**SQL AGGREGATE FUNCTIONS**

An aggregate function is a function that performs a calculation on a set of values, and returns a single value.

Aggregate functions are often used with the GROUP BY clause of the SELECT statement. The GROUP BY clause splits the result-set into groups of values and the aggregate function can be used to return a single value for each group.

The most commonly used SQL aggregate functions are:

* MIN() - returns the smallest value within the selected column
* MAX() - returns the largest value within the selected column
* COUNT() - returns the number of rows in a set
* SUM() - returns the total sum of a numerical column
* AVG() - returns the average value of a numerical column

Aggregate functions ignore null values (except for COUNT(\*)).

# **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 Example**[**Get your own SQL Server**](https://www.w3schools.com/sql/sql_server.asp)

Find the lowest price in the Price column:

SELECT MIN(Price)  
FROM Products;

### **MAX Example**

Find the highest price in the Price column:

SELECT MAX(Price)  
FROM Products;

## Syntax

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

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

## Demo Database

Below is a selection from the [**Products**](https://www.w3schools.com/sql/trysql.asp?filename=trysql_products) table used in the examples:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 22 |
| 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 21.35 |

## Set Column Name (Alias)

When you use MIN() or MAX(), the returned column will not have a descriptive name. To give the column a descriptive name, use the AS keyword:

### **Example**

SELECT MIN(Price) AS SmallestPrice  
FROM Products;

## Use MIN() with GROUP BY

Here we use the MIN() function and the GROUP BY clause, to return the smallest price for each category in the Products table:

### **Example**

SELECT MIN(Price) AS SmallestPrice, CategoryID  
FROM Products  
GROUP BY CategoryID;

# **SQL COUNT() Function**

## The SQL COUNT() Function

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

### **Example**[**Get your own SQL Server**](https://www.w3schools.com/sql/sql_server.asp)

Find the total number of rows in the Products table:

SELECT COUNT(\*)  
FROM Products;

## Syntax

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

## Demo Database

Below is a selection from the [**Products**](https://www.w3schools.com/sql/trysql.asp?filename=trysql_products) table used in the examples:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 22 |
| 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 21.35 |

## Specify Column

You can specify a column name instead of the asterix symbol (\*).

If you specify a column name instead of (\*), NULL values will not be counted.

### **Example**

Find the number of products where the ProductName is not null:

SELECT COUNT(ProductName)  
FROM Products;

## Add a WHERE Clause

You can add a WHERE clause to specify conditions:

### **Example**

Find the number of products where Price is higher than 20:

SELECT COUNT(ProductID)  
FROM Products  
WHERE Price > 20;

## Ignore Duplicates

You can ignore duplicates by using the DISTINCT keyword in the COUNT() function.

If DISTINCT is specified, rows with the same value for the specified column will be counted as one.

### **Example**

How many different prices are there in the Products table:

SELECT COUNT(DISTINCT Price)  
FROM Products;

## Use an Alias

Give the counted column a name by using the AS keyword.

### **Example**

Name the column "Number of records":

SELECT COUNT(\*) AS [Number of records]  
FROM Products;

## Use COUNT() with GROUP BY

Here we use the COUNT() function and the GROUP BY clause, to return the number of records for each category in the Products table:

### **Example**

SELECT COUNT(\*) AS [Number of records], CategoryID  
FROM Products  
GROUP BY CategoryID;

# **SQL SUM() Function**

## The SQL SUM() Function

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

### **Example**[**Get your own SQL Server**](https://www.w3schools.com/sql/sql_server.asp)

Return the sum of all Quantity fields in the OrderDetails table:

SELECT SUM(Quantity)  
FROM OrderDetails;

## Syntax

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

## Demo Database

Below is a selection from the **[OrderDetails](https://www.w3schools.com/sql/trysql.asp?filename=trysql_orderdetails" \t "_blank)** table used in the examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **OrderDetailID** | **OrderID** | **ProductID** | **Quantity** |
| 1 | 10248 | 11 | 12 |
| 2 | 10248 | 42 | 10 |
| 3 | 10248 | 72 | 5 |
| 4 | 10249 | 14 | 9 |
| 5 | 10249 | 51 | 40 |

## Add a WHERE Clause

You can add a WHERE clause to specify conditions:

### **Example**

Return the sum of the Quantity field for the product with ProductID 11:

SELECT SUM(Quantity)  
FROM OrderDetails  
WHERE ProductId = 11;

## Use an Alias

Give the summarized column a name by using the AS keyword.

### **Example**

Name the column "total":

SELECT SUM(Quantity) AS total  
FROM OrderDetails;

## Use SUM() with GROUP BY

Here we use the SUM() function and the GROUP BY clause, to return the Quantity for each OrderID in the OrderDetails table:

### **Example**

SELECT OrderID, SUM(Quantity) AS [Total Quantity]  
FROM OrderDetails  
GROUP BY OrderID;

You will learn more about the [GROUP BY](https://www.w3schools.com/sql/sql_groupby.asp) clause later in this tutorial.

## SUM() With an Expression

The parameter inside the SUM() function can also be an expression.

If we assume that each product in the OrderDetails column costs 10 dollars, we can find the total earnings in dollars by multiply each quantity with 10:

### **Example**

SELECT SUM(Quantity \* 10)  
FROM OrderDetails; Use an expression inside the SUM() function:

We can also join the OrderDetails table to the Products table to find the actual amount, instead of assuming it is 10 dollars:

### **Example**

SELECT SUM(Price \* Quantity) Join OrderDetails with Products, and use SUM() to find the total amount:

FROM OrderDetails  
LEFT JOIN Products ON OrderDetails.ProductID = Products.ProductID;

You will learn more about [Joins](https://www.w3schools.com/sql/sql_join.asp) later in this tutorial.

# **SQL AVG() Function**

## The SQL AVG() Function

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

### **Example**[**Get your own SQL Server**](https://www.w3schools.com/sql/sql_server.asp)

Find the average price of all products:

SELECT AVG(Price)  
FROM Products;

**Note:** NULL values are ignored.

## Syntax

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

## Demo Database

Below is a selection from the [**Products**](https://www.w3schools.com/sql/trysql.asp?filename=trysql_products) table used in the examples:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 22 |
| 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 21.35 |

## Add a WHERE Clause

You can add a WHERE clause to specify conditions:

### **Example**

Return the average price of products in category 1:

SELECT AVG(Price)  
FROM Products  
WHERE CategoryID = 1;

## Use an Alias

Give the AVG column a name by using the AS keyword.

### **Example**

Name the column "average price":

SELECT AVG(Price) AS [average price]  
FROM Products;

## Higher Than Average

To list all records with a higher price than average, we can use the AVG() function in a sub query:

### **Example**

Return all products with a higher price than the average price:

SELECT \* FROM Products  
WHERE price > (SELECT AVG(price) FROM Products);

## Use AVG() with GROUP BY

Here we use the AVG() function and the GROUP BY clause, to return the average price for each category in the Products table:

### **Example**

SELECT AVG(Price) AS AveragePrice, CategoryID  
FROM Products  
GROUP BY CategoryID;