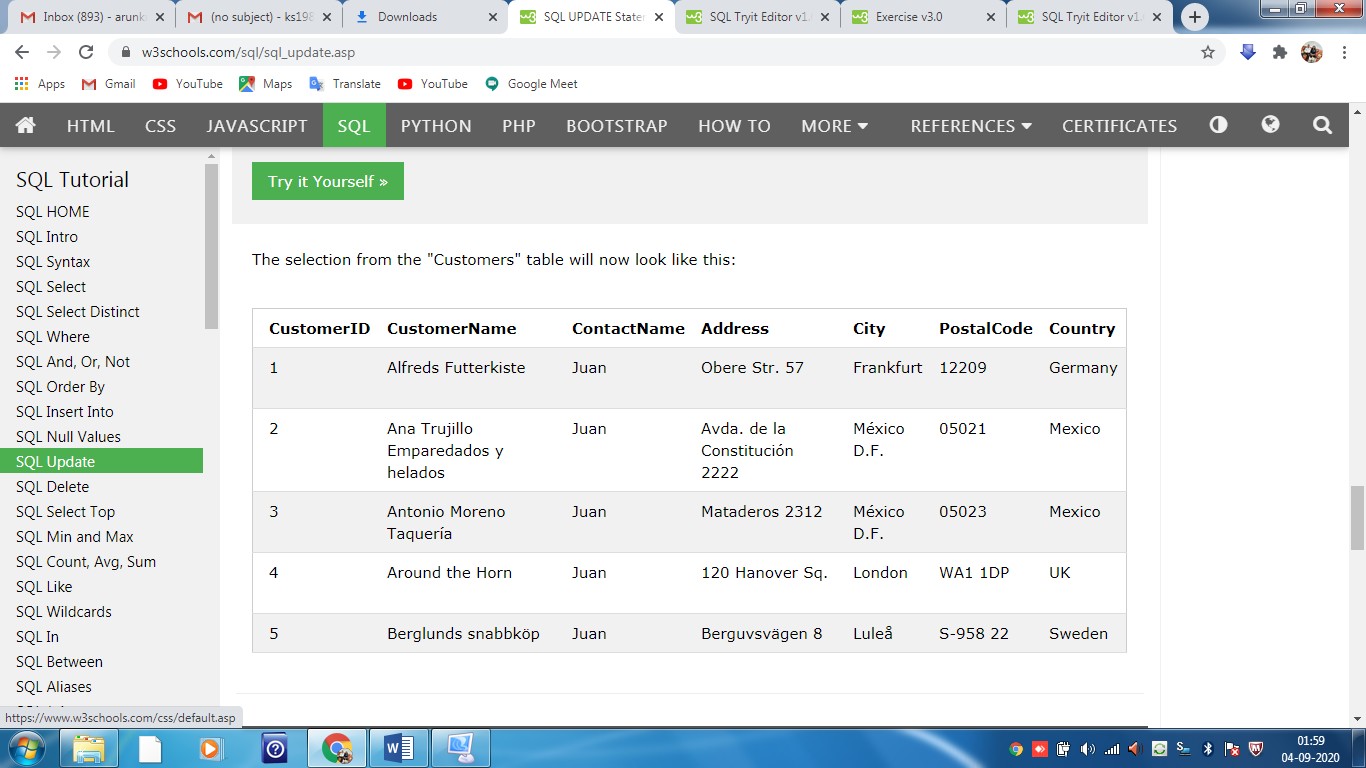
UPDATE Customers  
SET ContactName='Juan'  
WHERE Country='Mexico';



## Update Warning!

Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!

UPDATE Customers  
SET ContactName='Juan';



SQL DELETE Statement

## The SQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

### DELETE Syntax

DELETE FROM table\_name WHERE condition;

**Note:** 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!

## Example-SQL DELETE

## The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

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

## 

## Delete All Records

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

DELETE FROM *table\_name*;

The following SQL statement deletes all rows in the "Customers" table, without deleting the table:

### Example

DELETE FROM Customers;

SQL TOP, LIMIT or ROWNUM Clause

## The SQL SELECT TOP Clause

The SELECT TOP clause is used to specify the number of records to return.

The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

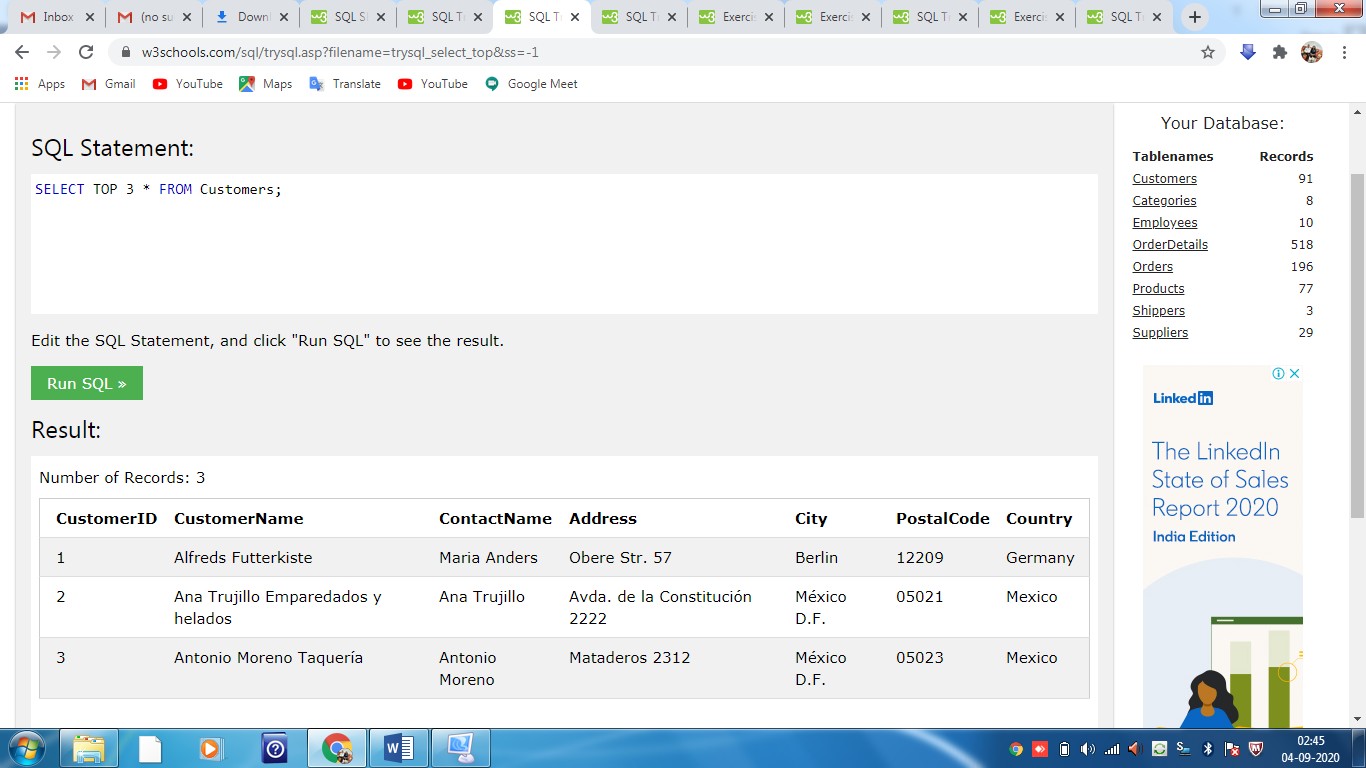
**Note:** Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses ROWNUM.

SELECT TOP number|*percent* column\_name(s)  
FROM table\_nameWHERE condition;

## SQL TOP, LIMIT and ROWNUM Examples

The following SQL statement selects the first three records from the "Customers" table (for SQL Server/MS Access):

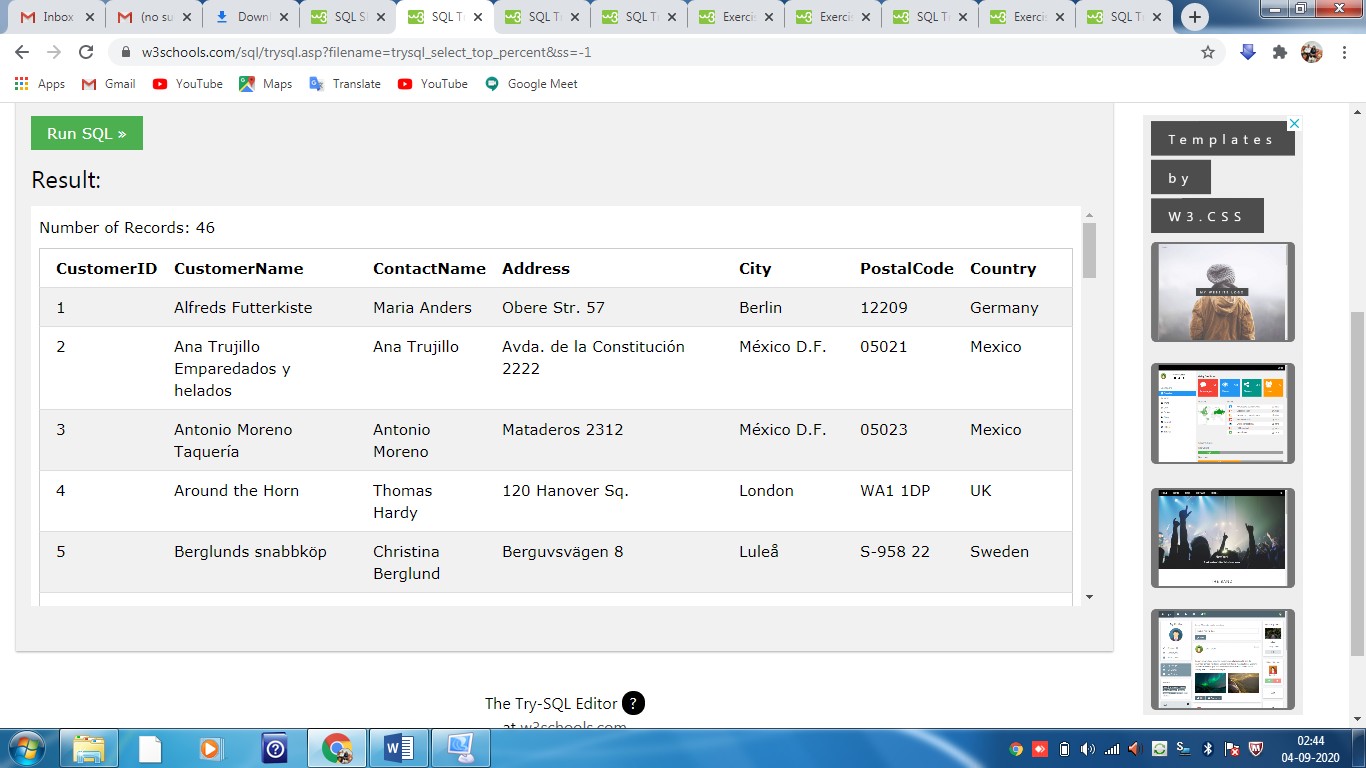
SELECT TOP 3 \* FROM Customers;



## SQL TOP PERCENT Example

The following SQL statement selects the first 50% of the records from the "Customers" table (for SQL Server/MS Access):

SELECT TOP 50 PERCENT \* FROM Customers;



## ADD a WHERE CLAUSE

The following SQL statement selects the first three records from the "Customers" table, where the country is "Germany" (for SQL Server/MS Access):

SELECT TOP 3 \* FROM Customers  
WHERE Country='Germany';

SQL MIN() and MAX() Functions

## The SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.

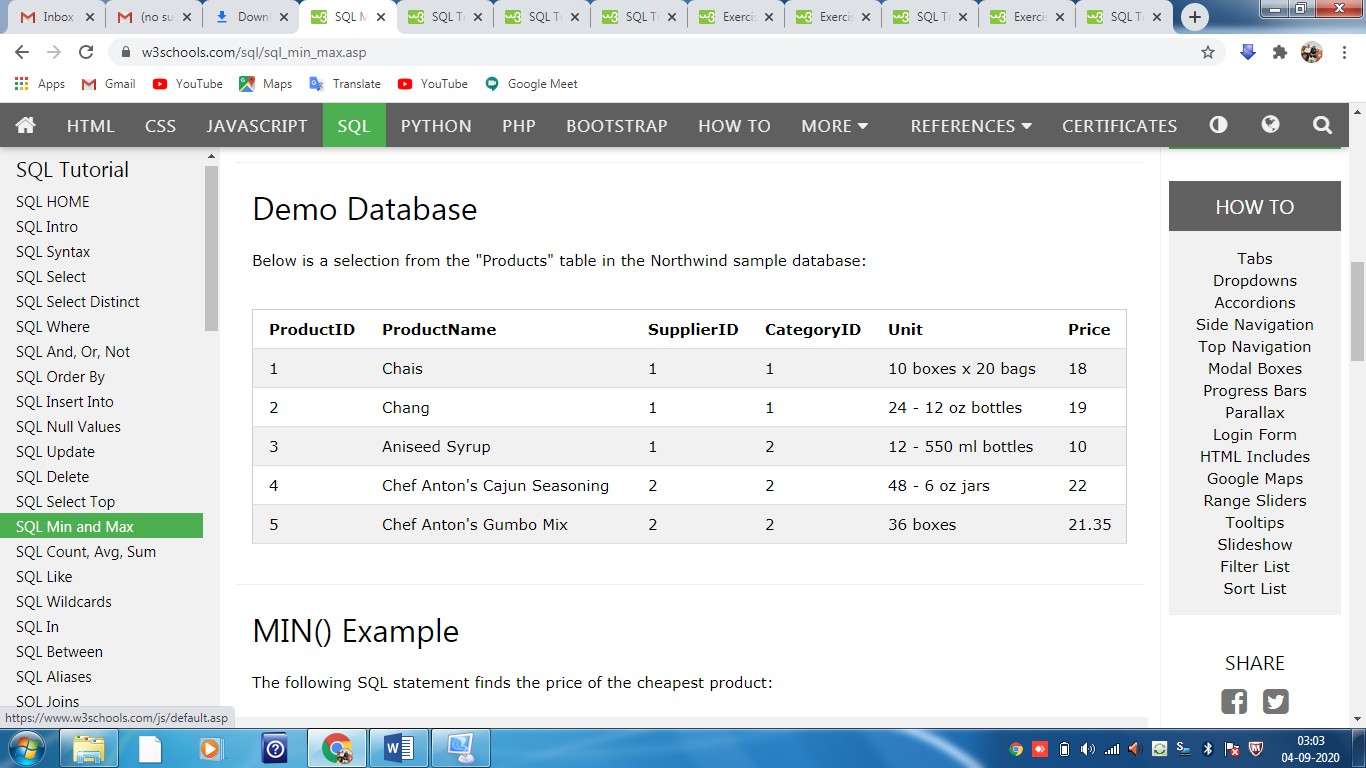
The MAX() function returns the largest value of the selected column.

### MIN() Syntax

SELECT MIN(column\_name)  
FROM table\_name  
WHERE condition;

### MAX() Syntax

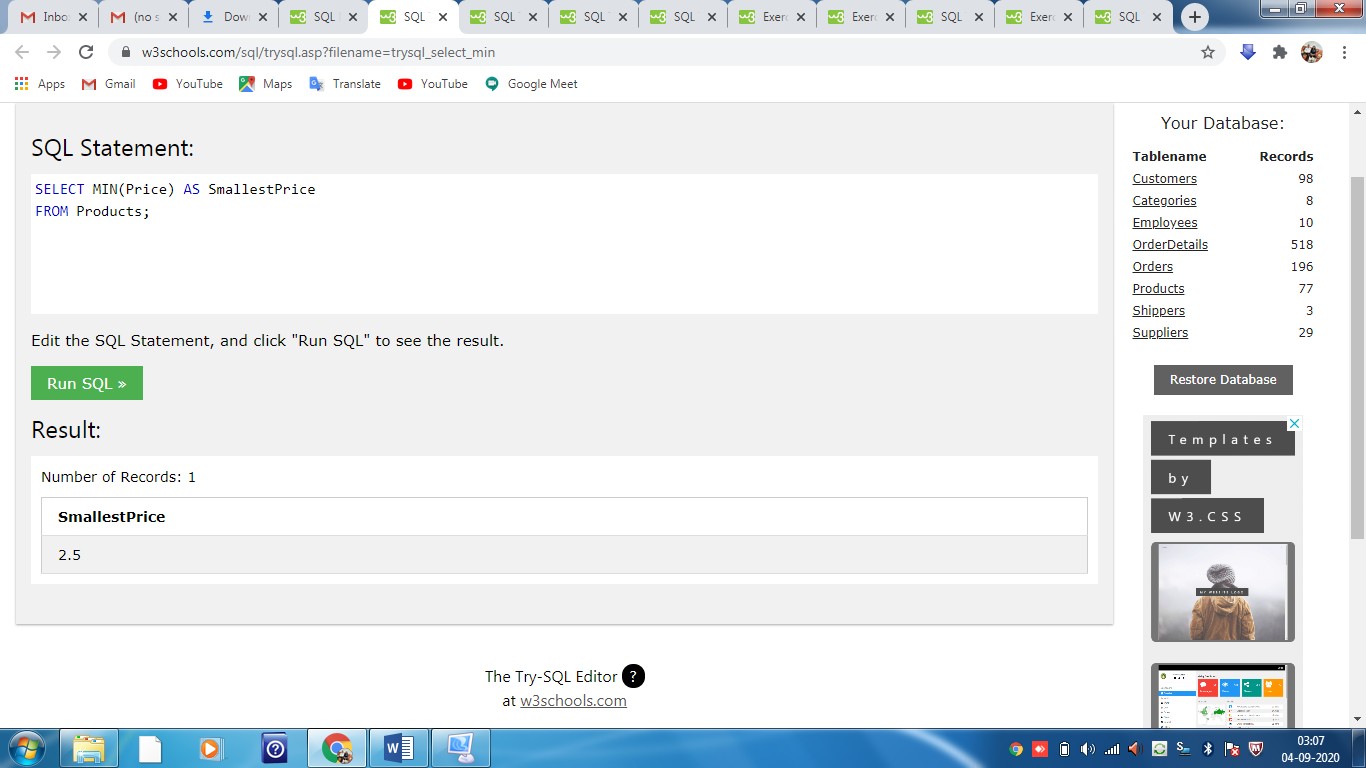
SELECT MAX(column\_name)  
FROM table\_name  
WHERE condition;



## MIN() Example

The following SQL statement finds the price of the cheapest product:

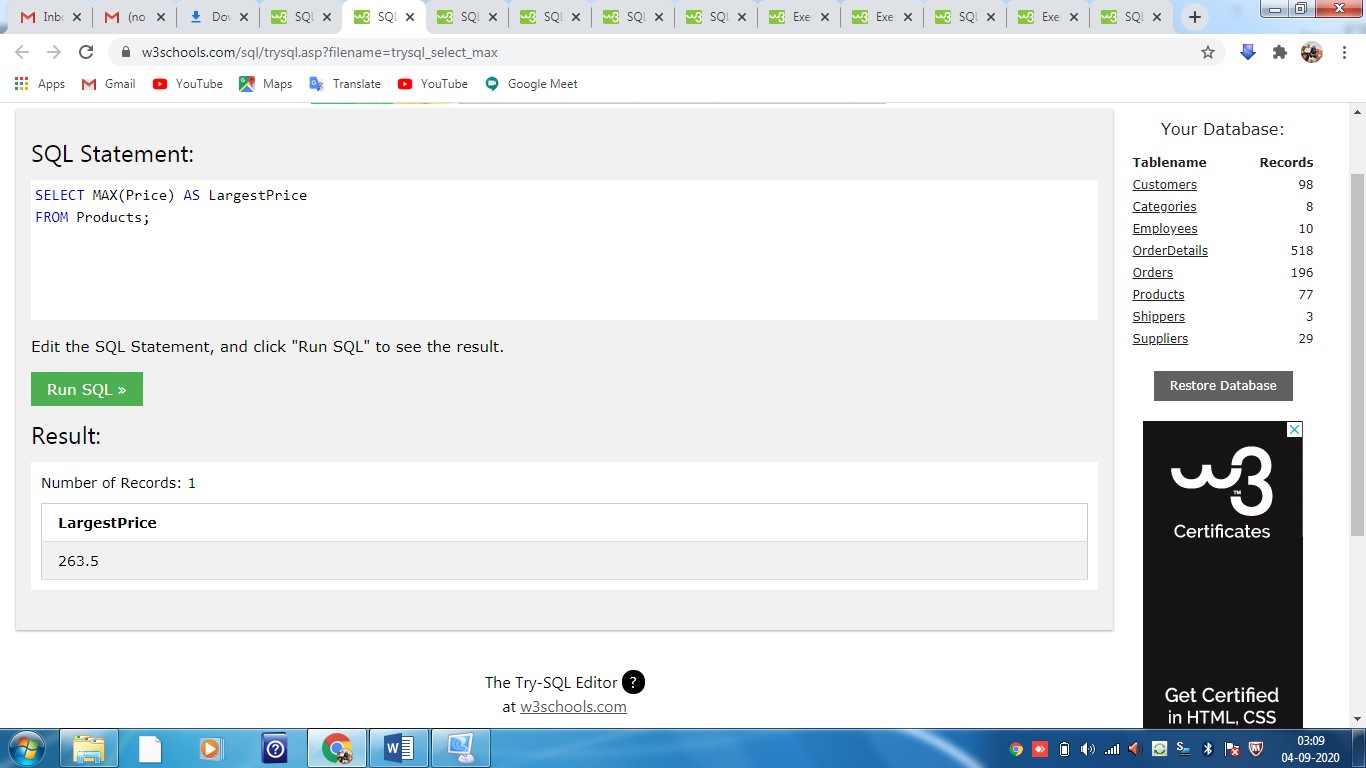
SELECT MIN(Price) AS SmallestPrice  
FROM Products;



## MAX() Example

The following SQL statement finds the price of the most expensive product:

SELECT MAX(Price) AS LargestPrice  
FROM Products;



SQL COUNT(), AVG() and SUM() Functions

## The SQL COUNT(), AVG() and SUM() Functions

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

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

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

### COUNT() Syntax

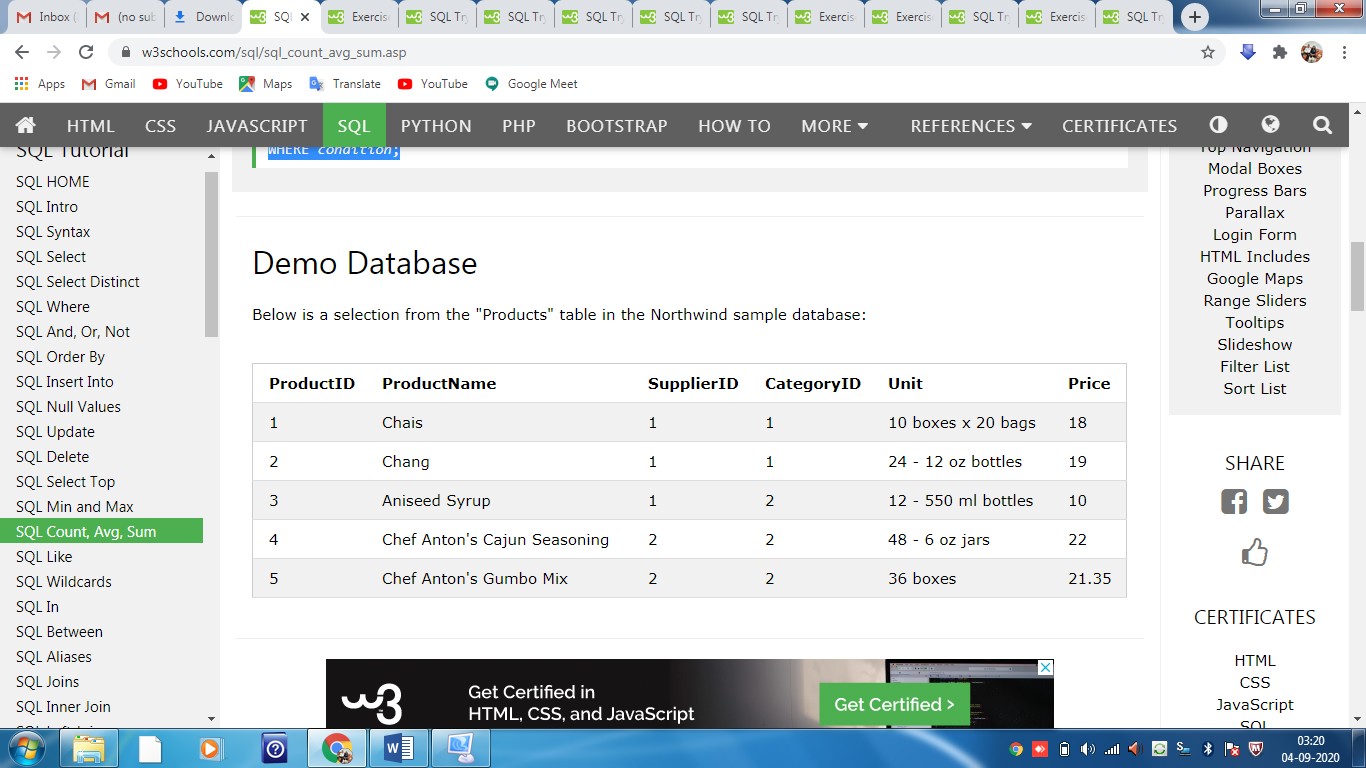
SELECT COUNT(column\_name)  
FROM table\_name  
WHERE condition;

### AVG() Syntax

SELECT AVG(column\_name)  
FROM table\_name  
WHERE condition;

### SUM() Syntax

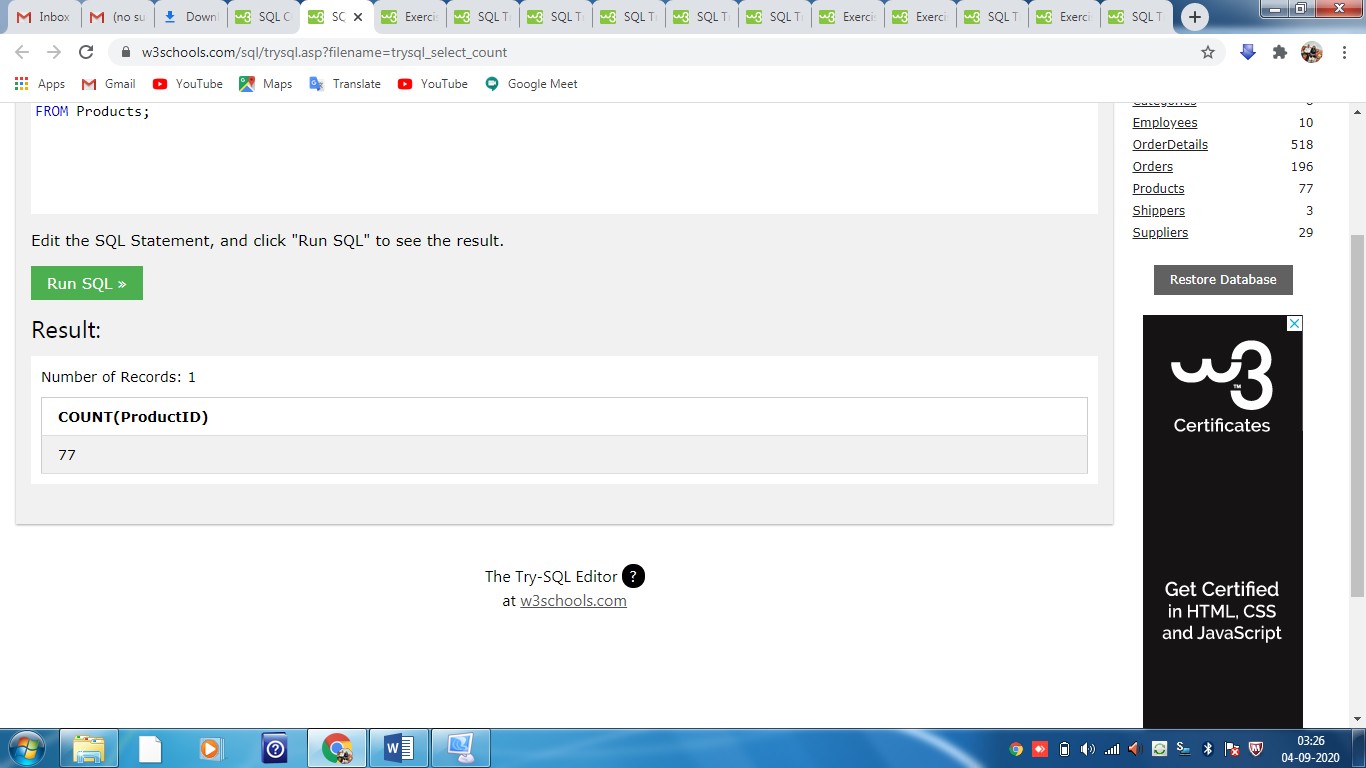
SELECT SUM(column\_name)  
FROM table\_name  
WHERE condition;

****

## COUNT() Example

The following SQL statement finds the number of products:

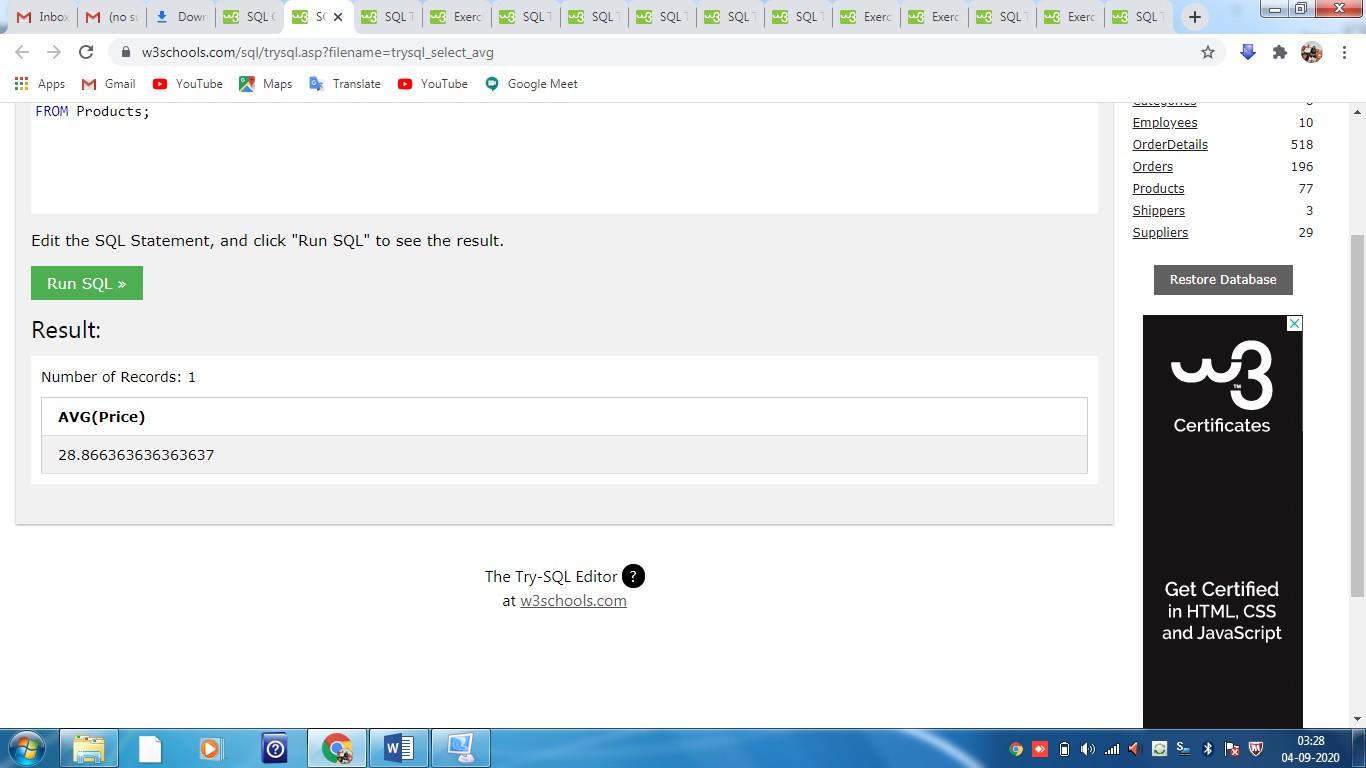
SELECT COUNT(ProductID)  
FROM Products;



## AVG() Example

The following SQL statement finds the average price of all products:

SELECT AVG(Price)  
FROM Products;



**Note:** NULL values are ignored.

## SUM() Example

The following SQL statement finds the sum of the "Quantity" fields in the "OrderDetails" table:

SELECT SUM(Quantity)  
FROM OrderDetails;

SQL LIKE Operator

The SQL LIKE Operator

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

There are two wildcards often used in conjunction with the LIKE operator:

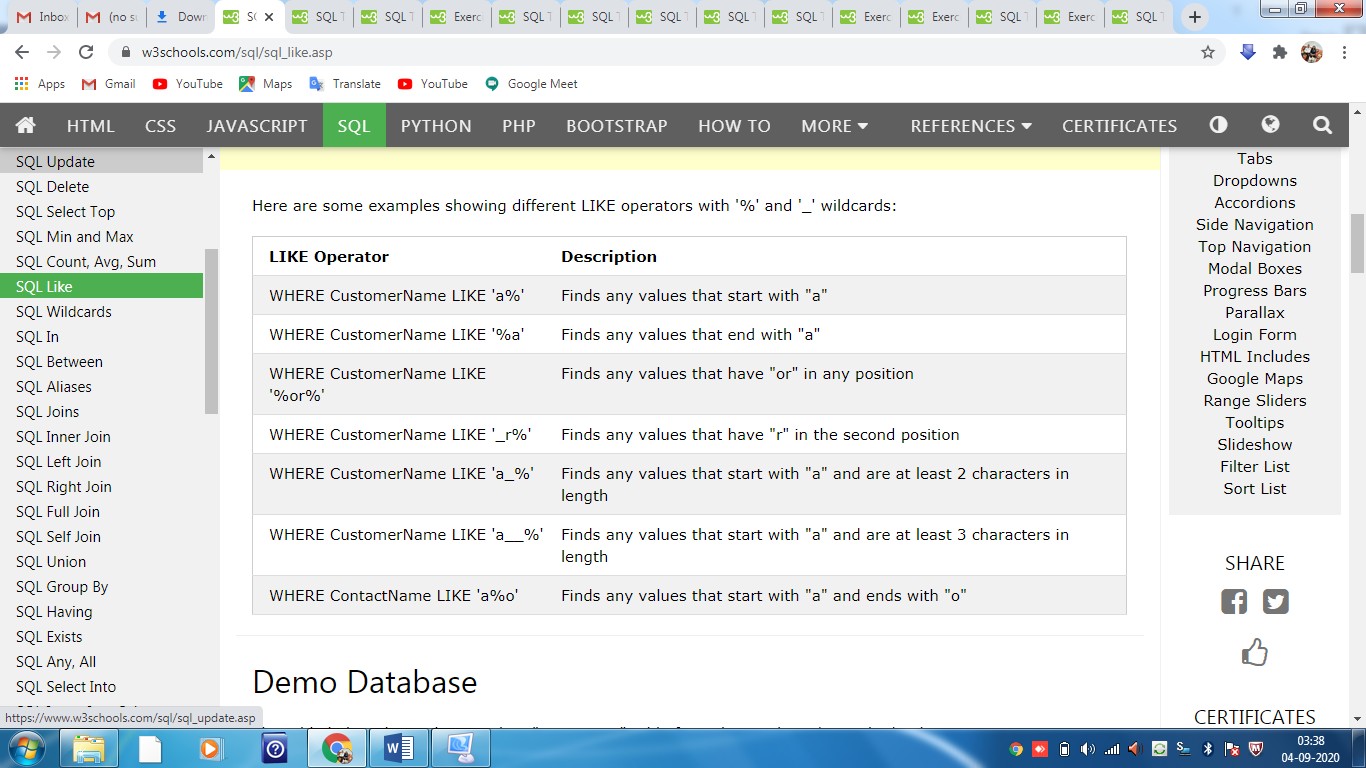
* % - The percent sign represents zero, one, or multiple characters
* \_ - The underscore represents a single character

**Note:** MS Access uses an asterisk (\*) instead of the percent sign (%), and a question mark (?) instead of the underscore (\_).

The percent sign and the underscore can also be used in combinations!

### LIKE Syntax

SELECT column1, column2, ...  
FROM table\_name  
WHERE columnN LIKE pattern;



## SQL LIKE Examples

The following SQL statement selects all customers with a CustomerName starting with "a":

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

The following SQL statement selects all customers with a CustomerName ending with "a":

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

The following SQL statement selects all customers with a CustomerName that have "or" in any position:

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

The following SQL statement selects all customers with a CustomerName that have "r" in the second position:

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

The following SQL statement selects all customers with a CustomerName that starts with "a" and are at least 3 characters in length:

SELECT \* FROM Customers  
WHERE CustomerName LIKE 'a\_\_%';

The following SQL statement selects all customers with a ContactName that starts with "a" and ends with "o":

SELECT \* FROM Customers  
WHERE ContactName LIKE 'a%o';

SQL IN Operator

## The SQL IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

### IN Syntax

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name IN (value1, value2, ...);

or:

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name IN (*SELECT* STATEMENT);

## IN Operator Examples

The following SQL statement selects all customers that are located in "Germany", "France" or "UK":

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



SQL BETWEEN Operator

## The SQL BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

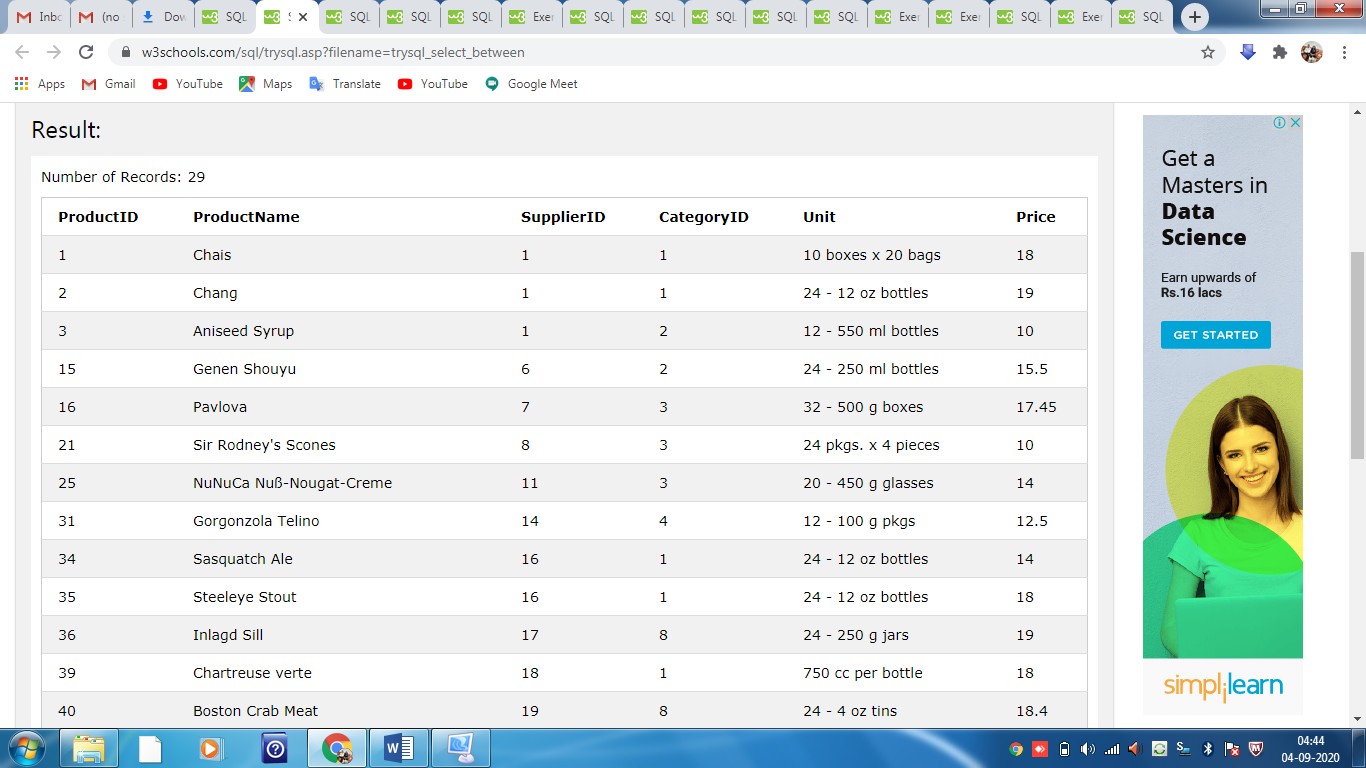
### BETWEEN Syntax

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name BETWEEN value1 AND value2;

## BETWEEN Example

The following SQL statement selects all products with a price BETWEEN 10 and 20:

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



SQL Joins

## SQL JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Let's look at a selection from the "Orders" table:

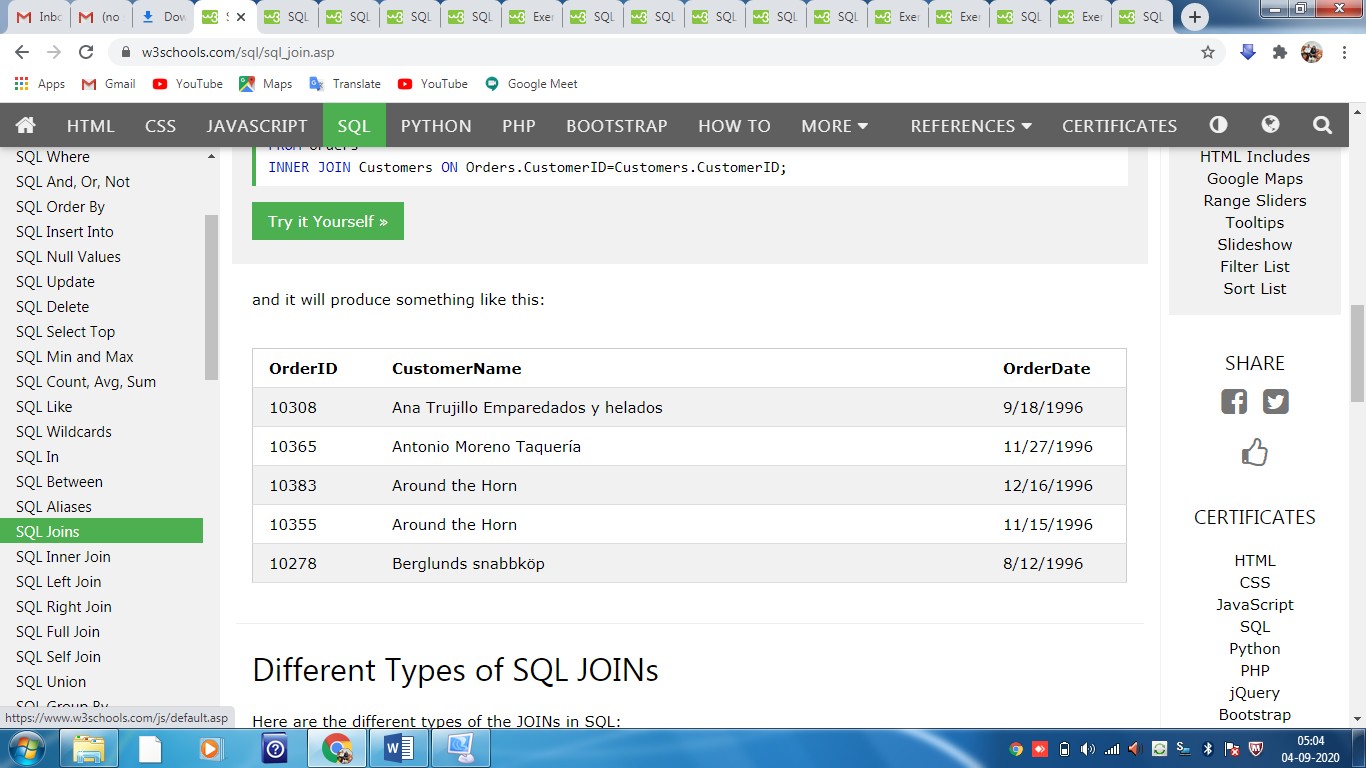
|  |  |  |
| --- | --- | --- |
| **OrderID** | **CustomerID** | **OrderDate** |
| 10308 | 2 | 1996-09-18 |
| 10309 | 37 | 1996-09-19 |
| 10310 | 77 | 1996-09-20 |

Then, look at a selection from the "Customers" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mexico |

Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

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



Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

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

SQL INNER JOIN Keyword

## SQL INNER JOIN Keyword

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

### INNER JOIN Syntax

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2ON table1.column\_name = table2.column\_name;



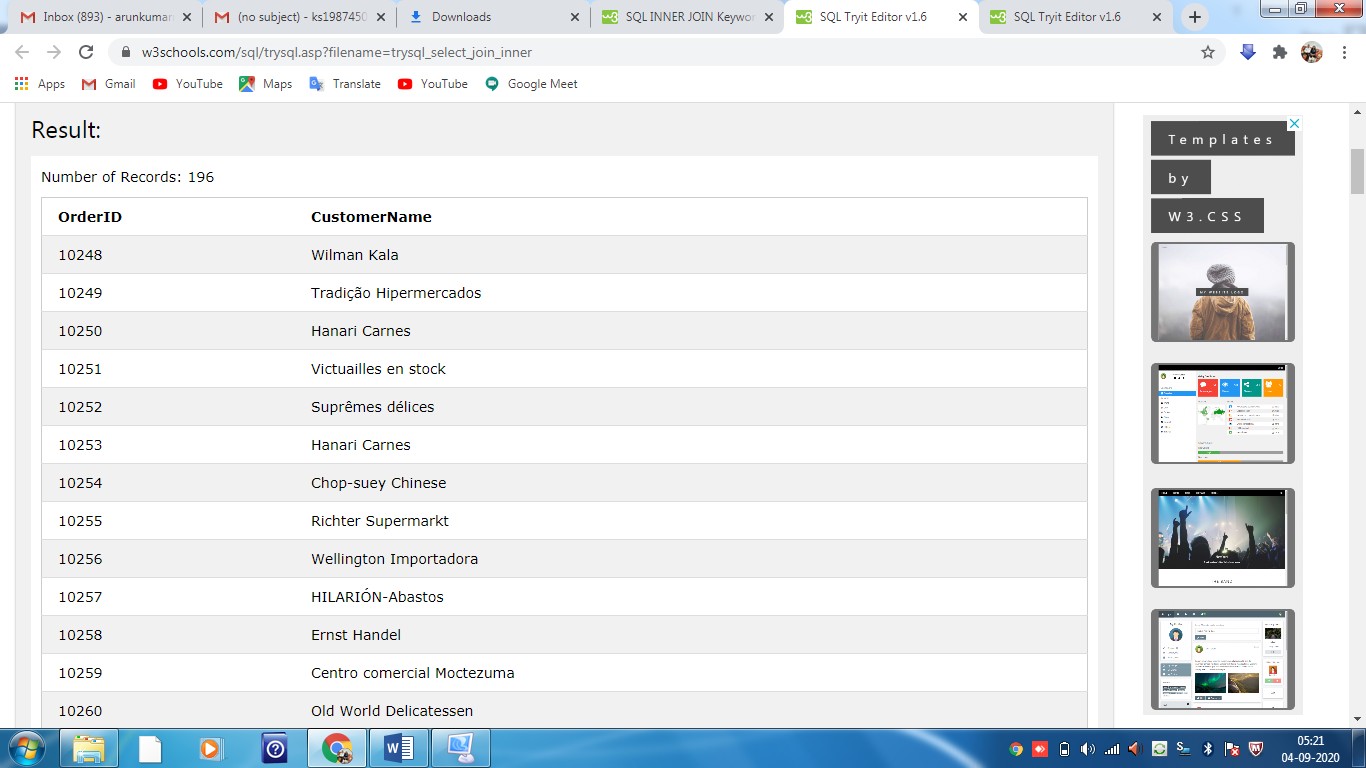
## SQL INNER JOIN Example

## 

## The following SQL statement selects all orders with customer information:

SELECT Orders.OrderID, Customers.CustomerName  
FROM Orders  
INNER JOIN Customers

ON Orders.CustomerID = Customers.CustomerID;



## JOIN Three Tables

The following SQL statement selects all orders with customer and shipper information:

SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName  
FROM Orders  
INNER JOIN Customers

ON Orders.CustomerID = Customers.CustomerID)  
INNER JOIN Shippers

ON Orders.ShipperID = Shippers.ShipperID);



SQL LEFT JOIN Keyword

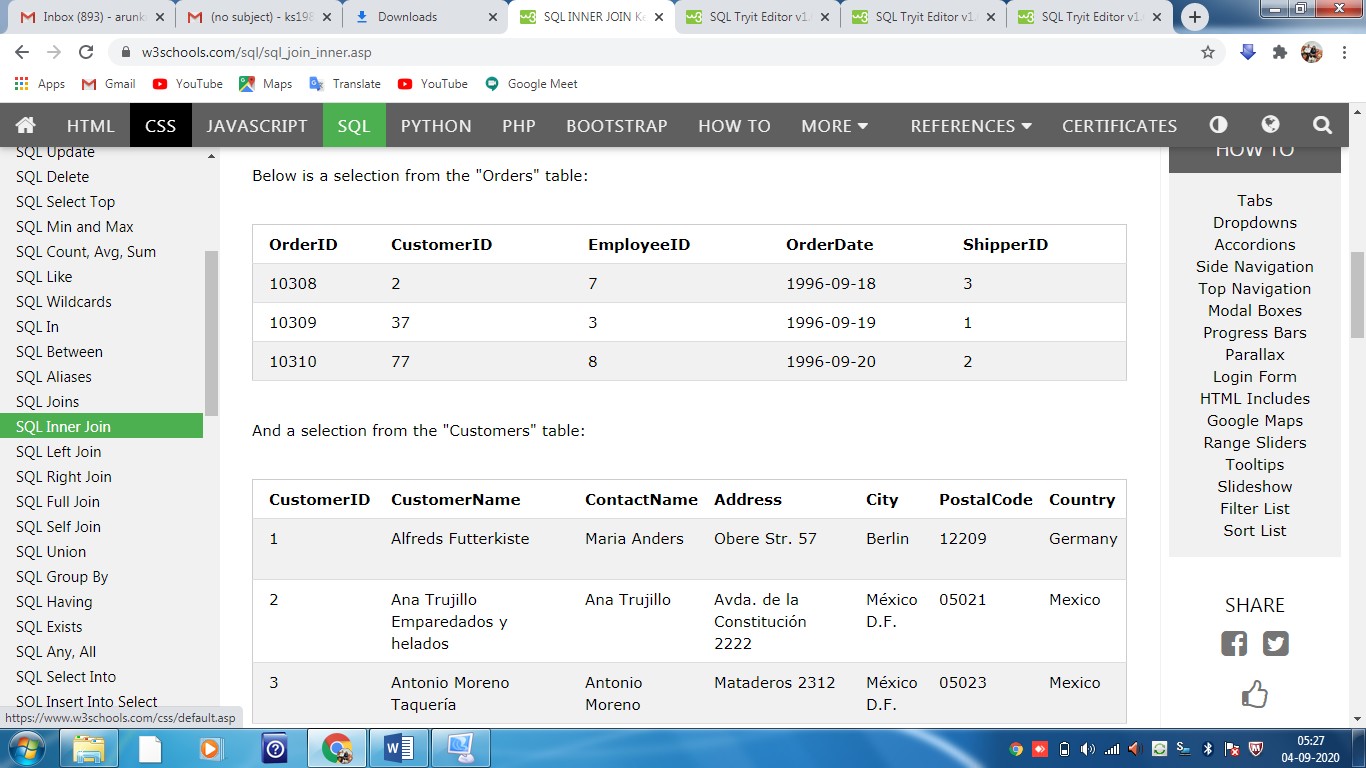
## SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

### LEFT JOIN Syntax

SELECT column\_name(s)  
FROM table1  
LEFT JOIN table2ON table1.column\_name = table2.column\_name;



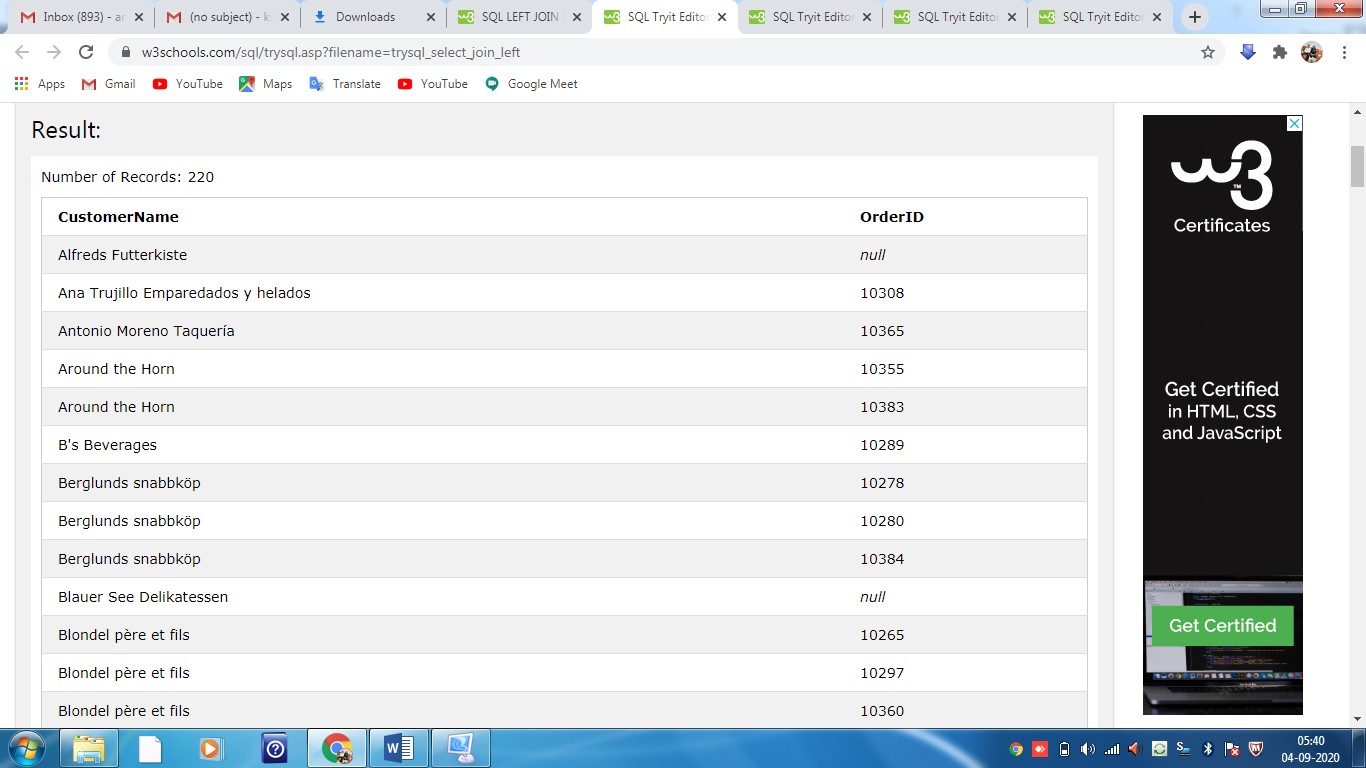
****

## SQL LEFT JOIN Example

The following SQL statement will select all customers, and any orders they might have:

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
LEFT JOIN Orders

ON Customers.CustomerID = Orders.CustomerID  
ORDER BY Customers.CustomerName;



SQL RIGHT JOIN Keyword

## SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

### RIGHT JOIN Syntax

SELECT column\_name(s)  
FROM table1  
RIGHT JOIN table2ON table1.column\_name = table2.column\_name;

**Note:** In some databases RIGHT JOIN is called RIGHT OUTER JOIN.



SQL FULL OUTER JOIN Keyword

## SQL FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

**Note:** FULL OUTER JOIN can potentially return very large result-sets!

**Tip:** FULL OUTER JOIN and FULL JOIN are the same.

### FULL OUTER JOIN Syntax

SELECT column\_name(s)  
FROM table1  
FULL OUTER JOIN table2ON table1.column\_name = table2.column\_nameWHERE condition;



## SQL FULL OUTER JOIN Example

The following SQL statement selects all customers, and all orders:

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID  
ORDER BY Customers.CustomerName;

SQL GROUP BY Statement

## The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

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

### GROUP BY Syntax

SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)ORDER BY column\_name(s);