**SQL**

SQL is a standard language for storing, manipulating and retrieving data in databases.

# **Introduction to SQL**

SQL is a standard language for accessing and manipulating databases.

## What is SQL?

* SQL stands for Structured Query Language
* SQL lets you access and manipulate databases
* SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

## What Can SQL do?

* SQL can execute queries against a database
* SQL can retrieve data from a database
* SQL can insert records in a database
* SQL can update records in a database
* SQL can delete records from a database
* SQL can create new databases
* SQL can create new tables in a database
* SQL can create stored procedures in a database
* SQL can create views in a database
* SQL can set permissions on tables, procedures, and views

## SQL is a Standard - BUT....

Although SQL is an ANSI/ISO standard, there are different versions of the SQL language.

However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.

**Note:** Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard!

## Using SQL in Your Web Site

To build a web site that shows data from a database, you will need:

* An RDBMS database program (i.e. MS Access, SQL Server, MySQL)
* To use a server-side scripting language, like PHP or ASP
* To use SQL to get the data you want
* To use HTML / CSS to style the page

**ExampleTop of Form**

### SQL Statement:

SELECT \* FROM Customers;

### Result:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 8 | Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |
| 9 | Bon app' | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |

## RDBMS

RDBMS stands for Relational Database Management System.

RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

Look at the "Customers" table:

### Example

SELECT \* FROM Customers;

Every table is broken up into smaller entities called fields. The fields in the Customers table consist of CustomerID, CustomerName, ContactName, Address, City, PostalCode and Country. A field is a column in a table that is designed to maintain specific information about every record in the table.

A record, also called a row, is each individual entry that exists in a table. For example, there are 91 records in the above Customers table. A record is a horizontal entity in a table.

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

## Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"), and contain records (rows) with data.

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

The table above contains five records (one for each customer) and seven columns (CustomerID, CustomerName, ContactName, Address, City, PostalCode, and Country).

* SQL keywords are NOT case sensitive: select is the same as SELECT

## Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement.

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

## Some of The Most Important SQL Commands

* **SELECT** - extracts data from a database
* **UPDATE** - updates data in a database
* **DELET**E - 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.

### Example

Return data from the Customers table:

### SQL Statement:

SELECT CustomerName,City FROM Customers;

|  |  |
| --- | --- |
| **CustomerName** | **City** |
| Alfreds Futterkiste | Berlin |
| Ana Trujillo Emparedados y helados | México D.F. |
| Antonio Moreno Taquería | México D.F. |
| Around the Horn | London |
| Berglunds snabbköp | Luleå |
| Blauer See Delikatessen | Mannheim |
| Blondel père et fils | Strasbourg |
| Bólido Comidas preparadas | Madrid |
| Bon app' | Marseille |
| Bottom-Dollar Marketse | Tsawassen |
| B's Beverages | London |
| Cactus Comidas para llevar | Buenos Aires |
| Centro comercial Moctezuma | México D.F. |
| Chop-suey Chinese | Bern |
| Comércio Mineiro | São Paulo |

### Result:

Number of Records: 14

## Syntax

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

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

The table\_name represents the name of the table you want to select data from.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

## Demo Database

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

## Select ALL columns

If you want to return all columns, without specifying every column name, you can use the SELECT \* syntax:

### Example

Return all the columns from the Customers table:

### SQL Statement:

SELECT \* FROM Customers;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 8 | Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |
| 9 | Bon app' | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |
| 10 | Bottom-Dollar Marketse | Elizabeth Lincoln | 23 Tsawassen Blvd. | Tsawassen | T2F 8M4 | Canada |

Number of Records: 10

## ​The SQL SELECT DISTINCT Statement

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

### Example

Select all the different countries from the "Customers" table:

### SQL Statement:

SELECT DISTINCT Country FROM Customers;

### Result:

|  |
| --- |
| **Country** |
| Argentina |
| Austria |
| Belgium |
| Brazil |
| Canada |
| Denmark |
| Finland |
| France |
| Germany |
| Ireland |

Number of Records: 10

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

## Syntax

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

## Demo Database

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

## SELECT Example Without DISTINCT

If you omit the DISTINCT keyword, the SQL statement returns the **"**Country" value from all the records of the "Customers" table**:**

### SQL Statement:

SELECT Country FROM Customers;

### ​Result:

Number of Records: 14

|  |
| --- |
| **Country** |
| Germany |
| Mexico |
| Mexico |
| UK |
| Sweden |
| Germany |
| France |
| Spain |
| France |
| Canada |
| UK |
| Argentina |
| Mexico |
| Switzerland |

## Count Distinct

By using the DISTINCT keyword in a function called COUNT, we can return the number of different countries.

### Example

SELECT COUNT(DISTINCT Country) FROM Customers;

**Note: The**COUNT(DISTINCT column\_name) is not supported in Microsoft Access databases.

### Example

SELECT Count(\*) AS DistinctCountries  
FROM (SELECT DISTINCT Country FROM Customers);

## The SQL WHERE Clause

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

### Example

Select all customers from Mexico:

### SQL Statement:

SELECT \* FROM Customers

WHERE Country='Mexico';

Number of Records: 5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 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 |
| 13 | Centro comercial Moctezuma | Francisco Chang | Sierras de Granada 9993 | México D.F. | 05022 | Mexico |
| 58 | Pericles Comidas clásicas | Guillermo Fernández | Calle Dr. Jorge Cash 321 | México D.F. | 05033 | Mexico |
| 80 | Tortuga Restaurante | Miguel Angel Paolino | Avda. Azteca 123 | México D.F. | 05033 | Mexico |

## Syntax

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

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

**Note:** The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc.!

## Demo Database

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

## Text Fields vs. Numeric Fields

SQL requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes:

### Example

### SQL Statement:

SELECT \* FROM Customers

WHERE CustomerID=1;

### Result:

Number of Records: 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |

## Operators in The WHERE Clause

You can use other operators than the = operator to filter the search.

### Example

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 81 | Tradição Hipermercados | Anabela Domingues | Av. Inês de Castro, 414 | São Paulo | 05634-030 | Brazil |
| 82 | Trail's Head Gourmet Provisioners | Helvetius Nagy | 722 DaVinci Blvd. | Kirkland | 98034 | USA |
| 83 | Vaffeljernet | Palle Ibsen | Smagsløget 45 | Århus | 8200 | Denmark |
| 84 | Victuailles en stock | Mary Saveley | 2, rue du Commerce | Lyon | 69004 | France |
| 85 | Vins et alcools Chevalier | Paul Henriot | 59 rue de l'Abbaye | Reims | 51100 | France |
| 86 | Die Wandernde Kuh | Rita Müller | Adenauerallee 900 | Stuttgart | 70563 | Germany |
| 87 | Wartian Herkku | Pirkko Koskitalo | Torikatu 38 | Oulu | 90110 | Finland |
| 88 | Wellington Importadora | Paula Parente | Rua do Mercado, 12 | Resende | 08737-363 | Brazil |
| 89 | White Clover Markets | Karl Jablonski | 305 - 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |

Select all customers with a CustomerID greater than 80:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 8 | Northwoods Cranberry Sauce | 3 | 2 | 12 - 12 oz jars | 40 |
| 9 | Mishi Kobe Niku | 4 | 6 | 18 - 500 g pkgs. | 97 |
| 10 | Ikura | 4 | 8 | 12 - 200 ml jars | 31 |
| 12 | Queso Manchego La Pastora | 5 | 4 | 10 - 500 g pkgs. | 38 |
| 17 | Alice Mutton | 7 | 6 | 20 - 1 kg tins | 39 |
| 18 | Carnarvon Tigers | 7 | 8 | 16 kg pkg. | 62.5 |
| 20 | Sir Rodney's Marmalade | 8 | 3 | 30 gift boxes | 81 |
| 26 | Gumbär Gummibärchen | 11 | 3 | 100 - 250 g bags | 31.23 |
| 27 | Schoggi Schokolade | 11 | 3 | 100 - 100 g pieces | 43.9 |
| 28 | Rössle Sauerkraut | 12 | 7 | 25 - 825 g cans | 45.6 |
| 29 | Thüringer Rostbratwurst | 12 | 6 | 50 bags x 30 sausgs. | 123.79 |
| 32 | Mascarpone Fabioli | 14 | 4 | 24 - 200 g pkgs. | 32 |
| 38 | Côte de Blaye | 18 | 1 | 12 - 75 cl bottles | 263.5 |
| 43 | Ipoh Coffee | 20 | 1 | 16 - 500 g tins | 46 |
| 51 | Manjimup Dried Apples | 24 | 7 | 50 - 300 g pkgs. | 53 |
| 53 | Perth Pasties | 24 | 6 | 48 pieces | 32.8 |
| 56 | Gnocchi di nonna Alice | 26 | 5 | 24 - 250 g pkgs. | 38 |
| 59 | Raclette Courdavault | 28 | 4 | 5 kg pkg. | 55 |
| 60 | Camembert Pierrot | 28 | 4 | 15 - 300 g rounds | 34 |
| 62 | Tarte au sucre | 29 | 3 | 48 pies | 49.3 |
| 63 | Vegie-spread | 7 | 2 | 15 - 625 g jars | 43.9 |
| 64 | Wimmers gute Semmelknödel | 12 | 5 | 20 bags x 4 pieces | 33.25 |
| 69 | Gudbrandsdalsost | 15 | 4 | 10 kg pkg. | 36 |
| 72 | Mozzarella di Giovanni | 14 | 4 | 24 - 200 g pkgs. | 34.8 |

### SQL Statement:

SELECT \* FROM Customers

WHERE CustomerID > 80;

​Number of Records: 11

### SQL Statement:

### EQUAL TO OPERATOR

SELECT \* FROM Products

WHERE Price = 18;

### Result:

Number of Records: 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 35 | Steeleye Stout | 16 | 1 | 24 - 12 oz bottles | 18 |
| 39 | Chartreuse verte | 18 | 1 | 750 cc per bottle | 18 |
| 76 | Lakkalikööri | 23 | 1 | 500 ml | 18 |

**Greater than**

### SQL Statement:

SELECT \* FROM Products

WHERE Price > 30**;**

Number of Records: 24

**LESS THAN**

### SQL Statement:

SELECT \* FROM Products

WHERE Price < 30;

### Result:

### Number of Records: 10

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 |
| 6 | Grandma's Boysenberry Spread | 3 | 2 | 12 - 8 oz jars | 25 |
| 11 | Queso Cabrales | 5 | 4 | 1 kg pkg. | 21 |
| 13 | Konbu | 6 | 8 | 2 kg box | 6 |
| 14 | Tofu | 6 | 7 | 40 - 100 g pkgs. | 23.25 |
| 15 | Genen Shouyu | 6 | 2 | 24 - 250 ml bottles | 15.5 |

**Greater than or equal**

### SQL Statement:

SELECT \* FROM Products

WHERE Price >= 30**;**

### Result:

Number of Records: 12

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 7 | Uncle Bob's Organic Dried Pears | 3 | 7 | 12 - 1 lb pkgs. | 30 |
| 8 | Northwoods Cranberry Sauce | 3 | 2 | 12 - 12 oz jars | 40 |
| 9 | Mishi Kobe Niku | 4 | 6 | 18 - 500 g pkgs. | 97 |
| 10 | Ikura | 4 | 8 | 12 - 200 ml jars | 31 |
| 12 | Queso Manchego La Pastora | 5 | 4 | 10 - 500 g pkgs. | 38 |
| 17 | Alice Mutton | 7 | 6 | 20 - 1 kg tins | 39 |
| 18 | Carnarvon Tigers | 7 | 8 | 16 kg pkg. | 62.5 |
| 20 | Sir Rodney's Marmalade | 8 | 3 | 30 gift boxes | 81 |
| 26 | Gumbär Gummibärchen | 11 | 3 | 100 - 250 g bags | 31.23 |
| 27 | Schoggi Schokolade | 11 | 3 | 100 - 100 g pieces | 43.9 |
| 28 | Rössle Sauerkraut | 12 | 7 | 25 - 825 g cans | 45.6 |
| 29 | Thüringer Rostbratwurst | 12 | 6 | 50 bags x 30 sausgs. | 123.79 |

**Less than or equal**

### SQL Statement:

SELECT \* FROM Products

WHERE Price <= 30;

### Result:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 |
| 6 | Grandma's Boysenberry Spread | 3 | 2 | 12 - 8 oz jars | 25 |
| 7 | Uncle Bob's Organic Dried Pears | 3 | 7 | 12 - 1 lb pkgs. | 30 |
| 11 | Queso Cabrales | 5 | 4 | 1 kg pkg. | 21 |
| 13 | Konbu | 6 | 8 | 2 kg box | 6 |
| 14 | Tofu | 6 | 7 | 40 - 100 g pkgs. | 23.25 |
| 15 | Genen Shouyu | 6 | 2 | 24 - 250 ml bottles | 15.5 |
| 16 | Pavlova | 7 | 3 | 32 - 500 g boxes | 17.45 |
| 19 | Teatime Chocolate Biscuits | 8 | 3 | 10 boxes x 12 pieces | 9.2 |
| 21 | Sir Rodney's Scones | 8 | 3 | 24 pkgs. x 4 pieces | 10 |

Number of Records: 14

**Not equal. Note: In some versions of SQL this operator may be written as !=**

### SQL Statement:

SELECT \* FROM Products

WHERE Price <> 18;

Number of Records: 12

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 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 |
| 6 | Grandma's Boysenberry Spread | 3 | 2 | 12 - 8 oz jars | 25 |
| 7 | Uncle Bob's Organic Dried Pears | 3 | 7 | 12 - 1 lb pkgs. | 30 |
| 8 | Northwoods Cranberry Sauce | 3 | 2 | 12 - 12 oz jars | 40 |
| 9 | Mishi Kobe Niku | 4 | 6 | 18 - 500 g pkgs. | 97 |
| 10 | Ikura | 4 | 8 | 12 - 200 ml jars | 31 |
| 11 | Queso Cabrales | 5 | 4 | 1 kg pkg. | 21 |
| 12 | Queso Manchego La Pastora | 5 | 4 | 10 - 500 g pkgs. | 38 |
| 13 | Konbu | 6 | 8 | 2 kg box | 6 |

### BETWEEN A CERTAIN RANGE

### SQL Statement:

SELECT \* FROM Products

WHERE Price BETWEEN 50 AND 60;

Number of Records: 2

### ​ Result:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** | | 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France | | 15 | Comércio Mineiro | Pedro Afonso | Av. dos Lusíadas, 23 | São Paulo | 05432-043 | Brazil | | 21 | Familia Arquibaldo | Aria Cruz | Rua Orós, 92 | São Paulo | 05442-030 | Brazil | | 30 | Godos Cocina Típica | José Pedro Freyre | C/ Romero, 33 | Sevilla | 41101 | Spain | | 35 | HILARIÓN-Abastos | Carlos Hernández | Carrera 22 con Ave. Carlos Soublette #8-35 | San Cristóbal | 5022 | Venezuela | | 45 | Let's Stop N Shop | Jaime Yorres | 87 Polk St. Suite 5 | San Francisco | 94117 | USA | | 59 | Piccolo und mehr | Georg Pipps | Geislweg 14 | Salzburg | 5020 | Austria | | 62 | Queen Cozinha | Lúcia Carvalho | Alameda dos Canàrios, 891 | São Paulo | 05487-020 | Brazil | | 70 | Santé Gourmet | Jonas Bergulfsen | Erling Skakkes gate 78 | Stavern | 4110 | Norway | | 81 | Tradição Hipermercados | Anabela Domingues | Av. Inês de Castro, 414 | São Paulo | 05634-030 | Brazil | | 86 | Die Wandernde Kuh | Rita Müller | Adenauerallee 900 | Stuttgart | 70563 | Germany | | 89 | White Clover Markets | Karl Jablonski | 305 - 14th Ave. S. Suite 3B | Seattle | 98128 | USA |   51 | Manjimup Dried Apples | 24 | 7 | 50 - 300 g pkgs. | 53 |
| 59 | Raclette Courdavault | 28 | 4 | 5 kg pkg. | 55 |

### Search for a pattern

### SQL Statement:

SELECT \* FROM Customers

WHERE City LIKE 's%';

### Result:

Number of Records: 8

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 11 | B's Beverages | Victoria Ashworth | Fauntleroy Circus | London | EC2 5NT | UK |
| 16 | Consolidated Holdings | Elizabeth Brown | Berkeley Gardens 12 Brewery | London | WX1 6LT | UK |
| 19 | Eastern Connection | Ann Devon | 35 King George | London | WX3 6FW | UK |
| 53 | North/South | Simon Crowther | South House 300 Queensbridge | London | SW7 1RZ | UK |
| 57 | Paris spécialités | Marie Bertrand | 265, boulevard Charonne | Paris | 75012 | France |
| 72 | Seven Seas Imports | Hari Kumar | 90 Wadhurst Rd. | London | OX15 4NB | UK |
| 74 | Spécialités du monde | Dominique Perrier | 25, rue Lauriston | Paris | 75016 | France |

**IN** To specify multiple possible values for a column

### SQL Statement:

SELECT \* FROM Customers

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

### ​ Result:

Number of Records: 8

## The SQL ORDER BY

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

### SQL Statement:

SELECT \* FROM Products

ORDER BY Price;

### Result:

Number of Records: 33

## Syntax

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

## 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 |

## DESC

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

### Example

Sort the products from highest to lowest price:

SELECT \* FROM Products  
ORDER BY Price DESC;

**RESULT**

Number of Records: 77

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 38 | Côte de Blaye | 18 | 1 | 12 - 75 cl bottles | 263.5 |
| 29 | Thüringer Rostbratwurst | 12 | 6 | 50 bags x 30 sausgs. | 123.79 |
| 9 | Mishi Kobe Niku | 4 | 6 | 18 - 500 g pkgs. | 97 |
| 20 | Sir Rodney's Marmalade | 8 | 3 | 30 gift boxes | 81 |
| 18 | Carnarvon Tigers | 7 | 8 | 16 kg pkg. | 62.5 |
| 59 | Raclette Courdavault | 28 | 4 | 5 kg pkg. | 55 |
| 51 | Manjimup Dried Apples | 24 | 7 | 50 - 300 g pkgs. | 53 |
| 62 | Tarte au sucre | 29 | 3 | 48 pies | 49.3 |
| 43 | Ipoh Coffee | 20 | 1 | 16 - 500 g tins | 46 |
| 28 | Rössle Sauerkraut | 12 | 7 | 25 - 825 g cans | 45.6 |
| 27 | Schoggi Schokolade | 11 | 3 | 100 - 100 g pieces | 43.9 |
| 63 | Vegie-spread | 7 | 2 | 15 - 625 g jars | 43.9 |
| 8 | Northwoods Cranberry Sauce | 3 | 2 | 12 - 12 oz jars | 40 |
| 17 | Alice Mutton | 7 | 6 | 20 - 1 kg tins | 39 |
| 56 | Gnocchi di nonna Alice | 26 | 5 | 24 - 250 g pkgs. | 38 |
| 12 | Queso Manchego La Pastora | 5 | 4 | 10 - 500 g pkgs. | 38 |
| 69 | Gudbrandsdalsost | 15 | 4 | 10 kg pkg. | 36 |
| 72 | Mozzarella di Giovanni | 14 | 4 | 24 - 200 g pkgs. | 34.8 |
| 60 | Camembert Pierrot | 28 | 4 | 15 - 300 g rounds | 34 |
| 64 | Wimmers gute Semmelknödel | 12 | 5 | 20 bags x 4 pieces | 33.25 |
| 53 | Perth Pasties | 24 | 6 | 48 pieces | 32.8 |
| 32 | Mascarpone Fabioli | 14 | 4 | 24 - 200 g pkgs. | 32 |
| 26 | Gumbär Gummibärchen | 11 | 3 | 100 - 250 g bags | 31.23 |
| 10 | Ikura | 4 | 8 | 12 - 200 ml jars | 31 |
| 7 | Uncle Bob's Organic Dried Pears | 3 | 7 | 12 - 1 lb pkgs. | 30 |
| 61 | Sirop d'érable | 29 | 2 | 24 - 500 ml bottles | 28.5 |
| 37 | Gravad lax | 17 | 8 | 12 - 500 g pkgs. | 26 |
| 30 | Nord-Ost Matjeshering | 13 | 8 | 10 - 200 g glasses | 25.89 |
| 6 | Grandma's Boysenberry Spread | 3 | 2 | 12 - 8 oz jars | 25 |
| 55 | Pâté chinois | 25 | 6 | 24 boxes x 2 pies | 24 |
| 14 | Tofu | 6 | 7 | 40 - 100 g pkgs. | 23.25 |
| 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 22 |
| 71 | Fløtemysost | 15 | 4 | 10 - 500 g pkgs. | 21.5 |
| 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 21.35 |
| 65 | Louisiana Fiery Hot Pepper Sauce | 2 | 2 | 32 - 8 oz bottles | 21.05 |
| 22 | Gustaf's Knäckebröd | 9 | 5 | 24 - 500 g pkgs. | 21 |
| 11 | Queso Cabrales | 5 | 4 | 1 kg pkg. | 21 |
| 49 | Maxilaku | 23 | 3 | 24 - 50 g pkgs. | 20 |
| 57 | Ravioli Angelo | 26 | 5 | 24 - 250 g pkgs. | 19.5 |
| 44 | Gula Malacca | 20 | 2 | 20 - 2 kg bags | 19.45 |
| 36 | Inlagd Sill | 17 | 8 | 24 - 250 g jars | 19 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 40 | Boston Crab Meat | 19 | 8 | 24 - 4 oz tins | 18.4 |
| 39 | Chartreuse verte | 18 | 1 | 750 cc per bottle | 18 |
| 76 | Lakkalikööri | 23 | 1 | 500 ml | 18 |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 35 | Steeleye Stout | 16 | 1 | 24 - 12 oz bottles | 18 |
| 16 | Pavlova | 7 | 3 | 32 - 500 g boxes | 17.45 |
| 66 | Louisiana Hot Spiced Okra | 2 | 2 | 24 - 8 oz jars | 17 |
| 50 | Valkoinen suklaa | 23 | 3 | 12 - 100 g bars | 16.25 |
| 15 | Genen Shouyu | 6 | 2 | 24 - 250 ml bottles | 15.5 |
| 70 | Outback Lager | 7 | 1 | 24 - 355 ml bottles | 15 |
| 73 | Röd Kaviar | 17 | 8 | 24 - 150 g jars | 15 |
| 25 | NuNuCa Nuß-Nougat-Creme | 11 | 3 | 20 - 450 g glasses | 14 |
| 42 | Singaporean Hokkien Fried Mee | 20 | 5 | 32 - 1 kg pkgs. | 14 |
| 34 | Sasquatch Ale | 16 | 1 | 24 - 12 oz bottles | 14 |
| 67 | Laughing Lumberjack Lager | 16 | 1 | 24 - 12 oz bottles | 14 |

## Order Alphabetically

For string values the ORDER BY keyword will order alphabetically:

### Example

Sort the products alphatbetically by ProductName:

### SQL Statement:

SELECT \* FROM Products  
ORDER BY ProductName;

### Result:

Number of Records: 18

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 17 | Alice Mutton | 7 | 6 | 20 - 1 kg tins | 39 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 40 | Boston Crab Meat | 19 | 8 | 24 - 4 oz tins | 18.4 |
| 60 | Camembert Pierrot | 28 | 4 | 15 - 300 g rounds | 34 |
| 18 | Carnarvon Tigers | 7 | 8 | 16 kg pkg. | 62.5 |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 39 | Chartreuse verte | 18 | 1 | 750 cc per bottle | 18 |
| 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 |
| 48 | Chocolade | 22 | 3 | 10 pkgs. | 12.75 |
| 38 | Côte de Blaye | 18 | 1 | 12 - 75 cl bottles | 263.5 |
| 58 | Escargots de Bourgogne | 27 | 8 | 24 pieces | 13.25 |
| 52 | Filo Mix | 24 | 5 | 16 - 2 kg boxes | 7 |
| 71 | Fløtemysost | 15 | 4 | 10 - 500 g pkgs. | 21.5 |
| 33 | Geitost | 15 | 4 | 500 g | 2.5 |
| 15 | Genen Shouyu | 6 | 2 | 24 - 250 ml bottles | 15.5 |
| 56 | Gnocchi di nonna Alice | 26 | 5 | 24 - 250 g pkgs. | 38 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 47 | Zaanse koeken | 22 | 3 | 10 - 4 oz boxes | 9.5 |
| 64 | Wimmers gute Semmelknödel | 12 | 5 | 20 bags x 4 pieces | 33.25 |
| 63 | Vegie-spread | 7 | 2 | 15 - 625 g jars | 43.9 |
| 50 | Valkoinen suklaa | 23 | 3 | 12 - 100 g bars | 16.25 |
| 7 | Uncle Bob's Organic Dried Pears | 3 | 7 | 12 - 1 lb pkgs. | 30 |
| 23 | Tunnbröd | 9 | 5 | 12 - 250 g pkgs. | 9 |
| 54 | Tourtière | 25 | 6 | 16 pies | 7.45 |
| 14 | Tofu | 6 | 7 | 40 - 100 g pkgs. | 23.25 |
| 29 | Thüringer Rostbratwurst | 12 | 6 | 50 bags x 30 sausgs. | 123.79 |
| 19 | Teatime Chocolate Biscuits | 8 | 3 | 10 boxes x 12 pieces | 9.2 |
| 62 | Tarte au sucre | 29 | 3 | 48 pies | 49.3 |
| 35 | Steeleye Stout | 16 | 1 | 24 - 12 oz bottles | 18 |
| 46 | Spegesild | 21 | 8 | 4 - 450 g glasses | 12 |
| 61 | Sirop d'érable | 29 | 2 | 24 - 500 ml bottles | 28.5 |
| 21 | Sir Rodney's Scones | 8 | 3 | 24 pkgs. x 4 pieces | 10 |
| 20 | Sir Rodney's Marmalade | 8 | 3 | 30 gift boxes | 81 |
| 42 | Singaporean Hokkien Fried Mee | 20 | 5 | 32 - 1 kg pkgs. | 14 |
| 68 | Scottish Longbreads | 8 | 3 | 10 boxes x 8 pieces | 12.5 |
| 27 | Schoggi Schokolade | 11 | 3 | 100 - 100 g pieces | 43.9 |
| 34 | Sasquatch Ale | 16 | 1 | 24 - 12 oz bottles | 14 |

## Alphabetically DESC

To sort the table reverse alphabetically, use the DESC keyword:

### Example

Sort the products by ProductName in reverse order:

### SQL Statement:

SELECT \* FROM Products  
ORDER BY ProductName DESC;

Number of Records: 77

## 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:

### Example

### SQL Statement:

SELECT \* FROM Customers

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 12 | Cactus Comidas para llevar | Patricio Simpson | Cerrito 333 | Buenos Aires | 1010 | Argentina |
| 54 | Océano Atlántico Ltda. | Yvonne Moncada | Ing. Gustavo Moncada 8585 Piso 20-A | Buenos Aires | 1010 | Argentina |
| 64 | Rancho grande | Sergio Gutiérrez | Av. del Libertador 900 | Buenos Aires | 1010 | Argentina |
| 20 | Ernst Handel | Roland Mendel | Kirchgasse 6 | Graz | 8010 | Austria |
| 59 | Piccolo und mehr | Georg Pipps | Geislweg 14 | Salzburg | 5020 | Austria |
| 50 | Maison Dewey | Catherine Dewey | Rue Joseph-Bens 532 | Bruxelles | B-1180 | Belgium |
| 76 | Suprêmes délices | Pascale Cartrain | Boulevard Tirou, 255 | Charleroi | B-6000 | Belgium |
| 15 | Comércio Mineiro | Pedro Afonso | Av. dos Lusíadas, 23 | São Paulo | 05432-043 | Brazil |
| 21 | Familia Arquibaldo | Aria Cruz | Rua Orós, 92 | São Paulo | 05442-030 | Brazil |

### Result:

Number of Records: 9

## The SQL AND Operator

The WHERE clause can contain one or many AND operators.

The AND operator is used to filter records based on more than one condition, like if you want to return all customers from Spain that starts with the letter 'G':

### Example

Select all customers from Spain that starts with the letter 'G':

SELECT \*  
FROM Customers  
WHERE Country = 'Spain' AND CustomerName LIKE 'G%';

### Result:

Number of Records: 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 29 | Galería del gastrónomo | Eduardo Saavedra | Rambla de Cataluña, 23 | Barcelona | 08022 | Spain |
| 30 | Godos Cocina Típica | José Pedro Freyre | C/ Romero, 33 | Sevilla | 41101 | Spain |

## Syntax

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

## AND vs OR

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

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

## Demo Database

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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 |

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

## All Conditions Must Be True

The following SQL statement selects all fields from Customers where Country is "Germany" AND City is "Berlin" AND PostalCode is higher than 12000:

### Example

SELECT \* FROM Customers  
WHERE Country = 'Germany'  
AND City = 'Berlin'  
AND PostalCode > '12000';

### Result:

Number of Records: 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |

## Combining AND and OR

You can combine the AND and OR operators.

The following SQL statement selects all customers from Spain that starts with a "G" or an "R".

Make sure you use parenthesis to get the correct result.

Select all Spanish customers that starts with either "G" or "R":

SELECT \* FROM Customers  
WHERE Country = 'Spain' AND (CustomerName LIKE 'G%' OR CustomerName LIKE 'R%');

### SQL Statement:

SELECT \* FROM Customers

WHERE Country = 'Spain'

AND (CustomerName LIKE 'G%' OR CustomerName LIKE 'R%');

### Result:

Number of Records: 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 29 | Galería del gastrónomo | Eduardo Saavedra | Rambla de Cataluña, 23 | Barcelona | 08022 | Spain |
| 30 | Godos Cocina Típica | José Pedro Freyre | C/ Romero, 33 | Sevilla | 41101 | Spain |
| 69 | Romero y tomillo | Alejandra Camino | Gran Vía, 1 | Madrid | 28001 | Spain |

Without parenthesis, the select statement will return all customers from Spain that starts with a "G", plus all customers that starts with an "R", regardless of the country value:

### Example

Select all customers that either:  
are from Spain and starts with either "G", or  
starts with the letter "R":

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 29 | Galería del gastrónomo | Eduardo Saavedra | Rambla de Cataluña, 23 | Barcelona | 08022 | Spain |
| 30 | Godos Cocina Típica | José Pedro Freyre | C/ Romero, 33 | Sevilla | 41101 | Spain |
| 64 | Rancho grande | Sergio Gutiérrez | Av. del Libertador 900 | Buenos Aires | 1010 | Argentina |
| 65 | Rattlesnake Canyon Grocery | Paula Wilson | 2817 Milton Dr. | Albuquerque | 87110 | USA |
| 66 | Reggiani Caseifici | Maurizio Moroni | Strada Provinciale 124 | Reggio Emilia | 42100 | Italy |
| 67 | Ricardo Adocicados | Janete Limeira | Av. Copacabana, 267 | Rio de Janeiro | 02389-890 | Brazil |
| 68 | Richter Supermarkt | Michael Holz | Grenzacherweg 237 | Genève | 1203 | Switzerland |
| 69 | Romero y tomillo | Alejandra Camino | Gran Vía, 1 | Madrid | 28001 | Spain |

SELECT \* FROM Customers  
WHERE Country = 'Spain' AND CustomerName LIKE 'G%' OR CustomerName LIKE 'R%';

### Result:

Number of Records: 8

## The SQL OR Operator

The WHERE clause can contain one or more OR operators.

The OR operator is used to filter records based on more than one condition, like if you want to return all customers from Germany but also those from Spain:

### Example

Select all customers from Germany or Spain:

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

### Result:

Number of Records: 16

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 8 | Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |
| 17 | Drachenblut Delikatessend | Sven Ottlieb | Walserweg 21 | Aachen | 52066 | Germany |
| 22 | FISSA Fabrica Inter. Salchichas S.A. | Diego Roel | C/ Moralzarzal, 86 | Madrid | 28034 | Spain |
| 25 | Frankenversand | Peter Franken | Berliner Platz 43 | München | 80805 | Germany |
| 29 | Galería del gastrónomo | Eduardo Saavedra | Rambla de Cataluña, 23 | Barcelona | 08022 | Spain |
| 30 | Godos Cocina Típica | José Pedro Freyre | C/ Romero, 33 | Sevilla | 41101 | Spain |
| 39 | Königlich Essen | Philip Cramer | Maubelstr. 90 | Brandenburg | 14776 | Germany |
| 44 | Lehmanns Marktstand | Renate Messner | Magazinweg 7 | Frankfurt a.M. | 60528 | Germany |
| 52 | Morgenstern Gesundkost | Alexander Feuer | Heerstr. 22 | Leipzig | 04179 | Germany |
| 56 | Ottilies Käseladen | Henriette Pfalzheim | Mehrheimerstr. 369 | Köln | 50739 | Germany |
| 63 | QUICK-Stop | Horst Kloss | Taucherstraße 10 | Cunewalde | 01307 | Germany |
| 69 | Romero y tomillo | Alejandra Camino | Gran Vía, 1 | Madrid | 28001 | Spain |
| 79 | Toms Spezialitäten | Karin Josephs | Luisenstr. 48 | Münster | 44087 | Germany |
| 86 | Die Wandernde Kuh | Rita Müller | Adenauerallee 900 | Stuttgart | 70563 | Germany |

## Syntax

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

## OR vs AND

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

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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 |

## Demo Database

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

## At Least One Condition Must Be True

The following SQL statement selects all fields from Customers where either City is "Berlin", CustomerName starts with the letter "G" or Country is "Norway":

### Example

SELECT \* FROM Customers  
WHERE City = 'Berlin' OR CustomerName LIKE 'G%' OR Country = 'Norway';

### Result:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 29 | Galería del gastrónomo | Eduardo Saavedra | Rambla de Cataluña, 23 | Barcelona | 08022 | Spain |
| 30 | Godos Cocina Típica | José Pedro Freyre | C/ Romero, 33 | Sevilla | 41101 | Spain |
| 31 | Gourmet Lanchonetes | André Fonseca | Av. Brasil, 442 | Campinas | 04876-786 | Brazil |
| 32 | Great Lakes Food Market | Howard Snyder | 2732 Baker Blvd. | Eugene | 97403 | USA |
| 33 | GROSELLA-Restaurante | Manuel Pereira | 5ª Ave. Los Palos Grandes | Caracas | 1081 | Venezuela |
| 70 | Santé Gourmet | Jonas Bergulfsen | Erling Skakkes gate 78 | Stavern | 4110 | Norway |

Number of Records: 7

## Combining AND and OR

You can combine the AND and OR operators.

The following SQL statement selects all customers from Spain that starts with a "G" or an "R".

Make sure you use parenthesis to get the correct result.

### Example

Select all Spanish customers that starts with either "G" or "R":

SELECT \* FROM Customers  
WHERE Country = 'Spain' AND (CustomerName LIKE 'G%' OR CustomerName LIKE 'R%');

### Result:

Number of Records: 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 29 | Galería del gastrónomo | Eduardo Saavedra | Rambla de Cataluña, 23 | Barcelona | 08022 | Spain |
| 30 | Godos Cocina Típica | José Pedro Freyre | C/ Romero, 33 | Sevilla | 41101 | Spain |
| 69 | Romero y tomillo | Alejandra Camino | Gran Vía, 1 | Madrid | 28001 | Spain |

Without parenthesis, the select statement will return all customers from Spain that starts with a "G", plus all customers that starts with an "R", regardless of the country value:

### Example

Select all customers that either:  
are from Spain and starts with either "G", or  
starts with the letter "R":

SELECT \* FROM Customers  
WHERE Country = 'Spain' AND CustomerName LIKE 'G%' OR CustomerName LIKE 'R%';

### Result:

Number of Records: 8

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 29 | Galería del gastrónomo | Eduardo Saavedra | Rambla de Cataluña, 23 | Barcelona | 08022 | Spain |
| 30 | Godos Cocina Típica | José Pedro Freyre | C/ Romero, 33 | Sevilla | 41101 | Spain |
| 64 | Rancho grande | Sergio Gutiérrez | Av. del Libertador 900 | Buenos Aires | 1010 | Argentina |
| 65 | Rattlesnake Canyon Grocery | Paula Wilson | 2817 Milton Dr. | Albuquerque | 87110 | USA |
| 66 | Reggiani Caseifici | Maurizio Moroni | Strada Provinciale 124 | Reggio Emilia | 42100 | Italy |
| 67 | Ricardo Adocicados | Janete Limeira | Av. Copacabana, 267 | Rio de Janeiro | 02389-890 | Brazil |
| 68 | Richter Supermarkt | Michael Holz | Grenzacherweg 237 | Genève | 1203 | Switzerland |
| 69 | Romero y tomillo | Alejandra Camino | Gran Vía, 1 | Madrid | 28001 | Spain |

## The NOT Operator

The NOT operator is used in combination with other operators to give the opposite result, also called the negative result.

In the select statement below we want to return all customers that are NOT from Spain:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 9 | Bon app' | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |
| 10 | Bottom-Dollar Marketse | Elizabeth Lincoln | 23 Tsawassen Blvd. | Tsawassen | T2F 8M4 | Canada |
| 11 | B's Beverages | Victoria Ashworth | Fauntleroy Circus | London | EC2 5NT | UK |
| 12 | Cactus Comidas para llevar | Patricio Simpson | Cerrito 333 | Buenos Aires | 1010 | Argentina |
| 13 | Centro comercial Moctezuma | Francisco Chang | Sierras de Granada 9993 | México D.F. | 05022 | Mexico |
| 14 | Chop-suey Chinese | Yang Wang | Hauptstr. 29 | Bern | 3012 | Switzerland |
| 15 | Comércio Mineiro | Pedro Afonso | Av. dos Lusíadas, 23 | São Paulo | 05432-043 | Brazil |

### Example

Select only the customers that are NOT from Spain:

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

### SQL Statement:

SELECT \* FROM Customers

WHERE NOT Country = 'Spain';

### Result:

Number of Records: 86

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 9 | Bon app' | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |
| 10 | Bottom-Dollar Marketse | Elizabeth Lincoln | 23 Tsawassen Blvd. | Tsawassen | T2F 8M4 | Canada |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 8 | Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |
| 9 | Bon app' | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |
| 10 | Bottom-Dollar Marketse | Elizabeth Lincoln | 23 Tsawassen Blvd. | Tsawassen | T2F 8M4 | Canada |
| 11 | B's Beverages | Victoria Ashworth | Fauntleroy Circus | London | EC2 5NT | UK |
| 12 | Cactus Comidas para llevar | Patricio Simpson | Cerrito 333 | Buenos Aires | 1010 | Argentina |
| 13 | Centro comercial Moctezuma | Francisco Chang | Sierras de Granada 9993 | México D.F. | 05022 | Mexico |
| 14 | Chop-suey Chinese | Yang Wang | Hauptstr. 29 | Bern | 3012 | Switzerland |
| 15 | Comércio Mineiro | Pedro Afonso | Av. dos Lusíadas, 23 | São Paulo | 05432-043 | Brazil |
| 16 | Consolidated Holdings | Elizabeth Brown | Berkeley Gardens 12 Brewery | London | WX1 6LT | UK |
| 17 | Drachenblut Delikatessend | Sven Ottlieb | Walserweg 21 | Aachen | 52066 | Germany |
| 18 | Du monde entier | Janine Labrune | 67, rue des Cinquante Otages | Nantes | 44000 | France |
| 19 | Eastern Connection | Ann Devon | 35 King George | London | WX3 6FW | UK |
| 20 | Ernst Handel | Roland Mendel | Kirchgasse 6 | Graz | 8010 | Austria |
| 21 | Familia Arquibaldo | Aria Cruz | Rua Orós, 92 | São Paulo | 05442-030 | Brazil |
| 22 | FISSA Fabrica Inter. Salchichas S.A. | Diego Roel | C/ Moralzarzal, 86 | Madrid | 28034 | Spain |
| 23 | Folies gourmandes | Martine Rancé | 184, chaussée de Tournai | Lille | 59000 | France |
| 24 | Folk och fä HB | Maria Larsson | Åkergatan 24 | Bräcke | S-844 67 | Sweden |
| 25 | Frankenversand | Peter Franken | Berliner Platz 43 | München | 80805 | Germany |
| 26 | France restauration | Carine Schmitt | 54, rue Royale | Nantes | 44000 | France |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 8 | Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |
| 9 | Bon app' | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |
| 61 | Que Delícia | Bernardo Batista | Rua da Panificadora, 12 | Rio de Janeiro | 02389-673 | Brazil |
| 62 | Queen Cozinha | Lúcia Carvalho | Alameda dos Canàrios, 891 | São Paulo | 05487-020 | Brazil |
| 63 | QUICK-Stop | Horst Kloss | Taucherstraße 10 | Cunewalde | 01307 | Germany |
| 64 | Rancho grande | Sergio Gutiérrez | Av. del Libertador 900 | Buenos Aires | 1010 | Argentina |
| 65 | Rattlesnake Canyon Grocery | Paula Wilson | 2817 Milton Dr. | Albuquerque | 87110 | USA |
| 66 | Reggiani Caseifici | Maurizio Moroni | Strada Provinciale 124 | Reggio Emilia | 42100 | Italy |
| 67 | Ricardo Adocicados | Janete Limeira | Av. Copacabana, 267 | Rio de Janeiro | 02389-890 | Brazil |
| 68 | Richter Supermarkt | Michael Holz | Grenzacherweg 237 | Genève | 1203 | Switzerland |
| 69 | Romero y tomillo | Alejandra Camino | Gran Vía, 1 | Madrid | 28001 | Spain |
| 70 | Santé Gourmet | Jonas Bergulfsen | Erling Skakkes gate 78 | Stavern | 4110 | Norway |
| 71 | Save-a-lot Markets | Jose Pavarotti | 187 Suffolk Ln. | Boise | 83720 | USA |
| 72 | Seven Seas Imports | Hari Kumar | 90 Wadhurst Rd. | London | OX15 4NB | UK |
| 73 | Simons bistro | Jytte Petersen | Vinbæltet 34 | København | 1734 | Denmark |
| 74 | Spécialités du monde | Dominique Perrier | 25, rue Lauriston | Paris | 75016 | France |
| 75 | Split Rail Beer & Ale | Art Braunschweiger | P.O. Box 555 | Lander | 82520 | USA |
| 76 | Suprêmes délices | Pascale Cartrain | Boulevard Tirou, 255 | Charleroi | B-6000 | Belgium |
| 77 | The Big Cheese | Liz Nixon | 89 Jefferson Way Suite 2 | Portland | 97201 | USA |
| 78 | The Cracker Box | Liu Wong | 55 Grizzly Peak Rd. | Butte | 59801 | USA |
| 79 | Toms Spezialitäten | Karin Josephs | Luisenstr. 48 | Münster | 44087 | Germany |
| 80 | Tortuga Restaurante | Miguel Angel Paolino | Avda. Azteca 123 | México D.F. | 05033 | Mexico |
| 81 | Tradição Hipermercados | Anabela Domingues | Av. Inês de Castro, 414 | São Paulo | 05634-030 | Brazil |
| 82 | Trail's Head Gourmet Provisioners | Helvetius Nagy | 722 DaVinci Blvd. | Kirkland | 98034 | USA |
| 83 | Vaffeljernet | Palle Ibsen | Smagsløget 45 | Århus | 8200 | Denmark |
| 84 | Victuailles en stock | Mary Saveley | 2, rue du Commerce | Lyon | 69004 | France |
| 85 | Vins et alcools Chevalier | Paul Henriot | 59 rue de l'Abbaye | Reims | 51100 | France |
| 86 | Die Wandernde Kuh | Rita Müller | Adenauerallee 900 | Stuttgart | 70563 | Germany |
| 87 | Wartian Herkku | Pirkko Koskitalo | Torikatu 38 | Oulu | 90110 | Finland |
| 88 | Wellington Importadora | Paula Parente | Rua do Mercado, 12 | Resende | 08737-363 | Brazil |
| 89 | White Clover Markets | Karl Jablonski | 305 - 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |

In the example above, the NOT operator is used in combination with the = operator, but it can be used in combination with other comparison and/or logical operators. See examples below.

## Syntax

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

## Demo Database

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

## NOT LIKE

### Example

Select customers that does not start with the letter 'A':

SELECT \* FROM Customers  
WHERE CustomerName NOT LIKE 'A%';

### SQL Statement:

SELECT \* FROM Customers

WHERE CustomerName NOT LIKE 'A%';

### Result:

Number of Records: 87

## NOT BETWEEN

### Example

Select customers with a customerID not between 10 and 60:

SELECT \* FROM Customers  
WHERE CustomerID NOT BETWEEN 10 AND 60;

### Result:

## NOT IN

### Example

Select customers that are not from Paris or London:

SELECT \* FROM Customers  
WHERE City NOT IN ('Paris', 'London');

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 8 | Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |
| 9 | Bon app' | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |
| 10 | Bottom-Dollar Marketse | Elizabeth Lincoln | 23 Tsawassen Blvd. | Tsawassen | T2F 8M4 | Canada |
| 11 | B's Beverages | Victoria Ashworth | Fauntleroy Circus | London | EC2 5NT | UK |
| 12 | Cactus Comidas para llevar | Patricio Simpson | Cerrito 333 | Buenos Aires | 1010 | Argentina |
| 13 | Centro comercial Moctezuma | Francisco Chang | Sierras de Granada 9993 | México D.F. | 05022 | Mexico |
| 14 | Chop-suey Chinese | Yang Wang | Hauptstr. 29 | Bern | 3012 | Switzerland |
| 15 | Comércio Mineiro | Pedro Afonso | Av. dos Lusíadas, 23 | São Paulo | 05432-043 | Brazil |
| 16 | Consolidated Holdings | Elizabeth Brown | Berkeley Gardens 12 Brewery | London | WX1 6LT | UK |
| 17 | Drachenblut Delikatessend | Sven Ottlieb | Walserweg 21 | Aachen | 52066 | Germany |
| 18 | Du monde entier | Janine Labrune | 67, rue des Cinquante Otages | Nantes | 44000 | France |
| 19 | Eastern Connection | Ann Devon | 35 King George | London | WX3 6FW | UK |
| 20 | Ernst Handel | Roland Mendel | Kirchgasse 6 | Graz | 8010 | Austria |
| 21 | Familia Arquibaldo | Aria Cruz | Rua Orós, 92 | São Paulo | 05442-030 | Brazil |
| 22 | FISSA Fabrica Inter. Salchichas S.A. | Diego Roel | C/ Moralzarzal, 86 | Madrid | 28034 | Spain |
| 23 | Folies gourmandes | Martine Rancé | 184, chaussée de Tournai | Lille | 59000 | France |
| 24 | Folk och fä HB | Maria Larsson | Åkergatan 24 | Bräcke | S-844 67 | Sweden |
| 25 | Frankenversand | Peter Franken | Berliner Platz 43 | München | 80805 | Germany |
| 26 | France restauration | Carine Schmitt | 54, rue Royale | Nantes | 44000 | France |
| 27 | Franchi S.p.A. | Paolo Accorti | Via Monte Bianco 34 | Torino | 10100 | Italy |
| 28 | Furia Bacalhau e Frutos do Mar | Lino Rodriguez | Jardim das rosas n. 32 | Lisboa | 1675 | Portugal |
| 29 | Galería del gastrónomo | Eduardo Saavedra | Rambla de Cataluña, 23 | Barcelona | 08022 | Spain |
| 30 | Godos Cocina Típica | José Pedro Freyre | C/ Romero, 33 | Sevilla | 41101 | Spain |
| 31 | Gourmet Lanchonetes | André Fonseca | Av. Brasil, 442 | Campinas | 04876-786 | Brazil |
| 32 | Great Lakes Food Market | Howard Snyder | 2732 Baker Blvd. | Eugene | 97403 | USA |
| 33 | GROSELLA-Restaurante | Manuel Pereira | 5ª Ave. Los Palos Grandes | Caracas | 1081 | Venezuela |
| 34 | Hanari Carnes | Mario Pontes | Rua do Paço, 67 | Rio de Janeiro | 05454-876 | Brazil |
| 35 | HILARIÓN-Abastos | Carlos Hernández | Carrera 22 con Ave. Carlos Soublette #8-35 | San Cristóbal | 5022 | Venezuela |
| 36 | Hungry Coyote Import Store | Yoshi Latimer | City Center Plaza 516 Main St. | Elgin | 97827 | USA |
| 37 | Hungry Owl All-Night Grocers | Patricia McKenna | 8 Johnstown Road | Cork |  | Ireland |
| 38 | Island Trading | Helen Bennett | Garden House Crowther Way | Cowes | PO31 7PJ | UK |
| 39 | Königlich Essen | Philip Cramer | Maubelstr. 90 | Brandenburg | 14776 | Germany |
| 40 | La corne d'abondance | Daniel Tonini | 67, avenue de l'Europe | Versailles | 78000 | France |
| 41 | La maison d'Asie | Annette Roulet | 1 rue Alsace-Lorraine | Toulouse | 31000 | France |
| 42 | Laughing Bacchus Wine Cellars | Yoshi Tannamuri | 1900 Oak St. | Vancouver | V3F 2K1 | Canada |
| 43 | Lazy K Kountry Store | John Steel | 12 Orchestra Terrace | Walla Walla | 99362 | USA |
| 44 | Lehmanns Marktstand | Renate Messner | Magazinweg 7 | Frankfurt a.M. | 60528 | Germany |
| 45 | Let's Stop N Shop | Jaime Yorres | 87 Polk St. Suite 5 | San Francisco | 94117 | USA |
| 46 | LILA-Supermercado | Carlos González | Carrera 52 con Ave. Bolívar #65-98 Llano Largo | Barquisimeto | 3508 | Venezuela |
| 47 | LINO-Delicateses | Felipe Izquierdo | Ave. 5 de Mayo Porlamar | I. de Margarita | 4980 | Venezuela |
| 48 | Lonesome Pine Restaurant | Fran Wilson | 89 Chiaroscuro Rd. | Portland | 97219 | USA |
| 49 | Magazzini Alimentari Riuniti | Giovanni Rovelli | Via Ludovico il Moro 22 | Bergamo | 24100 | Italy |
| 50 | Maison Dewey | Catherine Dewey | Rue Joseph-Bens 532 | Bruxelles | B-1180 | Belgium |

### Result:

Number of Records: 9

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 8 | Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |
| 9 | Bon app' | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |
| 10 | Bottom-Dollar Marketse | Elizabeth Lincoln | 23 Tsawassen Blvd. | Tsawassen | T2F 8M4 | Canada |

## NOT Greater Than

### Example

Select customers with a CustomerId not greater than 50:

SELECT \* FROM Customers  
WHERE NOT CustomerID > 50;

### Result:

Number of Records: 50

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** | | 50 | Maison Dewey | Catherine Dewey | Rue Joseph-Bens 532 | Bruxelles | B-1180 | Belgium | | 51 | Mère Paillarde | Jean Fresnière | 43 rue St. Laurent | Montréal | H1J 1C3 | Canada | | 52 | Morgenstern Gesundkost | Alexander Feuer | Heerstr. 22 | Leipzig | 04179 | Germany | | 53 | North/South | Simon Crowther | South House 300 Queensbridge | London | SW7 1RZ | UK | | 54 | Océano Atlántico Ltda. | Yvonne Moncada | Ing. Gustavo Moncada 8585 Piso 20-A | Buenos Aires | 1010 | Argentina | | 55 | Old World Delicatessen | Rene Phillips | 2743 Bering St. | Anchorage | 99508 | USA | | 56 | Ottilies Käseladen | Henriette Pfalzheim | Mehrheimerstr. 369 | Köln | 50739 | Germany | | 57 | Paris spécialités | Marie Bertrand | 265, boulevard Charonne | Paris | 75012 | France | | 58 | Pericles Comidas clásicas | Guillermo Fernández | Calle Dr. Jorge Cash 321 | México D.F. | 05033 | Mexico | | 59 | Piccolo und mehr | Georg Pipps | Geislweg 14 | Salzburg | 5020 | Austria | | 60 | Princesa Isabel Vinhoss | Isabel de Castro | Estrada da saúde n. 58 | Lisboa | 1756 | Portugal |   7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 8 | Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |
| 9 | Bon app' | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |
| 10 | Bottom-Dollar Marketse | Elizabeth Lincoln | 23 Tsawassen Blvd. | Tsawassen | T2F 8M4 | Canada |

## NOT Less Than

### Example

Select customers with a CustomerID not less than 50:

SELECT \* FROM Customers  
WHERE NOT CustomerId < 50;

### Result:

Number of Records: 11

## The SQL INSERT INTO Statement

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

### INSERT INTO Syntax

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

1. Specify both the column names and the values to be inserted:

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

2. 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. Here, the INSERT INTO syntax would be as follows:

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

## Demo Database

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 89 | White Clover Markets | Karl Jablonski | 305 - 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |

## INSERT INTO Example

The following SQL statement inserts a new record in the "Customers" table:

### Example

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 89 | White Clover Markets | Karl Jablonski | 305 - 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |
| 92 | Cardinal | Tom B. Erichsen | Skagen 21 | Stavanger | 4006 | Norway |

The selection from the "Customers" table will now look like this:

**Did you notice that we did not insert any number into the CustomerID field?**  
The CustomerID column is an [auto-increment](https://www.w3schools.com/sql/sql_autoincrement.asp) field and will be generated automatically when a new record is inserted into the table.

## 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):

### Example

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

The selection from the "Customers" table will now look like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 89 | White Clover Markets | Karl Jablonski | 305 - 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |
| 92 | Cardinal | null | null | Stavanger | null | Norway |

## Insert Multiple Rows

It is also possible to insert multiple rows in one statement.

To insert multiple rows of data, we use the same INSERT INTO statement, but with multiple values:

### Example

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)  
VALUES  
('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway'),  
('Greasy Burger', 'Per Olsen', 'Gateveien 15', 'Sandnes', '4306', 'Norway'),  
('Tasty Tee', 'Finn Egan', 'Streetroad 19B', 'Liverpool', 'L1 0AA', 'UK');

Make sure you separate each set of values with a comma ,.

The selection from the "Customers" table will now look like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 89 | White Clover Markets | Karl Jablonski | 305 - 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |
| 92 | Cardinal | Tom B. Erichsen | Skagen 21 | Stavanger | 4006 | Norway |
| 93 | Greasy Burger | Per Olsen | Gateveien 15 | Sandnes | 4306 | Norway |
| 94 | Tasty Tee | Finn Egan | Streetroad 19B | Liverpool | L1 0AA | UK |

## What is a NULL Value?

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. Then, the field will be saved with a NULL value.

**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?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

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;

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

## Demo Database

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

## 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;**

### Result:

|  |  |  |
| --- | --- | --- |
| **CustomerName** | **ContactName** | **Address** |

Number of Records: 0

**Tip:** Always use IS NULL to look for NULL values.

## 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:

### Example

|  |  |  |
| --- | --- | --- |
| **CustomerName** | **ContactName** | **Address** |
| Alfreds Futterkiste | Maria Anders | Obere Str. 57 |
| Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 |
| Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 |
| Around the Horn | Thomas Hardy | 120 Hanover Sq. |
| Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 |
| Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 |
| Blondel père et fils | Frédérique Citeaux | 24, place Kléber |
| Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 |
| Bon app' | Laurence Lebihans | 12, rue des Bouchers |
| Bottom-Dollar Marketse | Elizabeth Lincoln | 23 Tsawassen Blvd. |
| B's Beverages | Victoria Ashworth | Fauntleroy Circus |

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

### Result:

Number of Records: 10

## 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!

## Demo Database

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

## UPDATE Table

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person and a new city.

### Example

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

The selection from the "Customers" table will now look like this:

## UPDATE Multiple Records

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

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

### Example

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

The selection from the "Customers" table will now look like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Alfred Schmidt | Obere Str. 57 | Frankfurt | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Juan | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Juan | 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 |

## Update Warning!

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

### Example

UPDATE Customers  
SET ContactName='Juan';

The selection from the "Customers" table will now look like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

## 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!

## Demo Database

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

## SQL DELETE Example

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

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

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 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 |

The "Customers" table will now look like this:

## 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;

## Delete a Table

To delete the table completely, use the DROP TABLE statement:

### Example

Remove the Customers table:

DROP TABLE Customers;

## 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.

### Example

Select only the first 3 records of the Customers table:

SELECT TOP 3 \* FROM Customers;

### Result:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

Number of Records: 3

**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 FETCH FIRST n ROWS ONLY and ROWNUM.

**SQL Server / MS Access Syntax:**

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

**MySQL Syntax:**

SELECT column\_name(s)  
FROM table\_nameWHERE condition  
LIMIT number;

**Oracle 12 Syntax:**

SELECT column\_name(s)  
FROM table\_nameORDER BY column\_name(s)  
FETCH FIRST number ROWS ONLY;

**Older Oracle Syntax:**

SELECT column\_name(s)  
FROM table\_name  
WHERE ROWNUM <= number;

**Older Oracle Syntax (with ORDER BY):**

SELECT \*  
FROM (SELECT column\_name(s) FROM table\_name ORDER BY column\_name(s))  
WHERE ROWNUM <= number;

## Demo Database

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

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

## LIMIT

The following SQL statement shows the equivalent example for **MySQL:**

### Example

Select the first 3 records of the Customers table:

SELECT \* FROM Customers  
LIMIT 3;

### Result:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

Number of Records: 3

## FETCH FIRST

|  |  |
| --- | --- |
| **PostalCode** | **Country** |
| 12209 | Germany |
| 05021 | Mexico |
| 05023 | Mexico |
| WA1 1DP | UK |
| S-958 22 | Sweden |
| 68306 | Germany |
| 67000 | France |
| 28023 | Spain |
| 13008 | France |
| T2F 8M4 | Canada |
| EC2 5NT | UK |
| 1010 | Argentina |
| 05022 | Mexico |
| 3012 | Switzerland |
| 05432-043 | Brazil |
| WX1 6LT | UK |
| 52066 | Germany |
| 44000 | France |
| WX3 6FW | UK |
| 8010 | Austria |
| 05442-030 | Brazil |
| 28034 | Spain |
| 59000 | France |
| S-844 67 | Sweden |
| 80805 | Germany |
| 44000 | France |
| 10100 | Italy |
| 1675 | Portugal |
| 08022 | Spain |
| 41101 | Spain |
| 04876-786 | Brazil |
| 97403 | USA |
| 1081 | Venezuela |
| 05454-876 | Brazil |
| 5022 | Venezuela |
| 97827 | USA |
|  | Ireland |
| PO31 7PJ | UK |
| 14776 | Germany |
| 78000 | France |
| 31000 | France |
| V3F 2K1 | Canada |
| 99362 | USA |
| 60528 | Germany |
| 94117 | USA |
| 3508 | Venezuela |

The following SQL statement shows the equivalent example for Oracle:

### Example

Select the first 3 records of the Customers table:

SELECT \* FROM Customers  
FETCH FIRST 3 ROWS ONLY;

## SQL TOP PERCENT Example

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

### Example

SELECT TOP 50 PERCENT \* FROM Customers;

### Result:

Number of Records: 46

The following SQL statement shows the equivalent example for Oracle:

### Example

SELECT \* FROM Customers  
FETCH FIRST 50 PERCENT ROWS ONLY;

## 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):

### Example

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

### Result:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 17 | Drachenblut Delikatessend | Sven Ottlieb | Walserweg 21 | Aachen | 52066 | Germany |

Number of Records: 3

The following SQL statement shows the equivalent example for

**Example**

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

### Result:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 17 | Drachenblut Delikatessend | Sven Ottlieb | Walserweg 21 | Aachen | 52066 | Germany |

Number of Records: 3

The following SQL statement shows the equivalent example for Oracle:

### Example

SELECT \* FROM Customers  
WHERE Country='Germany'  
FETCH FIRST 3 ROWS ONLY;

## ADD the ORDER BY Keyword

Add the ORDER BY keyword when you want to sort the result, and return the first 3 records of the sorted result.

For SQL Server and MS Access:

### Example

Sort the result reverse alphabetically by CustomerName, and return the first 3 records:

SELECT TOP 3 \* FROM Customers  
ORDER BY CustomerName DESC;

### Result:

Number of Records: 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 89 | White Clover Markets | Karl Jablonski | 305 - 14th Ave. S. Suite 3B | Seattle | 98128 | USA |

The following SQL statement shows the equivalent example for Oracle:

### Example

SELECT \* FROM Customers  
ORDER BY CustomerName DESC  
FETCH FIRST 3 ROWS ONLY;

## 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

**Find the lowest price**:

SELECT MIN(Price)  
FROM Products;

### Result:

|  |
| --- |
| **Expr1000** |
| 2.5 |

Number of Records: 1

### MAX Example

**Find the highest price**:

SELECT MAX(Price)  
FROM Products;

### Result:

Number of Records: 1

|  |
| --- |
| **Expr1000** |
| 263.5 |

## Syntax

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

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

## Demo Database

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 |

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

## Set Column Name (Alias)

When you use MIN() or MAX(), the returned column will be named MIN(field) or MAX(field) by default. To give the column a new name, use the AS keyword:

### Example

SELECT MIN(Price) AS SmallestPrice  
FROM Products;

### Result:

|  |
| --- |
| **SmallestPrice** |
| 2.5 |

Number of Records: 1

## The SQL COUNT() Function

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

### Example

Find the total number of products in the Products table:

SELECT COUNT(\*)  
FROM Products;

### Result:

|  |
| --- |
| **Expr1000** |
| 77 |

Number of Records: 1

## Syntax

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

## Demo Database

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 |

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

## 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;

### Result:

Number of Records: 1

|  |
| --- |
| **Expr1000** |
| 37 |

## Specify Column

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

If you specify a column 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;

### Result:

Number of Records: 1

|  |
| --- |
| **Expr1000** |
| 77 |

## 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;

### Result:

|  |
| --- |
| **COUNT(DISTINCT Price)** |
| 62 |

Number of Records: 1

## 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;

### Result:

Number of Records: 1

|  |
| --- |
| **number of records** |
| 77 |

## The SQL SUM() Function

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

### Example

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

SELECT SUM(Quantity)  
FROM OrderDetails;

### Result:

|  |
| --- |
| **Expr1000** |
| 12743 |

Number of Records: 1

## Syntax

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

## Demo Database

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

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

## Add a Where Clause

You can add a WHERE clause to specify conditions:

### Example

Return the number of orders made for the product with ProductID 11:

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

### Result:

|  |
| --- |
| **Expr1000** |
| 182 |

Number of Records: 1

## 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;

### Result:

Number of Records: 1

|  |
| --- |
| **total** |
| 12743 |

## 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

Use an expression inside the SUM() function:

SELECT SUM(Quantity \* 10)  
FROM OrderDetails;

### Result:

Number of Records: 1

|  |
| --- |
| **Expr1000** |
| 127430 |

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

### Example

Join OrderDetails with Products, and use SUM() to find the total amount:

SELECT SUM(Price \* Quantity)  
FROM OrderDetails  
LEFT JOIN Products ON OrderDetails.ProductID = Products.ProductID;

### Result:

Number of Records: 1

|  |
| --- |
| **Expr1000** |
| 386424.23 |

## The SQL AVG() Function

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

### Example

Find the average price of all products:

SELECT AVG(Price)  
FROM Products;

### Result:

Number of Records: 1

|  |
| --- |
| **Expr1000** |
| 28.8664 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 |

## 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:

## 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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 7 | Uncle Bob's Organic Dried Pears | 3 | 7 | 12 - 1 lb pkgs. | 30 |
| 8 | Northwoods Cranberry Sauce | 3 | 2 | 12 - 12 oz jars | 40 |
| 9 | Mishi Kobe Niku | 4 | 6 | 18 - 500 g pkgs. | 97 |
| 10 | Ikura | 4 | 8 | 12 - 200 ml jars | 31 |
| 12 | Queso Manchego La Pastora | 5 | 4 | 10 - 500 g pkgs. | 38 |
| 17 | Alice Mutton | 7 | 6 | 20 - 1 kg tins | 39 |
| 18 | Carnarvon Tigers | 7 | 8 | 16 kg pkg. | 62.5 |
| 20 | Sir Rodney's Marmalade | 8 | 3 | 30 gift boxes | 81 |
| 26 | Gumbär Gummibärchen | 11 | 3 | 100 - 250 g bags | 31.23 |
| 27 | Schoggi Schokolade | 11 | 3 | 100 - 100 g pieces | 43.9 |
| 28 | Rössle Sauerkraut | 12 | 7 | 25 - 825 g cans | 45.6 |
| 29 | Thüringer Rostbratwurst | 12 | 6 | 50 bags x 30 sausgs. | 123.79 |
| 32 | Mascarpone Fabioli | 14 | 4 | 24 - 200 g pkgs. | 32 |
| 38 | Côte de Blaye | 18 | 1 | 12 - 75 cl bottles | 263.5 |
| 43 | Ipoh Coffee | 20 | 1 | 16 - 500 g tins | 46 |
| 51 | Manjimup Dried Apples | 24 | 7 | 50 - 300 g pkgs. | 53 |
| 53 | Perth Pasties | 24 | 6 | 48 pieces | 32.8 |
| 56 | Gnocchi di nonna Alice | 26 | 5 | 24 - 250 g pkgs. | 38 |
| 59 | Raclette Courdavault | 28 | 4 | 5 kg pkg. | 55 |
| 60 | Camembert Pierrot | 28 | 4 | 15 - 300 g rounds | 34 |
| 62 | Tarte au sucre | 29 | 3 | 48 pies | 49.3 |
| 63 | Vegie-spread | 7 | 2 | 15 - 625 g jars | 43.9 |
| 64 | Wimmers gute Semmelknödel | 12 | 5 | 20 bags x 4 pieces | 33.25 |
| 69 | Gudbrandsdalsost | 15 | 4 | 10 kg pkg. | 36 |
| 72 | Mozzarella di Giovanni | 14 | 4 | 24 - 200 g pkgs. | 34.8 |

**Name the column "average price":**

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

### Result:

Number of Records: 1

|  |
| --- |
| **average price** |
| 28.8664 |

## 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

### Result:

Number of Records: 25

**SQL**

# 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 sign \_ represents one, single character

### Example

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

Syntax Select all customers that starts with the letter "a":

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

## Number of Records: 4

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

## Demo Database

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

## The \_ Wildcard

The \_ wildcard represents a single character.

It can be any character or number, but each \_ represents one, and only one, character.

### Example

Return all customers from a city that starts with 'L' followed by one wildcard character, then 'nd' and then two wildcard characters:

SELECT \* FROM Customers  
WHERE city LIKE 'L\_nd\_\_';

Number of Records: 7

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 11 | B's Beverages | Victoria Ashworth | Fauntleroy Circus | London | EC2 5NT | UK |
| 16 | Consolidated Holdings | Elizabeth Brown | Berkeley Gardens 12 Brewery | London | WX1 6LT | UK |
| 19 | Eastern Connection | Ann Devon | 35 King George | London | WX3 6FW | UK |
| 53 | North/South | Simon Crowther | South House 300 Queensbridge | London | SW7 1RZ | UK |
| 72 | Seven Seas Imports | Hari Kumar | 90 Wadhurst Rd. | London | OX15 4NB | UK |
| 75 | Split Rail Beer & Ale | Art Braunschweiger | P.O. Box 555 | Lander | 82520 | USA |

## The % Wildcard

The % wildcard represents any number of characters, even zero characters.

### Example

Return all customers from a city that contains the letter 'L':

SELECT \* FROM Customers  
WHERE city LIKE '%L%';

## Starts With

To return records that starts with a specific letter or phrase, add the % at the end of the letter or phrase.

### Example

Return all customers that starts with 'La':

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

**Tip:** You can also combine any number of conditions using AND or OR operators.

### Example

Return all customers that starts with 'a' or starts with 'b':

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

## Ends With

To return records that ends with a specific letter or phrase, add the % at the beginning of the letter or phrase.

### Example

Return all customers that ends with 'a':

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

**Tip:** You can also combine "starts with" and "ends with":

### Example

Return all customers that starts with "b" and ends with "s":

SELECT \* FROM Customers  
WHERE CustomerName LIKE 'b%s';

## Contains

To return records that contains a specific letter or phrase, add the % both before and after the letter or phrase.

### Example

Return all customers that contains the phrase 'or'

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

## Combine Wildcards

Any wildcard, like % and \_ , can be used in combination with other wildcards.

### Example

Return all customers that starts with "a" and are at least 3 characters in length:

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

### Example

Return all customers that have "r" in the second position:

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

## Without Wildcard

If no wildcard is specified, the phrase has to have an exact match to return a result.

### Example

Return all customers from Spain:

SELECT \* FROM Customers  
WHERE Country LIKE 'Spain';

# SQL Wildcards

**SQL Wildcard Characters**

A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the [LIKE](https://www.w3schools.com/sql/sql_like.asp) operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

### Example

Return all customers that starts with the letter 'a':

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

## Wildcard Characters

|  |  |
| --- | --- |
| **Symbol** | **Description** |
| % | Represents zero or more characters |
| \_ | Represents a single character |
| [] | Represents any single character within the brackets \* |
| ^ | Represents any character not in the brackets \* |
| - | Represents any single character within the specified range \* |
| {} | Represents any escaped character \*\* |

\* Not supported in PostgreSQL and MySQL databases.

\*\* Supported only in Oracle databases.

## Demo Database

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

## Using the % Wildcard

The % wildcard represents any number of characters, even zero characters.

### Example

Return all customers that ends with the pattern 'es':

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

### Example

Return all customers that contains the pattern 'mer':

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

## Using the \_ Wildcard

The \_ wildcard represents a single character.

It can be any character or number, but each \_ represents one, and only one, character.

### Example

Return all customers with a City starting with any character, followed by "ondon":

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

### Example

Return all customers with a City starting with "L", followed by any 3 characters, ending with "on":

SELECT \* FROM Customers  
WHERE City LIKE 'L\_\_\_on';

## Using the [] Wildcard

The [] wildcard returns a result if any of the characters inside gets a match.

### Example

Return all customers starting with either "b", "s", or "p":

SELECT \* FROM Customers  
WHERE CustomerName LIKE '[bsp]%';

## Using the - Wildcard

The - wildcard allows you to specify a range of characters inside the [] wildcard.

### Example

Return all customers starting with "a", "b", "c", "d", "e" or "f":

SELECT \* FROM Customers  
WHERE CustomerName LIKE '[a-f]%';

**Combine Wildcards**

Any wildcard, like % and \_ , can be used in combination with other wildcards.

### Example

Return all customers that starts with "a" and are at least 3 characters in length:

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

### Example

Return all customers that have "r" in the second position:

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

## Without Wildcard

If no wildcard is specified, the phrase has to have an exact match to return a result.

### Example

Return all customers from Spain:

SELECT \* FROM Customers  
WHERE Country LIKE 'Spain';

## Microsoft Access Wildcards

The Microsoft Access Database has some other wildcards:

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Description** | **Example** |
| \* | Represents zero or more characters | bl\* finds bl, black, blue, and blob |
| ? | Represents a single character | h?t finds hot, hat, and hit |
| [] | Represents any single character within the brackets | h[oa]t finds hot and hat, but not hit |
| ! | Represents any character not in the brackets | h[!oa]t finds hit, but not hot and hat |
| - | Represents any single character within the specified range | c[a-b]t finds cat and cbt |
| # | Represents any single numeric character | 2#5 finds 205, 215, 225, 235, 245, 255, 265, 275, 285, and 295 |

# **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.

### Example

Return all customers from 'Germany', 'France', or 'UK'

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

## Syntax

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

## Demo Database

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

## NOT IN

## By using the NOT keyword in front of the IN operator, you return all records that are NOT any of the values in the list.

### Example

Return all customers that are NOT from 'Germany', 'France', or 'UK':

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

## IN (SELECT)

You can also use IN with a subquery in the WHERE clause.

With a subquery you can return all records from the main query that are present in the result of the subquery.

### Example

Return all customers that have an order in the [**Orders**](https://www.w3schools.com/sql/trysql.asp?filename=trysql_orders) table:

SELECT \* FROM Customers  
WHERE CustomerID IN (SELECT CustomerID FROM Orders);

## NOT IN (SELECT)

The result in the example above returned 74 records, that means that there are 17 customers that haven't placed any orders.

Let us check if that is correct, by using the NOT IN operator.

### Example

Return all customers that have NOT placed any orders in the [**Orders**](https://www.w3schools.com/sql/trysql.asp?filename=trysql_orders) table:

SELECT \* FROM Customers  
WHERE CustomerID NOT IN (SELECT CustomerID FROM Orders);

# **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.

### Example

Selects all products with a price between 10 and 20:

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

## Syntax

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

## 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 |

**NOT BETWEEN**

To display the products outside the range of the previous example, use NOT BETWEEN:

### Example

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

## BETWEEN with IN

The following SQL statement selects all products with a price between 10 and 20. In addition, the CategoryID must be either 1,2, or 3:

### Example

SELECT \* FROM Products  
WHERE Price BETWEEN 10 AND 20  
AND CategoryID IN (1,2,3);

## BETWEEN Text Values

The following SQL statement selects all products with a ProductName alphabetically between Carnarvon Tigers and Mozzarella di Giovanni:

### Example

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

The following SQL statement selects all products with a ProductName between Carnarvon Tigers and Chef Anton's Cajun Seasoning:

### Example

SELECT \* FROM Products  
WHERE ProductName BETWEEN "Carnarvon Tigers" AND "Chef Anton's Cajun Seasoning"  
ORDER BY ProductName;

**NOT BETWEEN Text Values**The following SQL statement selects all products with a ProductName not between Carnarvon Tigers and Mozzarella di Giovanni:

### Example

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

## BETWEEN Dates

The following SQL statement selects all orders with an OrderDate between '01-July-1996' and '31-July-1996':

### Example

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

**OR**:

### Example

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

## Sample Table

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10248 | 90 | 5 | 7/4/1996 | 3 |
| 10249 | 81 | 6 | 7/5/1996 | 1 |
| 10250 | 34 | 4 | 7/8/1996 | 2 |
| 10251 | 84 | 3 | 7/9/1996 | 1 |
| 10252 | 76 | 4 | 7/10/1996 | 2 |

## SQL Aliases

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the AS keyword.

### Example

SELECT CustomerID AS ID  
FROM Customers;

## AS is Optional

Actually, in most database languages, you can skip the AS keyword and get the same result:

### Example

SELECT CustomerID ID  
FROM Customers;

## Syntax

When alias is used on column:

SELECT column\_name AS alias\_name  
FROM table\_name;

When alias is used on table:

SELECT column\_name(s)  
FROM table\_name AS alias\_name;

## Demo Database

Below is a selection from the [**Customers**](https://www.w3schools.com/sql/trysql.asp?filename=trysql_customers) and [**Orders**](https://www.w3schools.com/sql/trysql.asp?filename=trysql_orders) tables used in the examples:

### Customers

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

### Orders

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10248 | 90 | 5 | 7/4/1996 | 3 |
| 10249 | 81 | 6 | 7/5/1996 | 1 |
| 10250 | 34 | 4 | 7/8/1996 | 2 |

**Alias for Columns**

The following SQL statement creates two aliases, one for the CustomerID column and one for the CustomerName column:

### Example

SELECT CustomerID AS ID, CustomerName AS Customer  
FROM Customers;

## Using Aliases With a Space Character

If you want your alias to contain one or more spaces, like "My Great Products", surround your alias with square brackets or double quotes.

### Example

Using [square brackets] for aliases with space characters:

SELECT ProductName AS [My Great Products]  
FROM Products;

### Example

Using "double quotes" for aliases with space characters:

SELECT ProductName AS "My Great Products"  
FROM Products;

**Note:** Some database systems allows both [] and "", and some only allows one of them.

## Concatenate Columns

The following SQL statement creates an alias named "Address" that combine four columns (Address, PostalCode, City and Country):

### Example

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

**Note:** To get the SQL statement above to work in MySQL use the following:

### MySQL Example

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

**Note:** To get the SQL statement above to work in Oracle use the following:

### Oracle Example

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

## Alias for Tables

The same rules applies when you want to use an alias for a table.

### Example

Refer to the Customers table as Persons instead:

SELECT \* FROM Customers AS Persons;

It might seem useless to use aliases on tables, but when you are using more than one table in your queries, it can make the SQL statements shorter.

The following SQL statement selects all the orders from the customer with CustomerID=4 (Around the Horn). We use the "Customers" and "Orders" tables, and give them the table aliases of "c" and "o" respectively (Here we use aliases to make the SQL shorter):

### Example

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;

The following SQL statement is the same as above, but without aliases:

### Example

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

**Aliases can be useful when**:

* There are more than one table involved in a query
* Functions are used in the query
* Column names are big or not very readable
* Two or more columns are combined together

**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 |

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

### Example:

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

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

**Top of Form**

### SQL Statement:

SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate

FROM Orders

INNER JOIN Customers

ON Orders.CustomerID=Customers.CustomerID;

Edit the SQL Statement, and click "Run SQL" to see the result.

Bottom of Form

Run SQL »

### Result:

and it will produce something like this:

|  |  |  |
| --- | --- | --- |
| **OrderID** | **CustomerName** | **OrderDate** |
| 10308 | Ana Trujillo Emparedados y helados | 9/18/1996 |
| 10365 | Antonio Moreno Taquería | 11/27/1996 |
| 10383 | Around the Horn | 12/16/1996 |
| 10355 | Around the Horn | 11/15/1996 |
| 10278 | Berglunds snabbköp | 8/12/1996 |

**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**

**INNER JOIN**

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

Let's look at a selection of the [**Products**](https://www.w3schools.com/sql/trysql.asp?filename=trysql_products) table:

|  |  |  |  |
| --- | --- | --- | --- |
| **ProductID** | **ProductName** | **CategoryID** | **Price** |
| 1 | Chais | 1 | 18 |
| 2 | Chang | 1 | 19 |
| 3 | Aniseed Syrup | 2 | 10 |

And a selection of the [**Categories**](https://www.w3schools.com/sql/trysql.asp?filename=trysql_categories) table:

|  |  |  |
| --- | --- | --- |
| **CategoryID** | **CategoryName** | **Description** |
| 1 | Beverages | Soft drinks, coffees, teas, beers, and ales |
| 2 | Condiments | Sweet and savory sauces, relishes, spreads, and seasonings |
| 3 | Confections | Desserts, candies, and sweet breads |

We will join the Products table with the Categories table, by using the CategoryID field from both tables:

### Example:

Join Products and Categories with the INNER JOIN keyword:

SELECT ProductID, ProductName, CategoryName  
FROM Products  
INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID;

**Top of Form**

### SQL Statement:

SELECT ProductID, ProductName, CategoryName

FROM Products

INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID;

​Edit the SQL Statement, and click "Run SQL" to see the result.

**Bottom of Form**

### Result:



**Note:** The INNER JOIN keyword returns only rows with a match in both tables. Which means that if you have a product with no CategoryID, or with a CategoryID that is not present in the Categories table, that record would not be returned in the result.

**Syntax**

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

## Naming the Columns

It is a good practice to include the table name when specifying columns in the SQL statement.

### Example

**Specify the table names**:

SELECT Products.ProductID, Products.ProductName, Categories.CategoryName  
FROM Products  
INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID;

The example above works without specifying table names, because none of the specified column names are present in both tables. If you try to include CategoryID in the SELECT statement, you will get an error if you do not specify the table name (because CategoryID is present in both tables).

**JOIN or INNER JOIN**

JOIN and INNER JOIN will return the same result.

INNER is the default join type for JOIN, so when you write JOIN the parser actually writes INNER JOIN.

### Example

JOIN is the same as INNER JOIN:

SELECT Products.ProductID, Products.ProductName, Categories.CategoryName  
FROM Products  
JOIN Categories ON Products.CategoryID = Categories.CategoryID;

**Top of Form**

### SQL Statement:

SELECT Products.ProductID, Products.ProductName, Categories.CategoryName

FROM Products

JOIN Categories ON Products.CategoryID = Categories.CategoryID;

​

Number of Records: 16

|  |  |  |
| --- | --- | --- |
| **ProductID** | **ProductName** | **CategoryName** |
| 39 | Chartreuse verte | Beverages |
| 2 | Chang | Beverages |
| 24 | Guaraná Fantástica | Beverages |
| 34 | Sasquatch Ale | Beverages |
| 35 | Steeleye Stout | Beverages |
| 1 | Chais | Beverages |
| 38 | Côte de Blaye | Beverages |
| 43 | Ipoh Coffee | Beverages |
| 67 | Laughing Lumberjack Lager | Beverages |
| 70 | Outback Lager | Beverages |
| 75 | Rhönbräu Klosterbier | Beverages |
| 76 | Lakkalikööri | Beverages |
| 15 | Genen Shouyu | Condiments |
| 8 | Northwoods Cranberry Sauce | Condiments |
| 77 | Original Frankfurter grüne Soße | Condiments |
| 6 | Grandma's Boysenberry Spread | Condiments |
| 44 | Gula Malacca | Condiments |
| 5 | Chef Anton's Gumbo Mix | Condiments |

### Result:

## JOIN Three Tables

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

### Example

|  |  |  |
| --- | --- | --- |
| **OrderID** | **CustomerName** | **ShipperName** |
| 10290 | Comércio Mineiro | Speedy Express |
| 10284 | Lehmanns Marktstand | Speedy Express |
| 10388 | Seven Seas Imports | Speedy Express |
| 10390 | Ernst Handel | Speedy Express |
| 10296 | LILA-Supermercado | Speedy Express |
| 10349 | Split Rail Beer & Ale | Speedy Express |
| 10309 | Hungry Owl All-Night Grocers | Speedy Express |
| 10401 | Rattlesnake Canyon Grocery | Speedy Express |
| 10379 | Que Delícia | Speedy Express |
| 10288 | Reggiani Caseifici | Speedy Express |
| 10404 | Magazzini Alimentari Riuniti | Speedy Express |
| 10405 | LINO-Delicateses | Speedy Express |
| 10406 | Queen Cozinha | Speedy Express |
| 10408 | Folies gourmandes | Speedy Express |

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);

**Top of Form**

**SQL LEFT JOIN Keyword**

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records 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;

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



## Demo Database

We will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

And a selection from the "Orders" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

**SQL LEFT JOIN Example:**

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

### Example:

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

**Top of Form**

### SQL Statement:

SELECT Customers.CustomerName, Orders.OrderID

FROM Customers

LEFT JOIN Orders

ON Customers.CustomerID=Orders.CustomerID

ORDER BY Customers.CustomerName;

|  |  |
| --- | --- |
| **CustomerName** | **OrderID** |
| Alfreds Futterkiste |  |
| Ana Trujillo Emparedados y helados | 10308 |
| Antonio Moreno Taquería | 10365 |
| Around the Horn | 10383 |
| Around the Horn | 10355 |
| Berglunds snabbköp | 10278 |
| Berglunds snabbköp | 10280 |
| Berglunds snabbköp | 10384 |
| Blauer See Delikatessen |  |
| Blondel père et fils | 10360 |
| Blondel père et fils | 10297 |
| Blondel père et fils | 10436 |

​

Number of Records: 213

### Result:SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if 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.



## Demo Database

We will use the well-known Northwind sample database.

Below is a selection from the "Orders" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10308 | 2 | 7 | 1996-09-18 | 3 |

And a selection from the "Employees" table:

## SQL RIGHT JOIN Example

The following SQL statement will return all employees, and any orders they might have placed:

### Example:

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

# **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.

**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;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

And a selection from the "Orders" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

**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;

A selection from the result set may look like this:

|  |  |
| --- | --- |
| **CustomerName** | **OrderID** |
| *Null* | 10309 |
| *Null* | 10310 |
| Alfreds Futterkiste | *Null* |
| Ana Trujillo Emparedados y helados | 10308 |
| Antonio Moreno Taquería | *Null* |

**Note:** The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

# **SQL Self Join**

A self join is a regular join, but the table is joined with itself.

### Self Join Syntax

SELECT column\_name(s)  
FROM table1 T1, table1 T2  
WHERE condition;

T1 and T2 are different table aliases for the same table.

## Demo Database

We will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

## SQL Self Join Example

The following SQL statement matches customers that are from the same city:

### Example:

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;

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;

# **SQL UNION Operator**

**The SQL UNION Operator**

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

* Every SELECT statement within UNION must have the same number of columns
* The columns must also have similar data types
* The columns in every SELECT statement must also be in the same order

### UNION Syntax

SELECT column\_name(s) FROM table1  
UNION  
SELECT column\_name(s) FROM table2;

### UNION ALL Syntax

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;

### UNION ALL Syntax

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

SELECT column\_name(s) FROM table2;

**Note:** The column names in the result-set are usually equal to the column names in the first SELECT statement.

## Demo Database

We will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

And a selection from the "Suppliers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SupplierID** | **SupplierName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Exotic Liquid | Charlotte Cooper | 49 Gilbert St. | London | EC1 4SD | UK |
| 2 | New Orleans Cajun Delights | Shelley Burke | P.O. Box 78934 | New Orleans | 70117 | USA |
| 3 | Grandma Kelly's Homestead | Regina Murphy | 707 Oxford Rd. | Ann Arbor | 48104 | USA |

## SQL UNION With WHERE

The following SQL statement returns the German cities (only distinct values) from both the "Customers" and the "Suppliers" table:

### Example

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

**Top of Form**

### SQL Statement:

SELECT City, Country FROM Customers

WHERE Country='Germany'

UNION

SELECT City, Country FROM Suppliers

WHERE Country='Germany'

ORDER BY City;

### Result:

Number of Records: 13

|  |  |
| --- | --- |
| **City** | **Country** |
| Aachen | Germany |
| Berlin | Germany |
| Brandenburg | Germany |
| Cunewalde | Germany |
| Cuxhaven | Germany |
| Frankfurt | Germany |
| Frankfurt a.M. | Germany |
| Köln | Germany |
| Leipzig | Germany |
| Mannheim | Germany |
| München | Germany |
| Münster | Germany |

# **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);

## Demo Database

Below is a selection from the "Customers" table in the Northwind sample database:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

**SQL GROUP BY Examples**

The following SQL statement lists the number of customers in each country:

### Example:

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

**Top of Form**

### SQL Statement:

SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country;

​

|  |  |
| --- | --- |
| **Expr1000** | **Country** |
| 3 | Argentina |
| 2 | Austria |
| 2 | Belgium |
| 9 | Brazil |
| 3 | Canada |
| 2 | Denmark |
| 2 | Finland |
| 11 | France |
| 11 | Germany |
| 1 | Ireland |
| 3 | Italy |
| 5 | Mexico |
| 1 | Norway |

Number of Records: 21

### Result:

### Example

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

**Top of Form**

### SQL Statement:

SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country

ORDER BY COUNT(CustomerID) DESC;

​**Result**:

## Demo Database

Below is a selection from the "Orders" table in the Northwind sample database:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10248 | 90 | 5 | 1996-07-04 | 3 |
| 10249 | 81 | 6 | 1996-07-05 | 1 |
| 10250 | 34 | 4 | 1996-07-08 | 2 |

And a selection from the "Shippers" table:

|  |  |
| --- | --- |
| **ShipperID** | **ShipperName** |
| 1 | Speedy Express |
| 2 | United Package |
| 3 | Federal Shipping |

## GROUP BY With JOIN Example

The following SQL statement lists the number of orders sent by each shipper:

### Example

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

**Top of Form**

### SQL Statement:

SELECT Shippers.ShipperName,COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders

LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID

GROUP BY ShipperName;

​**Result**:

Number of Records: 3

|  |  |
| --- | --- |
| **ShipperName** | **NumberOfOrders** |
| Federal Shipping | 68 |
| Speedy Express | 54 |
| United Package | 74 |

# **SQL HAVING Clause:**

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

### HAVING Syntax:

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

**Below is a selection from the "Customers" table in the Northwind sample database**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

## SQL HAVING Examples:

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

### Example:

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

**Top of Form**

### SQL Statement:

SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country

HAVING COUNT(CustomerID) > 5;

​**Result**:

### Example:

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

**The SQL HAVING Clause**

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

### HAVING Syntax

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

## Demo Database

Below is a selection from the "Customers" table in the Northwind sample database:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

## SQL HAVING Examples

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

### Example:

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

The following SQL statement lists the number of customers in each country, sorted high to low (Only include countries with more than 5 customers):

### Example:

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

**Top of Form**

### SQL Statement:

SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country

HAVING COUNT(CustomerID) > 5

ORDER BY COUNT(CustomerID) DESC;

​**Result**:

## Demo Database

Below is a selection from the "Orders" table in the Northwind sample database:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10248 | 90 | 5 | 1996-07-04 | 3 |
| 10249 | 81 | 6 | 1996-07-05 | 1 |
| 10250 | 34 | 4 | 1996-07-08 | 2 |

And a selection from the "Employees" table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EmployeeID** | **LastName** | **FirstName** | **BirthDate** | **Photo** | **Notes** |
| 1 | Davolio | Nancy | 1968-12-08 | EmpID1.pic | Education includes a BA.... |
| 2 | Fuller | Andrew | 1952-02-19 | EmpID2.pic | Andrew received his BTS.... |
| 3 | Leverling | Janet | 1963-08-30 | EmpID3.pic | Janet has a BS degree.... |

## More HAVING Examples

The following SQL statement lists the employees that have registered more than 10 orders:

### Example

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;

**Top of Form**

### SQL Statement:

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;

**Result**:

### Example

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;

**Top of Form**

### SQL Statement:

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;

​Result:

# **SQL EXISTS Operator**

**The SQL EXISTS Operator**

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

### EXISTS Syntax

SELECT column\_name(s)  
FROM table\_name  
WHERE EXISTS  
(SELECT column\_name FROM table\_name WHERE condition

## Demo Database

Below is a selection from the "Products" table in the Northwind sample database:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 |

And a selection from the "Suppliers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SupplierID** | **SupplierName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Exotic Liquid | Charlotte Cooper | 49 Gilbert St. | London | EC1 4SD | UK |
| 2 | New Orleans Cajun Delights | Shelley Burke | P.O. Box 78934 | New Orleans | 70117 | USA |
| 3 | Grandma Kelly's Homestead | Regina Murphy | 707 Oxford Rd. | Ann Arbor | 48104 | USA |
| 4 | Tokyo Traders | Yoshi Nagase | 9-8 Sekimai Musashino-shi | Tokyo | 100 | Japan |

## SQL EXISTS Examples

The following SQL statement returns TRUE and lists the suppliers with a product price less than 20:

### Example:

SELECT SupplierName  
FROM Suppliers  
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price < 20);

## SQL EXISTS Examples

The following SQL statement returns TRUE and lists the suppliers with a product price less than 20:

### Example:

SELECT SupplierName  
FROM Suppliers  
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.su

### Example

SELECT SupplierName  
FROM Suppliers  
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price = 22);

## GROUP BY With JOIN Example

The following SQL statement lists the number of orders sent by each shipper:

### Example

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

**Top of Form**

### SQL Statement:

SELECT Shippers.ShipperName,COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders

LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID

GROUP BY ShipperName;

**The SQL HAVING Clause**

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

### HAVING Syntax

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

## Demo Database

Below is a selection from the "Customers" table in the Northwind sample database:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

## SQL HAVING Examples

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

### Example:

### Result:

Number of Records: 5

|  |  |
| --- | --- |
| **Expr1000** | **Country** |
| 9 | Brazil |
| 11 | France |
| 11 | Germany |
| 7 | UK |
| 13 | USA |

### Example:

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

**Top of Form**

### SQL Statement:

SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country

HAVING COUNT(CustomerID) > 5;

​Result:

Number of Records: 5

|  |  |
| --- | --- |
| **Expr1000** | **Country** |
| 13 | USA |
| 11 | Germany |
| 11 | France |
| 9 | Brazil |
| 7 | UK |

## Demo Database

Below is a selection from the "Orders" table in the Northwind sample database:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10248 | 90 | 5 | 1996-07-04 | 3 |
| 10249 | 81 | 6 | 1996-07-05 | 1 |
| 10250 | 34 | 4 | 1996-07-08 | 2 |

And a selection from the "Employees" table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EmployeeID** | **LastName** | **FirstName** | **BirthDate** | **Photo** | **Notes** |
| 1 | Davolio | Nancy | 1968-12-08 | EmpID1.pic | Education includes a BA.... |
| 2 | Fuller | Andrew | 1952-02-19 | EmpID2.pic | Andrew received his BTS.... |
| 3 | Leverling | Janet | 1963-08-30 | EmpID3.pic | Janet has a BS degree.... |

## More HAVING Examples

The following SQL statement lists the employees that have registered more than 10 orders:

### Example

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;

Result:

Number of Records: 8

|  |  |
| --- | --- |
| **LastName** | **NumberOfOrders** |
| Buchanan | 11 |
| Callahan | 27 |
| Davolio | 29 |
| Fuller | 20 |
| King | 14 |
| Leverling | 31 |
| Peacock | 40 |
| Suyama | 18 |

### Example

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;

**Result**:

Number of Records: 1

|  |  |
| --- | --- |
| **LastName** | **NumberOfOrders** |
| Davolio | 29 |

Top of Form

# **SQL EXISTS OPERATOR:**

**The SQL EXISTS Operator**

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

### EXISTS Syntax

SELECT column\_name(s)  
FROM table\_name  
WHERE EXISTS  
(SELECT column\_name FROM table\_name WHERE condition);

## Demo Database

Below is a selection from the "Products" table in the Northwind sample database:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 |

And a selection from the "Suppliers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SupplierID** | **SupplierName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Exotic Liquid | Charlotte Cooper | 49 Gilbert St. | London | EC1 4SD | UK |
| 2 | New Orleans Cajun Delights | Shelley Burke | P.O. Box 78934 | New Orleans | 70117 | USA |
| 3 | Grandma Kelly's Homestead | Regina Murphy | 707 Oxford Rd. | Ann Arbor | 48104 | USA |
| 4 | Tokyo Traders | Yoshi Nagase | 9-8 Sekimai Musashino-shi | Tokyo | 100 | Japan |

## SQL EXISTS Examples

The following SQL statement returns TRUE and lists the suppliers with a product price less than 20:

### Example:

SELECT SupplierName  
FROM Suppliers  
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price < 20);

**Result**:

Number of Records: 24

|  |
| --- |
| **SupplierName** |
| Exotic Liquid |
| New Orleans Cajun Delights |
| Tokyo Traders |
| Mayumi's |
| Pavlova, Ltd. |
| Specialty Biscuits, Ltd. |
| PB Knäckebröd AB |
| Refrescos Americanas LTDA |
| Heli Süßwaren GmbH & Co. KG |
| Plutzer Lebensmittelgroßmärkte AG |
| Formaggi Fortini s.r.l. |
| Norske Meierier |
| Bigfoot Breweries |
| Svensk Sjöföda AB |
| Aux joyeux ecclésiastiques |

The following SQL statement returns TRUE and lists the suppliers with a product price equal to 22:

### Example:

SELECT SupplierName  
FROM Suppliers  
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price = 22);

Result:

Number of Records: 1

|  |
| --- |
| **SupplierName** |
| New Orleans Cajun Delights |

# **SQL ANY and ALL Operators:**

# **The SQL ANY and ALL Operators**

The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

## The SQL ANY Operator

**The ANY operator**:

* returns a boolean value as a result
* returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

### ANY Syntax

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name operator ANY  
  (SELECT column\_name  FROM table\_name  WHERE condition);

**Note:** The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

**The SQL ALL Operator**

The ALL operator:

* returns a boolean value as a result
* returns TRUE if ALL of the subquery values meet the condition
* is used with SELECT, WHERE and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

### ALL Syntax With SELECT

SELECT ALL column\_name(s)  
FROM table\_name  
WHERE condition;

### ALL Syntax With WHERE or HAVING

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name operator ALL  
  (SELECT column\_name  FROM table\_name  WHERE condition);

**Note:** The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

**Demo Database**

Below is a selection from the **"Products"** table in the Northwind sample database:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 |
| 6 | Grandma's Boysenberry Spread | 3 | 2 | 12 - 8 oz jars | 25 |
| 7 | Uncle Bob's Organic Dried Pears | 3 | 7 | 12 - 1 lb pkgs. | 30 |
| 8 | Northwoods Cranberry Sauce | 3 | 2 | 12 - 12 oz jars | 40 |
| 9 | Mishi Kobe Niku | 4 | 6 | 18 - 500 g pkgs. | 97 |

And a selection from the **"OrderDetails"** table:

|  |  |  |  |
| --- | --- | --- | --- |
| **OrderDetailID** | **OrderID** | **ProductID** | **Quantity** |
| 1 | 10248 | 11 | 12 |
| 2 | 10248 | 42 | 10 |
| 3 | 10248 | 72 | 5 |
| 4 | 10249 | 14 | 9 |
| 5 | 10249 | 51 | 40 |
| 6 | 10250 | 41 | 10 |
| 7 | 10250 | 51 | 35 |
| 8 | 10250 | 65 | 15 |
| 9 | 10251 | 22 | 6 |
| 10 | 10251 | 57 | 15 |

**SQL ANY Example**

The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity equal to 10 (this will return TRUE because the Quantity column has some values of 10):

### Example:

SELECT ProductName  
FROM Products  
WHERE ProductID = ANY  
  (SELECT ProductID  
  FROM OrderDetails  
  WHERE Quantity = 10);

**Result**:

Number of Records: 31

|  |
| --- |
| **ProductName** |
| Chais |
| Chang |
| Chef Anton's Cajun Seasoning |
| Uncle Bob's Organic Dried Pears |
| Konbu |
| Tofu |
| Pavlova |
| Teatime Chocolate Biscuits |
| Sir Rodney's Scones |
| Guaraná Fantástica |
| NuNuCa Nuß-Nougat-Creme |
| Gumbär Gummibärchen |
| Thüringer Rostbratwurst |
| Nord-Ost Matjeshering |
| Sasquatch Ale |
| Steeleye Stout |
| Gravad lax |
| Côte de Blaye |

The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity larger than 99 (this will return TRUE because the Quantity column has some values larger than 99):

### Example

SELECT ProductName  
FROM Products  
WHERE ProductID = ANY  
  (SELECT ProductID  
  FROM OrderDetails  
  WHERE Quantity > 99);

Number of Records: 2

|  |
| --- |
| **ProductName** |
| Steeleye Stout |
| Pâté chinois |

The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity larger than 1000 (this will return FALSE because the Quantity column has no values larger than 1000):

### Example

SELECT ProductName  
FROM Products  
WHERE ProductID = ANY  
  (SELECT ProductID  
  FROM OrderDetails  
  WHERE Quantity > 1000);

## The SQL SELECT INTO Statement

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

### SELECT INTO Syntax

Copy all columns into a new table:

SELECT \*  
INTO newtable [IN externaldb]  
FROM oldtableWHERE condition;

Copy only some columns into a new table:

SELECT column1, column2, column3, ...  
INTO newtable [IN externaldb]  
FROM oldtableWHERE condition;

The new table will be created with the column-names and types as defined in the old table. You can create new column names using the AS clause.

## SQL SELECT INTO Examples

The following SQL statement creates a backup copy of Customers:

SELECT \* INTO CustomersBackup2017  
FROM Customers;

The following SQL statement uses the IN clause to copy the table into a new table in another database:

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

The following SQL statement copies only a few columns into a new table:

SELECT CustomerName, ContactName INTO CustomersBackup2017  
FROM Customers;

The following SQL statement copies only the German customers into a new table:

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

The following SQL statement copies data from more than one table into a new table:

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

**Tip:** SELECT INTO can also be used to create a new, empty table using the schema of another. Just add a WHERE clause that causes the query to return no data:

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

## The SQL INSERT INTO SELECT Statement

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.

The INSERT INTO SELECT statement requires that the data types in source and target tables match.

**Note:** The existing records in the target table are unaffected.

### INSERT INTO SELECT Syntax

Copy all columns from one table to another table:

INSERT INTO table2  
SELECT \* FROM table1WHERE condition;

Copy only some columns from one table into another table:

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

## Demo Database

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

We will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

And a selection from the "Suppliers" table:

## SQL INSERT INTO SELECT Examples

### Example

Copy "Suppliers" into "Customers" (the columns that are not filled with data, will contain NULL):

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

And a selection from the "Suppliers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SupplierID** | **SupplierName** | **ContactName** | **Address** | **City** | **Postal Code** | **Country** |
| 1 | Exotic Liquid | Charlotte Cooper | 49 Gilbert St. | Londona | EC1 4SD | UK |
| 2 | New Orleans Cajun Delights | Shelley Burke | P.O. Box 78934 | New Orleans | 70117 | USA |
| 3 | Grandma Kelly's Homestead | Regina Murphy | 707 Oxford Rd. | Ann Arbor | 48104 | USA |

## SQL INSERT INTO SELECT Examples

### Example

Copy "Suppliers" into "Customers" (the columns that are not filled with data, will contain NULL):

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

### Example

Copy "Suppliers" into "Customers" (fill all columns):

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

### Example

Copy only the German suppliers into "Customers":

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

## The SQL CASE Expression

The CASE expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

## CASE Syntax

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

## Demo Database

Below is a selection from the "OrderDetails" table in the Northwind sample database:

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

## SQL CASE Examples

The following SQL goes through conditions and returns a value when the first condition is met:

### Example

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;

### Result:

Number of Records: 14

|  |  |  |
| --- | --- | --- |
| **OrderID** | **Quantity** | **QuantityText** |
| 10248 | 12 | The quantity is under 30 |
| 10248 | 10 | The quantity is under 30 |
| 10248 | 5 | The quantity is under 30 |
| 10249 | 9 | The quantity is under 30 |
| 10249 | 40 | The quantity is greater than 30 |
| 10250 | 10 | The quantity is under 30 |
| 10250 | 35 | The quantity is greater than 30 |
| 10250 | 15 | The quantity is under 30 |
| 10251 | 6 | The quantity is under 30 |
| 10251 | 15 | The quantity is under 30 |
| 10251 | 20 | The quantity is under 30 |
| 10252 | 40 | The quantity is greater than 30 |
| 10252 | 25 | The quantity is under 30 |
| 10252 | 40 | The quantity is greater than 30 |
|  |  |  |

The following SQL will order the customers by City. However, if City is NULL, then order by Country:

### Example

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

### Result:

|  |  |  |
| --- | --- | --- |
| **CustomerName** | **City** | **Country** |
| Drachenblut Delikatessend | Aachen | Germany |
| Rattlesnake Canyon Grocery | Albuquerque | USA |
| Old World Delicatessen | Anchorage | USA |
| Vaffeljernet | Århus | Denmark |
| Galería del gastrónomo | Barcelona | Spain |
| LILA-Supermercado | Barquisimeto | Venezuela |
| Magazzini Alimentari Riuniti | Bergamo | Italy |
| Alfreds Futterkiste | Berlin | Germany |
| Chop-suey Chinese | Bern | Switzerland |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **ProductName** | **UnitPrice** | **UnitsInStock** | **UnitsOnOrder** |
| 1 | Jarlsberg | 10.45 | 16 | 15 |
| 2 | Mascarpone | 32.56 | 23 |  |
| 3 | Gorgonzola | 15.67 | 9 | 20 |

Number of Records: 9

## SQL IFNULL(), ISNULL(), COALESCE(), and NVL() Functions

Look at the following "Products" table:

Suppose that the "UnitsOnOrder" column is optional, and may contain NULL values.

Look at the following SELECT statement:

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

In the example above, if any of the "UnitsOnOrder" values are NULL, the result will be NULL.

## Solutions

**MySQL**

The MySQL [IFNULL()](https://www.w3schools.com/sql/func_mysql_ifnull.asp) function 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()](https://www.w3schools.com/sql/func_mysql_coalesce.asp) function, like this:

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

**SQL Server**

The SQL Server [ISNULL()](https://www.w3schools.com/sql/func_sqlserver_isnull.asp) function lets you return an alternative value when an expression is NULL:

SELECT ProductName, UnitPrice \* (UnitsInStock + ISNULL(UnitsOnOrder, 0))  
FROM Products;

or we can use the [COALESCE()](https://www.w3schools.com/sql/func_sqlserver_coalesce.asp) function, like this:

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

**MS Access**

The MS Access [IsNull()](https://www.w3schools.com/sql/func_msaccess_isnull.asp) 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;

**Oracle**

The Oracle NVL() function achieves the same result:

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

or we can use the COALESCE() function, like this:

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

## What is a Stored Procedure?

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

### Stored Procedure Syntax

CREATE PROCEDURE procedure\_name  
AS  
sql\_statement  
GO;

### Execute a Stored Procedure

EXEC procedure\_name;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

**Demo Database**

Below is a selection from the "Customers" table in the Northwind sample database:

## Stored Procedure Example

The following SQL statement creates a stored procedure named "SelectAllCustomers" that selects all records from the "Customers" table:

### Example

CREATE PROCEDURE SelectAllCustomers  
AS  
SELECT \* FROM Customers  
GO;

Execute the stored procedure above as follows:

### Example

EXEC SelectAllCustomers;

## Stored Procedure With One Parameter

The following SQL statement creates a stored procedure that selects Customers from a particular City from the "Customers" table:

### Example

CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)  
AS  
SELECT \* FROM Customers WHERE City = @City  
GO;

Execute the stored procedure above as follows:

### Example

EXEC SelectAllCustomers @City = 'London';

## Stored Procedure With Multiple Parameters

Setting up multiple parameters is very easy. Just list each parameter and the data type separated by a comma as shown below.

The following SQL statement creates a stored procedure that selects Customers from a particular City with a particular PostalCode from the "Customers" table:

### Example

CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode nvarchar(10)  
AS  
SELECT \* FROM Customers WHERE City = @City AND PostalCode = @PostalCode  
GO;

Execute the stored procedure above as follows:

### Example

EXEC SelectAllCustomers @City = 'London', @PostalCode = 'WA1 1DP';

## SQL Comments

Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.

**Note: Comments are not supported in Microsoft Access databases!**

## Single Line Comments

Single line comments start with --.

Any text between -- and the end of the line will be ignored (will not be executed).

The following example uses a single-line comment as an explanation:

### Example

--Select all:  
SELECT \* FROM Customers;

### Result:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 8 | Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |
| 9 | Bon app' | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |
| 10 | Bottom-Dollar Marketse | Elizabeth Lincoln | 23 Tsawassen Blvd. | Tsawassen | T2F 8M4 | Canada |

Number of Records: 10

The following example uses a single-line comment to ignore the end of a line:

### Example

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

RESULT

Number of Records: 91

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 8 | Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |
| 9 | Bon app' | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |
| 10 | Bottom-Dollar Marketse | Elizabeth Lincoln | 23 Tsawassen Blvd. | Tsawassen | T2F 8M4 | Canada |

The following example uses a single-line comment to ignore a statement:

### Example

--SELECT \* FROM Customers;  
SELECT \* FROM Products;

**RESULT**

### Error in SQL:

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '--SELECT \* FROM Customers; SELECT \* FROM Products' at line 1

## SQL Arithmetic Operators Add

SELECT 30 + 20;

### Result:

Number of Records: 1

|  |
| --- |
| **Expr1000** |
| 50 |

**Subtract**

SELECT 30 - 20;

|  |
| --- |
| **Expr1000** |
| 10 |

Number of Records: 1

**Multiply**

SELECT 30 \* 20;

Number of Records: 1

|  |
| --- |
| **Expr1000** |
| 600 |

**Divide**

SELECT 30 / 10;

Number of Records: 1

|  |
| --- |
| **Expr1000** |
| 3 |

**Modulo**

SELECT 17 % 5;

Number of Records: 1

|  |
| --- |
| **17 % 5** |
| |  |  | | --- | --- | | **Operator** | **Description** | | += | Add equals | | -= | Subtract equals | | \*= | Multiply equals | | /= | Divide equals | | %= | Modulo equals | | &= | Bitwise AND equals | | ^-= | Bitwise exclusive equals | | |\*= | Bitwise OR equals |   2 |

## SQL Bitwise Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise exclusive OR |

## SQL Compound Operators

## SQL Logical Operators

ALL

TRUE if all the subquery values meet the condition

### SQL Statement:

SELECT ProductName

FROM Products

WHERE ProductID = ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

### ​Result:

|  |
| --- |
| **ProductName** |

Number of Records: 0

**AND**

TRUE if all the conditions separated by AND is TRUE

SELECT \* FROM Customers

WHERE City = "London" AND Country = "UK";

### Result:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 11 | B's Beverages | Victoria Ashworth | Fauntleroy Circus | London | EC2 5NT | UK |
| 16 | Consolidated Holdings | Elizabeth Brown | Berkeley Gardens 12 Brewery | London | WX1 6LT | UK |
| 19 | Eastern Connection | Ann Devon | 35 King George | London | WX3 6FW | UK |
| 53 | North/South | Simon Crowther | South House 300 Queensbridge | London | SW7 1RZ | UK |
| 72 | Seven Seas Imports | Hari Kumar | 90 Wadhurst Rd. | London | OX15 4NB | UK |

Number of Records: 6

### SQL Statement:

SELECT \* FROM Products

WHERE Price > ANY (SELECT Price FROM Products WHERE Price > 50);

​

### Result:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 9 | Mishi Kobe Niku | 4 | 6 | 18 - 500 g pkgs. | 97 |
| 18 | Carnarvon Tigers | 7 | 8 | 16 kg pkg. | 62.5 |
| 20 | Sir Rodney's Marmalade | 8 | 3 | 30 gift boxes | 81 |
| 29 | Thüringer Rostbratwurst | 12 | 6 | 50 bags x 30 sausgs. | 123.79 |
| 38 | Côte de Blaye | 18 | 1 | 12 - 75 cl bottles | 263.5 |
| 59 | Raclette Courdavault | 28 | 4 | 5 kg pkg. | 55 |

Number of Records: 6

**BETWEEN** TRUE if the operand is within the range of comparisons

SELECT \* FROM Products

WHERE Price BETWEEN 50 AND 60;

### Result:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 51 | Manjimup Dried Apples | 24 | 7 | 50 - 300 g pkgs. | 53 |
| 59 | Raclette Courdavault | 28 | 4 | 5 kg pkg. | 55 |

Number of Records: 2

**EXISTS** TRUE if the subquery returns one or more records

### SQL Statement:

SELECT SupplierName

FROM Suppliers

WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price < 20);

​

### Result:

Number of Records: 24

|  |
| --- |
| **SupplierName** |
| Exotic Liquid |
| New Orleans Cajun Delights |
| Tokyo Traders |
| Mayumi's |
| Pavlova, Ltd. |
| Specialty Biscuits, Ltd. |
| PB Knäckebröd AB |
| Refrescos Americanas LTDA |
| Heli Süßwaren GmbH & Co. KG |
| Plutzer Lebensmittelgroßmärkte AG |
| Formaggi Fortini s.r.l. |
| Norske Meierier |
| Bigfoot Breweries |
| Svensk Sjöföda AB |
| Aux joyeux ecclésiastiques |
| New England Seafood Cannery |
| Leka Trading |
| Lyngbysild |
| Zaanse Snoepfabriek |
| Karkki Oy |
| G'day, Mate |
| Ma Maison |
| Pasta Buttini s.r.l. |
| Escargots Nouveaux |

**IN** TRUE if the operand is equal to one of a list of expressions

### SQL Statement:

SELECT \* FROM Customers

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

### Result:

Number of Records: 8

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 11 | B's Beverages | Victoria Ashworth | Fauntleroy Circus | London | EC2 5NT | UK |
| 16 | Consolidated Holdings | Elizabeth Brown | Berkeley Gardens 12 Brewery | London | WX1 6LT | UK |
| 19 | Eastern Connection | Ann Devon | 35 King George | London | WX3 6FW | UK |
| 53 | North/South | Simon Crowther | South House 300 Queensbridge | London | SW7 1RZ | UK |
| 57 | Paris spécialités | Marie Bertrand | 265, boulevard Charonne | Paris | 75012 | France |
| 72 | Seven Seas Imports | Hari Kumar | 90 Wadhurst Rd. | London | OX15 4NB | UK |
| 74 | Spécialités du monde | Dominique Perrier | 25, rue Lauriston | Paris | 75016 | France |

**LIKE** TRUE if the operand matches a pattern

### SQL Statement:

SELECT \* FROM Customers

WHERE City LIKE 's%';

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 7 | Blondel père et fils | Frédérique Citeaux | 24, place Kléber | Strasbourg | 67000 | France |
| 15 | Comércio Mineiro | Pedro Afonso | Av. dos Lusíadas, 23 | São Paulo | 05432-043 | Brazil |
| 21 | Familia Arquibaldo | Aria Cruz | Rua Orós, 92 | São Paulo | 05442-030 | Brazil |
| 30 | Godos Cocina Típica | José Pedro Freyre | C/ Romero, 33 | Sevilla | 41101 | Spain |
| 35 | HILARIÓN-Abastos | Carlos Hernández | Carrera 22 con Ave. Carlos Soublette #8-35 | San Cristóbal | 5022 | Venezuela |
| 45 | Let's Stop N Shop | Jaime Yorres | 87 Polk St. Suite 5 | San Francisco | 94117 | USA |
| 59 | Piccolo und mehr | Georg Pipps | Geislweg 14 | Salzburg | 5020 | Austria |
| 62 | Queen Cozinha | Lúcia Carvalho | Alameda dos Canàrios, 891 | São Paulo | 05487-020 | Brazil |
| 70 | Santé Gourmet | Jonas Bergulfsen | Erling Skakkes gate 78 | Stavern | 4110 | Norway |
| 81 | Tradição Hipermercados | Anabela Domingues | Av. Inês de Castro, 414 | São Paulo | 05634-030 | Brazil |
| 86 | Die Wandernde Kuh | Rita Müller | Adenauerallee 900 | Stuttgart | 70563 | Germany |
| 89 | White Clover Markets | Karl Jablonski | 305 - 14th Ave. S. Suite 3B | Seattle | 98128 | USA |

### Result:

Number of Records: 12

​**NOT** Displays a record if the condition(s) is NOT TRUE

SELECT \* FROM Customers

WHERE City NOT LIKE 's%';

### Result:

Number of Records: 10

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |
| 6 | Blauer See Delikatessen | Hanna Moos | Forsterstr. 57 | Mannheim | 68306 | Germany |
| 8 | Bólido Comidas preparadas | Martín Sommer | C/ Araquil, 67 | Madrid | 28023 | Spain |
| 9 | Bon app' | Laurence Lebihans | 12, rue des Bouchers | Marseille | 13008 | France |
| 10 | Bottom-Dollar Marketse | Elizabeth Lincoln | 23 Tsawassen Blvd. | Tsawassen | T2F 8M4 | Canada |

### OR TRUE if any of the conditions separated by OR is TRUE

### SQL Statement:

SELECT \* FROM Customers

WHERE City = "London" OR Country = "UK";

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 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 |
| 6 | Grandma's Boysenberry Spread | 3 | 2 | 12 - 8 oz jars | 25 |
| 7 | Uncle Bob's Organic Dried Pears | 3 | 7 | 12 - 1 lb pkgs. | 30 |
| 8 | Northwoods Cranberry Sauce | 3 | 2 | 12 - 12 oz jars | 40 |
| 9 | Mishi Kobe Niku | 4 | 6 | 18 - 500 g pkgs. | 97 |
| 10 | Ikura | 4 | 8 | 12 - 200 ml jars | 31 |
| 12 | Queso Manchego La Pastora | 5 | 4 | 10 - 500 g pkgs. | 38 |
| 14 | Tofu | 6 | 7 | 40 - 100 g pkgs. | 23.25 |
| 17 | Alice Mutton | 7 | 6 | 20 - 1 kg tins | 39 |
| 18 | Carnarvon Tigers | 7 | 8 | 16 kg pkg. | 62.5 |
| 20 | Sir Rodney's Marmalade | 8 | 3 | 30 gift boxes | 81 |
| 26 | Gumbär Gummibärchen | 11 | 3 | 100 - 250 g bags | 31.23 |
| 27 | Schoggi Schokolade | 11 | 3 | 100 - 100 g pieces | 43.9 |
| 28 | Rössle Sauerkraut | 12 | 7 | 25 - 825 g cans | 45.6 |
| 29 | Thüringer Rostbratwurst | 12 | 6 | 50 bags x 30 sausgs. | 123.79 |
| 30 | Nord-Ost Matjeshering | 13 | 8 | 10 - 200 g glasses | 25.89 |
| 32 | Mascarpone Fabioli | 14 | 4 | 24 - 200 g pkgs. | 32 |
| 37 | Gravad lax | 17 | 8 | 12 - 500 g pkgs. | 26 |
| 38 | Côte de Blaye | 18 | 1 | 12 - 75 cl bottles | 263.5 |
| 43 | Ipoh Coffee | 20 | 1 | 16 - 500 g tins | 46 |
| 51 | Manjimup Dried Apples | 24 | 7 | 50 - 300 g pkgs. | 53 |
| 53 | Perth Pasties | 24 | 6 | 48 pieces | 32.8 |
| 55 | Pâté chinois | 25 | 6 | 24 boxes x 2 pies | 24 |
| 56 | Gnocchi di nonna Alice | 26 | 5 | 24 - 250 g pkgs. | 38 |
| 59 | Raclette Courdavault | 28 | 4 | 5 kg pkg. | 55 |
| 60 | Camembert Pierrot | 28 | 4 | 15 - 300 g rounds | 34 |
| 61 | Sirop d'érable | 29 | 2 | 24 - 500 ml bottles | 28.5 |
| 62 | Tarte au sucre | 29 | 3 | 48 pies | 49.3 |
| 63 | Vegie-spread | 7 | 2 | 15 - 625 g jars | 43.9 |
| 64 | Wimmers gute Semmelknödel | 12 | 5 | 20 bags x 4 pieces | 33.25 |
| 65 | Louisiana Fiery Hot Pepper Sauce | 2 | 2 | 32 - 8 oz bottles | 21.05 |
| 69 | Gudbrandsdalsost | 15 | 4 | 10 kg pkg. | 36 |
| 71 | Fløtemysost | 15 | 4 | 10 - 500 g pkgs. | 21.5 |
| 72 | Mozzarella di Giovanni | 14 | 4 | 24 - 200 g pkgs. | 34.8 |

### Result:

Number of Records: 7

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 11 | B's Beverages | Victoria Ashworth | Fauntleroy Circus | London | EC2 5NT | UK |
| 16 | Consolidated Holdings | Elizabeth Brown | Berkeley Gardens 12 Brewery | London | WX1 6LT | UK |
| 19 | Eastern Connection | Ann Devon | 35 King George | London | WX3 6FW | UK |
| 38 | Island Trading | Helen Bennett | Garden House Crowther Way | Cowes | PO31 7PJ | UK |
| 53 | North/South | Simon Crowther | South House 300 Queensbridge | London | SW7 1RZ | UK |
| 72 | Seven Seas Imports | Hari Kumar | 90 Wadhurst Rd. | London | OX15 4NB | UK |

**SOME** TRUE if any of the subquery values meet the condition

### SQL Statement:

SELECT \* FROM Products

WHERE Price > SOME (SELECT Price FROM Products WHERE Price > 20);

### Result:

Number of Records: 35

Bottom of Form

# **SQL CREATE DATABASE Statement**

# The SQL CREATE DATABASE Statement

The CREATE DATABASE statement is used to create a new SQL database.

### Syntax

CREATE DATABASE databasename;

## CREATE DATABASE Example

The following SQL statement creates a database called "testDB":

### Example

CREATE DATABASE testDB;

**Tip:** Make sure you have admin privilege before creating any database. Once a database is created, you can check it in the list of databases with the following SQL command: SHOW DATABASES;

# SQL DROP DATABASE Statement

# **The SQL DROP DATABASE Statement**

The DROP DATABASE statement is used to drop an existing SQL database.

### Syntax

DROP DATABASE databasename;

**Note:** Be careful before dropping a database. Deleting a database will result in loss of complete information stored in the database!

## DROP DATABASE Example

The following SQL statement drops the existing database "testDB":

### Example

DROP DATABASE testDB;

**Tip:** Make sure you have admin privilege before dropping any database. Once a database is dropped, you can check it in the list of databases with the following SQL command: SHOW DATABASES;

# **SQL BACKUP DATABASE for SQL Server**

The SQL BACKUP DATABASE Statement

The BACKUP DATABASE statement is used in SQL Server to create a full back up of an existing SQL database.

### Syntax

BACKUP DATABASE databasename  
TO DISK = 'filepath';

## The SQL BACKUP WITH DIFFERENTIAL Statement

A differential back up only backs up the parts of the database that have changed since the last full database backup.

### Syntax

BACKUP DATABASE databasename  
TO DISK = 'filepath'  
WITH DIFFERENTIAL;

## BACKUP DATABASE Example

The following SQL statement creates a full back up of the existing database "testDB" to the D disk:

### Example

BACKUP DATABASE testDB  
TO DISK = 'D:\backups\testDB.bak';

**Tip:** Always back up the database to a different drive than the actual database. Then, if you get a disk crash, you will not lose your backup file along with the database.

**BACKUP WITH DIFFERENTIAL Example**

The following SQL statement creates a differential back up of the database "testDB":

### Example

BACKUP DATABASE testDB  
TO DISK = 'D:\backups\testDB.bak'  
WITH DIFFERENTIAL;

**Tip:** A differential back up reduces the back up time (since only the changes are backed up).

# SQL CREATE TABLE Statement

## The SQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a new table in a database.

### Syntax

CREATE TABLE table\_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
   ....  
);

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

**Tip:** For an overview of the available data types, go to our complete [Data Types Reference](https://www.w3schools.com/sql/sql_datatypes.asp).

## SQL CREATE TABLE Example

The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

### Example

CREATE TABLE Persons (  
    PersonID int,  
    LastName varchar(255),  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255)  
);

The PersonID column is of type int and will hold an integer.

The LastName, FirstName, Address, and City columns are of type varchar and will hold characters, and the maximum length for these fields is 255 characters.

The empty "Persons" table will now look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PersonID** | **LastName** | **FirstName** | **Address** | **City** |
|  |  |  |  |  |

**Tip:** The empty "Persons" table can now be filled with data with the SQL [INSERT INTO](https://www.w3schools.com/sql/sql_insert.asp) statement.

## Create Table Using Another Table

A copy of an existing table can also be created using CREATE TABLE.

The new table gets the same column definitions. All columns or specific columns can be selected.

If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

### Syntax

CREATE TABLE new\_table\_name AS  
    SELECT column1, column2,...  
    FROM existing\_table\_name  
    WHERE ....;

The following SQL creates a new table called "TestTables" (which is a copy of the "Customers" table):

### Example

CREATE TABLE TestTable AS  
SELECT customername, contactname  
FROM customers;

# SQL DROP TABLE Statement

## The SQL DROP TABLE Statement

The DROP TABLE statement is used to drop an existing table in a database.

### Syntax

DROP TABLE table\_name;

**Note:** Be careful before dropping a table. Deleting a table will result in loss of complete information stored in the table!

## SQL DROP TABLE Example

The following SQL statement drops the existing table "Shippers":

### Example

DROP TABLE Shippers;

## SQL TRUNCATE TABLE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

### Syntax

TRUNCATE TABLE table\_name;

# SQL ALTER TABLE Statement

## SQL ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

## ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

ALTER TABLE table\_name  
ADD column\_name datatype;

The following SQL adds an "Email" column to the "Customers" table:

### Example

ALTER TABLE Customers  
ADD Email varchar(255);

## ALTER TABLE - DROP COLUMN

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

ALTER TABLE table\_name  
DROP COLUMN column\_name;

The following SQL deletes the "Email" column from the "Customers" table:

### Example

ALTER TABLE Customers  
DROP COLUMN Email;

## ALTER TABLE - RENAME COLUMN

To rename a column in a table, use the following syntax:

ALTER TABLE table\_name  
RENAME COLUMN old\_name to new\_name;

## ALTER TABLE - ALTER/MODIFY DATATYPE

To change the data type of a column in a table, use the following syntax:

**SQL Server / MS Access:**

ALTER TABLE table\_name  
ALTER COLUMN column\_name datatype;

**My SQL / Oracle (prior version 10G):**

ALTER TABLE table\_name  
MODIFY COLUMN column\_name datatype;

**Oracle 10G and later:**

ALTER TABLE table\_name  
MODIFY column\_name datatype;

## SQL ALTER TABLE Example

Look at the "Persons" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Now we want to add a column named "DateOfBirth" in the "Persons" table.

**We use the following SQL statement**:

ALTER TABLE Persons  
ADD DateOfBirth date;

Notice that the new column, "DateOfBirth", is of type date and is going to hold a date. The data type specifies what type of data the column can hold. For a complete reference of all the data types available in MS Access, MySQL, and SQL Server, go to our complete [Data Types reference](https://www.w3schools.com/sql/sql_datatypes.asp).

The "Persons" table will now look like this:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **LastName** | **FirstName** | **Address** | **City** | **DateOfBirth** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |  |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |  |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |  |

## Change Data Type Example

Now we want to change the data type of the column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

ALTER TABLE Persons  
ALTER COLUMN DateOfBirth year;

Notice that the "DateOfBirth" column is now of type year and is going to hold a year in a two- or four-digit format.

## DROP COLUMN Example

Next, we want to delete the column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

ALTER TABLE Persons  
DROP COLUMN DateOfBirth;

The "Persons" table will now look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

# **SQL Constraints**

SQL constraints are used to specify rules for data in a table.

## SQL Create Constraints

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

### Syntax

CREATE TABLE table\_name (  
    column1 datatype *constraint*,  
    column2 datatype *constraint*,  
    column3 datatype *constraint*,  
    ....  
);

## SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

* [NOT NULL](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value
* [UNIQUE](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different
* [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [FOREIGN KEY](https://www.w3schools.com/sql/sql_foreignkey.asp) - Prevents actions that would destroy links between tables
* [CHECK](https://www.w3schools.com/sql/sql_check.asp) - Ensures that the values in a column satisfies a specific condition
* [DEFAULT](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column if no value is specified
* [CREATE INDEX](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly

## SQL NOT NULL Constraint

By default, a column can hold NULL values.

The NOT NULL constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

## SQL NOT NULL on CREATE TABLE

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

### Example

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255) NOT NULL,  
    Age int  
);

## SQL NOT NULL on ALTER TABLE

To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:

**SQL Server / MS Access:**

ALTER TABLE Persons  
ALTER COLUMN Age int NOT NULL;

**My SQL / Oracle (prior version 10G):**

ALTER TABLE Persons  
MODIFY COLUMN Age int NOT NULL;

**Oracle 10G and later:**

ALTER TABLE Persons  
MODIFY Age int NOT NULL;

# SQL UNIQUE Constraint

## SQL UNIQUE Constraint

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

## SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:

**SQL Server / Oracle / MS Access:**

CREATE TABLE Persons (  
    ID int NOT NULL UNIQUE,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int  
);

**MySQL:**

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    UNIQUE (ID)  
);

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

**MySQL / SQL Server / Oracle / MS Access:**

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CONSTRAINT UC\_Person UNIQUE (ID,LastName)  
);

## SQL UNIQUE Constraint on ALTER TABLE

To create a UNIQUE constraint on the "ID" column when the table is already created, use the following SQL:

**MySQL / SQL Server / Oracle / MS Access:**

ALTER TABLE Persons  
ADD UNIQUE (ID);

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

**MySQL / SQL Server / Oracle / MS Access:**

ALTER TABLE Persons  
ADD CONSTRAINT UC\_Person UNIQUE (ID,LastName);

## DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

**MySQL:**

ALTER TABLE Persons  
DROP INDEX UC\_Person;

**SQL Server / Oracle / MS Access:**

ALTER TABLE Persons  
DROP CONSTRAINT UC\_Person;

# SQL PRIMARY KEY Constraint

## SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

## SQL PRIMARY KEY on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

**MySQL:**

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    PRIMARY KEY (ID)  
);

**SQL Server / Oracle / MS Access:**

CREATE TABLE Persons (  
    ID int NOT NULL PRIMARY KEY,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int  
);

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

**MySQL / SQL Server / Oracle / MS Access:**

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName)  
);

**Note:** In the example above there is only ONE PRIMARY KEY (PK\_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).

## SQL PRIMARY KEY on ALTER TABLE

To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:

**MySQL / SQL Server / Oracle / MS Access:**

ALTER TABLE Persons  
ADD PRIMARY KEY (ID);

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

**MySQL / SQL Server / Oracle / MS Access:**

ALTER TABLE Persons  
ADD CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName);

**Note:** If you use ALTER TABLE to add a primary key, the primary key column(s) must have been declared to not contain NULL values (when the table was first created).

## DROP a PRIMARY KEY Constraint

To drop a PRIMARY KEY constraint, use the following SQL:

**MySQL:**

ALTER TABLE Persons  
DROP PRIMARY KEY;

**SQL Server / Oracle / MS Access:**

ALTER TABLE Persons  
DROP CONSTRAINT PK\_Person;

# SQL FOREIGN KEY Constraint

## SQL FOREIGN KEY Constraint

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

**Look at the following two tables**:

### Persons Table

|  |  |  |  |
| --- | --- | --- | --- |
| **PersonID** | **LastName** | **FirstName** | **Age** |
| 1 | Hansen | Ola | 30 |
| 2 | Svendson | Tove | 23 |
| 3 | Pettersen | Kari | 20 |

### Orders Table

|  |  |  |
| --- | --- | --- |
| **OrderID** | **OrderNumber** | **PersonID** |
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 2 |
| 4 | 24562 | 1 |

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

## SQL FOREIGN KEY on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

**MySQL:**

CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)  
);

**SQL Server / Oracle / MS Access:**

CREATE TABLE Orders (  
    OrderID int NOT NULL PRIMARY KEY,  
    OrderNumber int NOT NULL,  
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)  
);

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

**MySQL / SQL Server / Oracle / MS Access:**

CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    CONSTRAINT FK\_PersonOrder FOREIGN KEY (PersonID)  
    REFERENCES Persons(PersonID)  
);

## SQL FOREIGN KEY on ALTER TABLE

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

**MySQL / SQL Server / Oracle / MS Access:**

ALTER TABLE Orders  
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

**MySQL / SQL Server / Oracle / MS Access:**

ALTER TABLE Orders  
ADD CONSTRAINT FK\_PersonOrder  
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)

## DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL:

**MySQL:**

ALTER TABLE Orders  
DROP FOREIGN KEY FK\_PersonOrder;

**SQL Server / Oracle / MS Access:**

ALTER TABLE Orders  
DROP CONSTRAINT FK\_PersonOrder;

# SQL CHECK Constraint

SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

## SQL CHECK on CREATE TABLE

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

**MySQL:**

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CHECK (Age>=18)  
);

**SQL Server / Oracle / MS Access:**

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int CHECK (Age>=18)  
);

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

**MySQL / SQL Server / Oracle / MS Access:**

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    City varchar(255),  
    CONSTRAINT CHK\_Person CHECK (Age>=18 AND City='Sandnes')  
);

## SQL CHECK on ALTER TABLE

To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:

**MySQL / SQL Server / Oracle / MS Access:**

ALTER TABLE Persons  
ADD CHECK (Age>=18);

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

**MySQL / SQL Server / Oracle / MS Access:**

ALTER TABLE Persons  
ADD CONSTRAINT CHK\_PersonAge CHECK (Age>=18 AND City='Sandnes');

## DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

**SQL Server / Oracle / MS Access:**

ALTER TABLE Persons  
DROP CONSTRAINT CHK\_PersonAge;

**MySQL:**

ALTER TABLE Persons  
DROP CHECK CHK\_PersonAge;

## SQL DEFAULT Constraint

The DEFAULT constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

## SQL DEFAULT on CREATE TABLE

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

**My SQL / SQL Server / Oracle / MS Access:**

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    City varchar(255) DEFAULT 'Sandnes'  
);

The DEFAULT constraint can also be used to insert system values, by using functions like [GETDATE()](https://www.w3schools.com/sql/func_sqlserver_getdate.asp):

CREATE TABLE Orders (  
    ID int NOT NULL,  
    OrderNumber int NOT NULL,  
    OrderDate date DEFAULT GETDATE()  
);

## SQL DEFAULT on ALTER TABLE

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

**MySQL:**

ALTER TABLE Persons  
ALTER City SET DEFAULT 'Sandnes';

**SQL Server:**

ALTER TABLE Persons  
ADD CONSTRAINT df\_City  
DEFAULT 'Sandnes' FOR City;

**MS Access:**

ALTER TABLE Persons  
ALTER COLUMN City SET DEFAULT 'Sandnes';

**Oracle:**

ALTER TABLE Persons  
MODIFY City DEFAULT 'Sandnes';

## DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:

**MySQL:**

ALTER TABLE Persons  
ALTER City DROP DEFAULT;

**SQL Server / Oracle / MS Access:**

ALTER TABLE Persons  
ALTER COLUMN City DROP DEFAULT;

**SQL Server:**

ALTER TABLE Persons  
ALTER COLUMN City DROP DEFAULT

# SQL CREATE INDEX Statement

## SQL CREATE INDEX Statement

The CREATE INDEX statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

**Note:** Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

### CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

CREATE INDEX index\_name  
ON table\_name (column1, column2, ...);

### CREATE UNIQUE INDEX Syntax

Creates a unique index on a table. Duplicate values are not allowed:

CREATE UNIQUE INDEX index\_name  
ON table\_name (column1, column2, ...);

**Note:** The syntax for creating indexes varies among different databases. Therefore: Check the syntax for creating indexes in your database.

## CREATE INDEX Example

The SQL statement below creates an index named "idx\_lastname" on the "LastName" column in the "Persons" table:

CREATE INDEX idx\_lastname  
ON Persons (LastName);

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

**CREATE INDEX idx\_pname**  
ON Persons (LastName, FirstName);

## DROP INDEX Statement

The DROP INDEX statement is used to delete an index in a table.

**MS Access:**

DROP INDEX index\_name ON table\_name;

**SQL Server:**

DROP INDEX table\_name.index\_name;

**DB2/Oracle:**

DROP INDEX index\_name;

**MySQL:**

ALTER TABLE table\_nameDROP INDEX index\_name;

# SQL AUTO INCREMENT Field

## AUTO INCREMENT Field

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

## Syntax for MySQL

The following SQL statement defines the "Personid" column to be an auto-increment primary key field in the "Persons" table:

CREATE TABLE Persons (  
    Personid int NOT NULL AUTO\_INCREMENT,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    PRIMARY KEY (Personid)  
);

MySQL uses the AUTO\_INCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTO\_INCREMENT is 1, and it will increment by 1 for each new record.

To let the AUTO\_INCREMENT sequence start with another value, use the following SQL statement:

ALTER TABLE Persons AUTO\_INCREMENT=100;

To insert a new record into the "Persons" table, we will NOT have to specify a value for the "Personid" column (a unique value will be added automatically):

INSERT INTO Persons (FirstName,LastName)  
VALUES ('Lars','Monsen');

The SQL statement above would insert a new record into the "Persons" table. The "Personid" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

## Syntax for SQL Server

The following SQL statement defines the "Personid" column to be an auto-increment primary key field in the "Persons" table:

CREATE TABLE Persons (  
    Personid int IDENTITY(1,1) PRIMARY KEY,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int  
);

The MS SQL Server uses the IDENTITY keyword to perform an auto-increment feature.

In the example above, the starting value for IDENTITY is 1, and it will increment by 1 for each new record.

**Tip:** To specify that the "Personid" column should start at value 10 and increment by 5, change it to IDENTITY(10,5).

To insert a new record into the "Persons" table, we will NOT have to specify a value for the "Personid" column (a unique value will be added automatically):

INSERT INTO Persons (FirstName,LastName)  
VALUES ('Lars','Monsen');

The SQL statement above would insert a new record into the "Persons" table. The "Personid" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

## Syntax for Access

The following SQL statement defines the "Personid" column to be an auto-increment primary key field in the "Persons" table:

CREATE TABLE Persons (  
    Personid AUTOINCREMENT PRIMARY KEY,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int  
);

The MS Access uses the AUTOINCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTOINCREMENT is 1, and it will increment by 1 for each new record.

**Tip:** To specify that the "Personid" column should start at value 10 and increment by 5, change the autoincrement to AUTOINCREMENT(10,5).

To insert a new record into the "Persons" table, we will NOT have to specify a value for the "Personid" column (a unique value will be added automatically):

INSERT INTO Persons (FirstName,LastName)  
VALUES ('Lars','Monsen');

The SQL statement above would insert a new record into the "Persons" table. The "Personid" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

## Syntax for Oracle

In Oracle the code is a little bit more tricky.

You will have to create an auto-increment field with the sequence object (this object generates a number sequence).

Use the following CREATE SEQUENCE syntax:

CREATE SEQUENCE seq\_person  
MINVALUE 1  
START WITH 1  
INCREMENT BY 1  
CACHE 10;

The code above creates a sequence object called seq\_person, that starts with 1 and will increment by 1. It will also cache up to 10 values for performance. The cache option specifies how many sequence values will be stored in memory for faster access.

To insert a new record into the "Persons" table, we will have to use the nextval function (this function retrieves the next value from seq\_person sequence):

INSERT INTO Persons (Personid,FirstName,LastName)  
VALUES (seq\_person.nextval,'Lars','Monsen');

The SQL statement above would insert a new record into the "Persons" table. The "Personid" column would be assigned the next number from the seq\_person sequence. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

# **SQL Working With Dates**

## SQL Dates

The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.

As long as your data contains only the date portion, your queries will work as expected. However, if a time portion is involved, it gets more complicated.

## SQL Date Data Types

**MySQL** comes with the following data types for storing a date or a date/time value in the database:

* DATE - format YYYY-MM-DD
* DATETIME - format: YYYY-MM-DD HH:MI:SS
* TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
* YEAR - format YYYY or YY

**SQL Server** comes with the following data types for storing a date or a date/time value in the database:

* DATE - format YYYY-MM-DD
* DATETIME - format: YYYY-MM-DD HH:MI:SS
* SMALLDATETIME - format: YYYY-MM-DD HH:MI:SS
* TIMESTAMP - format: a unique number

**Note:** The date types are chosen for a column when you create a new table in your database!

## SQL Working with Dates

Look at the following table:

### Orders Table

|  |  |  |
| --- | --- | --- |
| **OrderId** | **ProductName** | **OrderDate** |
| 1 | Geitost | 2008-11-11 |
| 2 | Camembert Pierrot | 2008-11-09 |
| 3 | Mozzarella di Giovanni | 2008-11-11 |
| 4 | Mascarpone Fabioli | 2008-10-29 |

Now we want to select the records with an OrderDate of "2008-11-11" from the table above.

We use the following SELECT statement:

SELECT \* FROM Orders WHERE OrderDate='2008-11-11'

The result-set will look like this:

|  |  |  |
| --- | --- | --- |
| **OrderId** | **ProductName** | **OrderDate** |
| 1 | Geitost | 2008-11-11 |
| 3 | Mozzarella di Giovanni | 2008-11-11 |

**Note:** Two dates can easily be compared if there is no time component involved!

Now, assume that the "Orders" table looks like this (notice the added time-component in the "OrderDate" column):

|  |  |  |
| --- | --- | --- |
| **OrderId** | **ProductName** | **OrderDate** |
| 1 | Geitost | 2008-11-11 13:23:44 |
| 2 | Camembert Pierrot | 2008-11-09 15:45:21 |
| 3 | Mozzarella di Giovanni | 2008-11-11 11:12:01 |
| 4 | Mascarpone Fabioli | 2008-10-29 14:56:59 |

If we use the same SELECT statement as above:

SELECT \* FROM Orders WHERE OrderDate='2008-11-11'

we will get no result! This is because the query is looking only for dates with no time portion.

**Tip:** To keep your queries simple and easy to maintain, do not use time-components in your dates, unless you have to!

# **SQL Views**

## SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the CREATE VIEW statement.

### CREATE VIEW Syntax

CREATE VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

**Note:** A view always shows up-to-date data! The database engine recreates the view, every time a user queries it.

## SQL CREATE VIEW Examples

The following SQL creates a view that shows all customers from Brazil:

### Example

CREATE VIEW [Brazil Customers] AS  
SELECT CustomerName, ContactName  
FROM Customers  
WHERE Country = 'Brazil';

We can query the view above as follows:

### Example

SELECT \* FROM [Brazil Customers];

The following SQL creates a view that selects every product in the "Products" table with a price higher than the average price:

### Example

CREATE VIEW [Products Above Average Price] AS  
SELECT ProductName, Price  
FROM Products  
WHERE Price > (SELECT AVG(Price) FROM Products);

We can query the view above as follows:

### Example

SELECT \* FROM [Products Above Average Price];

## SQL Updating a View

A view can be updated with the CREATE OR REPLACE VIEW statement.

### SQL CREATE OR REPLACE VIEW Syntax

CREATE OR REPLACE VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

The following SQL adds the "City" column to the "Brazil Customers" view:

### Example

CREATE OR REPLACE VIEW [Brazil Customers] AS  
SELECT CustomerName, ContactName, City  
FROM Customers  
WHERE Country = 'Brazil';

## SQL Dropping a View

A view is deleted with the DROP VIEW statement.

### SQL DROP VIEW Syntax

DROP VIEW view\_name;

The following SQL drops the "Brazil Customers" view:

### Example

DROP VIEW [Brazil Customers];

## SQL Injection

SQL injection is a code injection technique that might destroy your database.

SQL injection is one of the most common web hacking techniques.

SQL injection is the placement of malicious code in SQL statements, via web page input.

## SQL in Web Pages

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will **unknowingly** run on your database.

Look at the following example which creates a SELECT statement by adding a variable (txtUserId) to a select string. The variable is fetched from user input (getRequestString):

### Example

txtUserId = getRequestString("UserId");  
txtSQL = "SELECT \* FROM Users WHERE UserId = " + txtUserId;

The rest of this chapter describes the potential dangers of using user input in SQL statements.

## SQL Injection Based on 1=1 is Always True

Look at the example above again. The original purpose of the code was to create an SQL statement to select a user, with a given user id.

If there is nothing to prevent a user from entering "wrong" input, the user can enter some "smart" input like this:

**UserId**:

Then, the SQL statement will look like this:

SELECT \* FROM Users WHERE UserId = 105 OR 1=1;

The SQL above is valid and will return ALL rows from the "Users" table, since **OR 1=1** is always TRUE.

Does the example above look dangerous? What if the "Users" table contains names and passwords?

The SQL statement above is much the same as this:

SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;

A hacker might get access to all the user names and passwords in a database, by simply inserting 105 OR 1=1 into the input field.

## SQL Injection Based on ""="" is Always True

Here is an example of a user login on a web site:

Username:

Password:

### Example

uName = getRequestString("username");  
uPass = getRequestString("userpassword");  
  
sql = 'SELECT \* FROM Users WHERE Name ="' + uName + '" AND Pass ="' + uPass + '"'

### Result

SELECT \* FROM Users WHERE Name ="John Doe" AND Pass ="myPass"

A hacker might get access to user names and passwords in a database by simply inserting " OR ""=" into the user name or password text box:

User Name:

Password:

The code at the server will create a valid SQL statement like this:

### Result

SELECT \* FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""

The SQL above is valid and will return all rows from the "Users" table, since **OR ""=""** is always TRUE.

## SQL Injection Based on Batched SQL Statements

Most databases support batched SQL statement.

A batch of SQL statements is a group of two or more SQL statements, separated by semicolons.

The SQL statement below will return all rows from the "Users" table, then delete the "Suppliers" table.

### Example

SELECT \* FROM Users; DROP TABLE Suppliers

Look at the following example:

### Example

txtUserId = getRequestString("UserId");  
txtSQL = "SELECT \* FROM Users WHERE UserId = " + txtUserId;

And the following input:

User id:

The valid SQL statement would look like this:

### Result

SELECT \* FROM Users WHERE UserId = 105; DROP TABLE Suppliers;

## Use SQL Parameters for Protection

To protect a web site from SQL injection, you can use SQL parameters.

SQL parameters are values that are added to an SQL query at execution time, in a controlled manner.

### ASP.NET Razor Example

txtUserId = getRequestString("UserId");  
txtSQL = "SELECT \* FROM Users WHERE UserId = @0";  
db.Execute(txtSQL,txtUserId);

Note that parameters are represented in the SQL statement by a @ marker.

The SQL engine checks each parameter to ensure that it is correct for its column and are treated literally, and not as part of the SQL to be executed.

### Another Example

txtNam = getRequestString("CustomerName");  
txtAdd = getRequestString("Address");  
txtCit = getRequestString("City");  
txtSQL = "INSERT INTO Customers (CustomerName,Address,City) Values(@0,@1,@2)";  
db.Execute(txtSQL,txtNam,txtAdd,txtCit);

## Examples

The following examples shows how to build parameterized queries in some common web languages.

**SELECT STATEMENT IN ASP.NET**:

txtUserId = getRequestString("UserId");  
sql = "SELECT \* FROM Customers WHERE CustomerId = @0";  
command = new SqlCommand(sql);  
command.Parameters.AddWithValue("@0",txtUserId);  
command.ExecuteReader();

**INSERT INTO STATEMENT IN ASP.NET**:

txtNam = getRequestString("CustomerName");  
txtAdd = getRequestString("Address");  
txtCit = getRequestString("City");  
txtSQL = "INSERT INTO Customers (CustomerName,Address,City) Values(@0,@1,@2)";  
command = new SqlCommand(txtSQL);  
command.Parameters.AddWithValue("@0",txtNam);  
command.Parameters.AddWithValue("@1",txtAdd);  
command.Parameters.AddWithValue("@2",txtCit);  
command.ExecuteNonQuery();

**INSERT INTO STATEMENT IN PHP**:

$stmt = $dbh->prepare("INSERT INTO Customers (CustomerName,Address,City)  
VALUES (:nam, :add, :cit)");  
$stmt->bindParam(':nam', $txtNam);  
$stmt->bindParam(':add', $txtAdd);  
$stmt->bindParam(':cit', $txtCit);  
$stmt->execute();

# **SQL Hosting**

If you want your web site to be able to store and retrieve data from a database, your web server should have access to a database-system that uses the SQL language.

If your web server is hosted by an Internet Service Provider (ISP), you will have to look for SQL hosting plans.

The most common SQL hosting databases are MS SQL Server, Oracle, MySQL, and MS Access.

## MS SQL Server

Microsoft's SQL Server is a popular database software for database-driven web sites with high traffic.

SQL Server is a very powerful, robust and full featured SQL database system.

## Oracle

Oracle is also a popular database software for database-driven web sites with high traffic.

Oracle is a very powerful, robust and full featured SQL database system.

## MySQL

MySQL is also a popular database software for web sites.

MySQL is a very powerful, robust and full featured SQL database system.

MySQL is an inexpensive alternative to the expensive Microsoft and Oracle solutions.

## MS Access

When a web site requires only a simple database, Microsoft Access can be a solution.

MS Access is not well suited for very high-traffic, and not as powerful as MySQL, SQL Server, or Oracle.

# SQL Data Types for MySQL, SQL Server, and MS Access

The data type of a column defines what value the column can hold: integer, character, money, date and time, binary, and so on.

## SQL Data Types

Each column in a database table is required to have a name and a data type.

An SQL developer must decide what type of data that will be stored inside each column when creating a table. The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

**Note:** Data types might have different names in different database. And even if the name is the same, the size and other details may be different! **Always check the documentation!**

## MySQL Data Types (Version 8.0)

In MySQL there are three main data types: string, numeric, and date and time.

### String Data Types

|  |  |
| --- | --- |
| **Data type** | **Description** |
| CHAR(size) | A FIXED length string (can contain letters, numbers, and special characters). The size parameter specifies the column length in characters - can be from 0 to 255. Default is 1 |
| VARCHAR(size) | A VARIABLE length string (can contain letters, numbers, and special characters). The size parameter specifies the maximum string length in characters - can be from 0 to 65535 |
| BINARY(size) | Equal to CHAR(), but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1 |
| VARBINARY(size) | Equal to VARCHAR(), but stores binary byte strings. The size parameter specifies the maximum column length in bytes. |
| TINYBLOB | For BLOBs (Binary Large Objects). Max length: 255 bytes |
| TINYTEXT | Holds a string with a maximum length of 255 characters |
| TEXT(size) | Holds a string with a maximum length of 65,535 bytes |
| BLOB(size) | For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data |
| MEDIUMTEXT | Holds a string with a maximum length of 16,777,215 characters |
| MEDIUMBLOB | For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data |
| LONGTEXT | Holds a string with a maximum length of 4,294,967,295 characters |
| LONGBLOB | For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data |
| ENUM(val1, val2, val3, ...) | A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them |
| SET(val1, val2, val3, ...) | A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list |

### Numeric Data Types

|  |  |
| --- | --- |
| **Data type** | **Description** |
| BIT(size) | A bit-value type. The number of bits per value is specified in size. The size parameter can hold a value from 1 to 64. The default value for size is 1. |
| TINYINT(size) | A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The size parameter specifies the maximum display width (which is 255) |
| BOOL | Zero is considered as false, nonzero values are considered as true. |
| BOOLEAN | Equal to BOOL |
| SMALLINT(size) | A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The size parameter specifies the maximum display width (which is 255) |
| MEDIUMINT(size) | A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The size parameter specifies the maximum display width (which is 255) |
| INT(size) | A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255) |
| INTEGER(size) | Equal to INT(size) |
| BIGINT(size) | A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The size parameter specifies the maximum display width (which is 255) |
| FLOAT(size, d) | A floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions |
| FLOAT(p) | A floating point number. MySQL uses the p value to determine whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT(). If p is from 25 to 53, the data type becomes DOUBLE() |
| DOUBLE(size, d) | A normal-size floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter |
| DOUBLE PRECISION(size, d) |  |
| DECIMAL(size, d) | An exact fixed-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. The maximum number for size is 65. The maximum number for d is 30. The default value for size is 10. The default value for d is 0. |
| DEC(size, d) | Equal to DECIMAL(size,d) |

**Note:** All the numeric data types may have an extra option: UNSIGNED or ZEROFILL. If you add the UNSIGNED option, MySQL disallows negative values for the column. If you add the ZEROFILL option, MySQL automatically also adds the UNSIGNED attribute to the column.

### Date and Time Data Types

|  |  |
| --- | --- |
| **Data type** | **Description** |
| DATE | A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31' |
| DATETIME(fsp) | A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time |
| TIMESTAMP(fsp) | A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT\_TIMESTAMP and ON UPDATE CURRENT\_TIMESTAMP in the column definition |
| TIME(fsp) | A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59' |
| YEAR | A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format. |

## SQL Server Data Types

### String Data Types

|  |  |  |  |
| --- | --- | --- | --- |
| **Data type** | **Description** | **Max size** | **Storage** |
| char(n) | Fixed width character string | 8,000 characters | Defined width |
| varchar(n) | Variable width character string | 8,000 characters | 2 bytes + number of chars |
| varchar(max) | Variable width character string | 1,073,741,824 characters | 2 bytes + number of chars |
| text | Variable width character string | 2GB of text data | 4 bytes + number of chars |
| nchar | Fixed width Unicode string | 4,000 characters | Defined width x 2 |
| nvarchar | Variable width Unicode string | 4,000 characters |  |
| nvarchar(max) | Variable width Unicode string | 536,870,912 characters |  |
| ntext | Variable width Unicode string | 2GB of text data |  |
| binary(n) | Fixed width binary string | 8,000 bytes |  |
| varbinary | Variable width binary string | 8,000 bytes |  |
| varbinary(max) | Variable width binary string | 2GB |  |
| image | Variable width binary string | 2GB |  |

### Numeric Data Types

|  |  |  |
| --- | --- | --- |
| **Data type** | **Description** | **Storage** |
| bit | Integer that can be 0, 1, or NULL |  |
| tinyint | Allows whole numbers from 0 to 255 | 1 byte |
| smallint | Allows whole numbers between -32,768 and 32,767 | 2 bytes |
| int | Allows whole numbers between -2,147,483,648 and 2,147,483,647 | 4 bytes |
| bigint | Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807 | 8 bytes |
| decimal(p,s) | Fixed precision and scale numbers.  Allows numbers from -10^38 +1 to 10^38 –1.  The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18.  The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0 | 5-17 bytes |
| numeric(p,s) | Fixed precision and scale numbers.  Allows numbers from -10^38 +1 to 10^38 –1.  The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18.  The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0 | 5-17 bytes |
| smallmoney | Monetary data from -214,748.3648 to 214,748.3647 | 4 bytes |
| money | Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807 | 8 bytes |
| float(n) | Floating precision number data from -1.79E + 308 to 1.79E + 308.  The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53. | 4 or 8 bytes |
| real | Floating precision number data from -3.40E + 38 to 3.40E + 38 | 4 bytes |

### Date and Time Data Types

|  |  |  |
| --- | --- | --- |
| **Data type** | **Description** | **Storage** |
| datetime | From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds | 8 bytes |
| datetime2 | From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds | 6-8 bytes |
| smalldatetime | From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute | 4 bytes |
| date | Store a date only. From January 1, 0001 to December 31, 9999 | 3 bytes |
| time | Store a time only to an accuracy of 100 nanoseconds | 3-5 bytes |
| datetimeoffset | The same as datetime2 with the addition of a time zone offset | 8-10 bytes |
| timestamp | Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable |  |

### Other Data Types

|  |  |
| --- | --- |
| **Data type** | **Description** |
| sql\_variant | Stores up to 8,000 bytes of data of various data types, except text, ntext, and timestamp |
| uniqueidentifier | Stores a globally unique identifier (GUID) |
| xml | Stores XML formatted data. Maximum 2GB |
| cursor | Stores a reference to a cursor used for database operations |
| table | Stores a result-set for later processing |

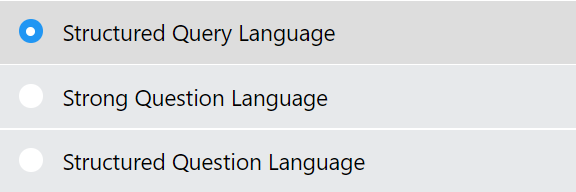
## MS Access Data Types

|  |  |  |
| --- | --- | --- |
| **Data type** | **Description** | **Storage** |
| Text | Use for text or combinations of text and numbers. 255 characters maximum |  |
| Memo | Memo is used for larger amounts of text. Stores up to 65,536 characters. **Note:** You cannot sort a memo field. However, they are searchable |  |
| Byte | Allows whole numbers from 0 to 255 | 1 byte |
| Integer | Allows whole numbers between -32,768 and 32,767 | 2 bytes |
| Long | Allows whole numbers between -2,147,483,648 and 2,147,483,647 | 4 bytes |
| Single | Single precision floating-point. Will handle most decimals | 4 bytes |
| Double | Double precision floating-point. Will handle most decimals | 8 bytes |
| Currency | Use for currency. Holds up to 15 digits of whole dollars, plus 4 decimal places. **Tip:** You can choose which country's currency to use | 8 bytes |
| AutoNumber | AutoNumber fields automatically give each record its own number, usually starting at 1 | 4 bytes |
| Date/Time | Use for dates and times | 8 bytes |
| Yes/No | A logical field can be displayed as Yes/No, True/False, or On/Off. In code, use the constants True and False (equivalent to -1 and 0). **Note:** Null values are not allowed in Yes/No fields | 1 bit |
| Ole Object | Can store pictures, audio, video, or other BLOBs (Binary Large Objects) | up to 1GB |
| Hyperlink | Contain links to other files, including web pages |  |
| Lookup Wizard | Let you type a list of options, which can then be chosen from a drop-down list | 4 bytes |

# **SQL Quiz**

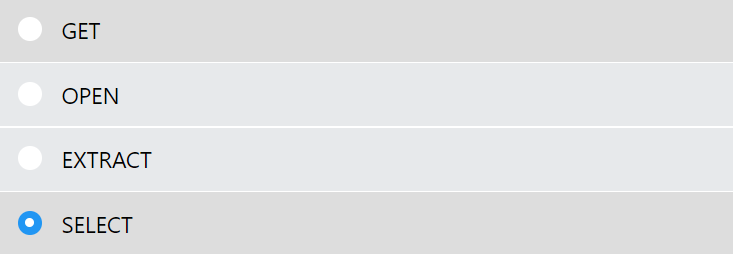
### Question 1 of 25:

What does SQL stand for?



### Question 2 of 25:

Which SQL statement is used to extract data from a database?



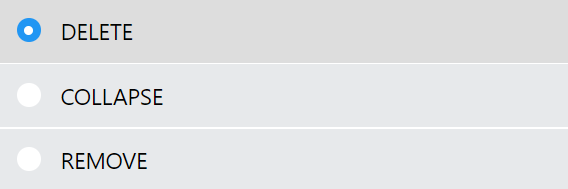
### Question 3 of 25:

Which SQL statement is used to update data in a database?



### Question 4 of 25:

Which SQL statement is used to delete data from a database?



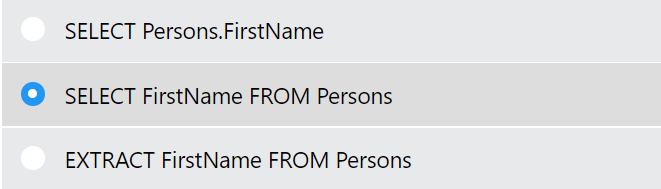
### Question 5 of 25:

Which SQL statement is used to insert new data in a database?



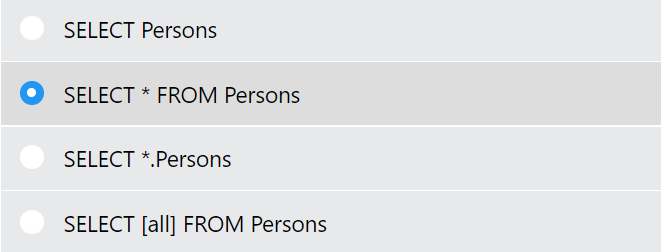
Question 6 of 25:

With SQL, how do you select a column named "FirstName" from a table named "Persons"?



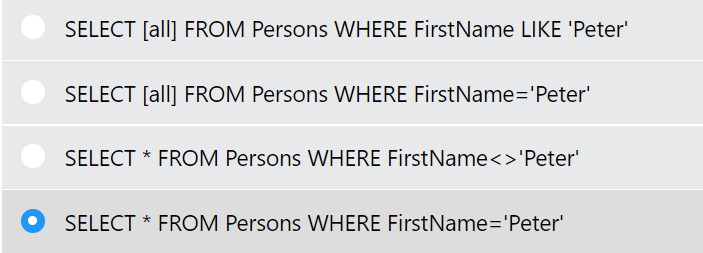
### Question 7 of 25:

With SQL, how do you select all the columns from a table named "Persons"?



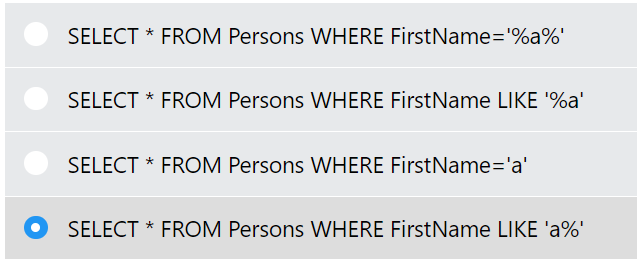
### Question 8 of 25:

With SQL, how do you select all the records from a table named "Persons" where the value of the column "FirstName" is "Peter"?



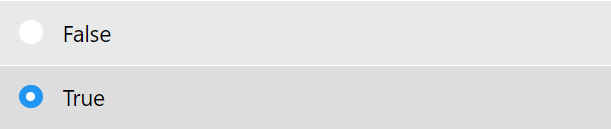
Question 9 of 25:

With SQL, how do you select all the records from a table named "Persons" where the value of the column "FirstName" starts with an "a"?



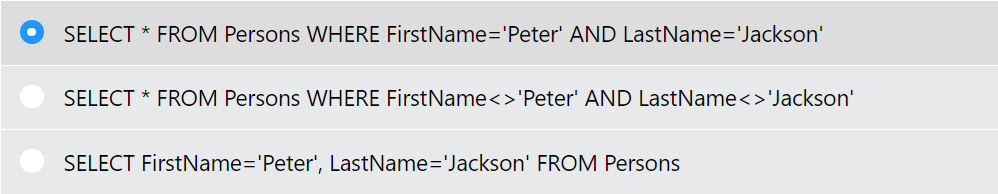
### Question 10 of 25:

The OR operator displays a record if ANY conditions listed are true. The AND operator displays a record if ALL of the conditions listed are true



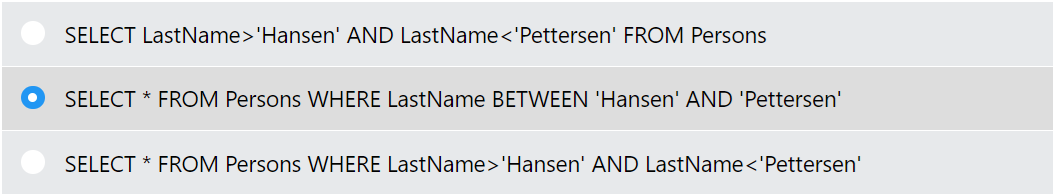
### Question 11 of 25:

With SQL, how do you select all the records from a table named "Persons" where the "FirstName" is "Peter" and the "LastName" is "Jackson"?



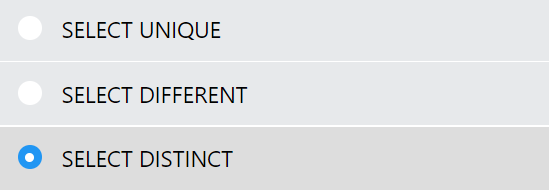
### Question 12 of 25:

With SQL, how do you select all the records from a table named "Persons" where the "LastName" is alphabetically between (and including) "Hansen" and "Pettersen"?



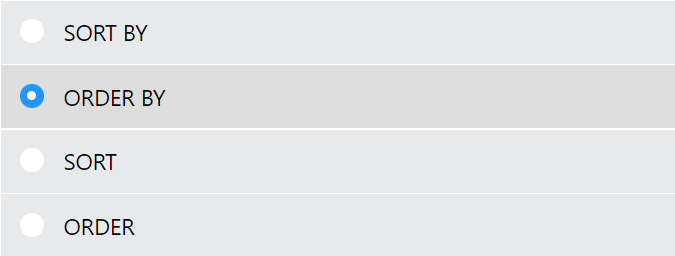
Question 13 of 25:

Which SQL statement is used to return only different values?



### Question 14 of 25:

Which SQL keyword is used to sort the result-set?



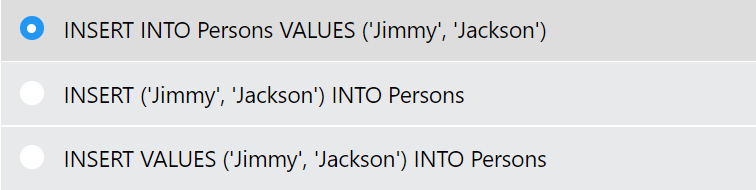
### Question 15 of 25:

With SQL, how can you return all the records from a table named "Persons" sorted descending by "FirstName"?



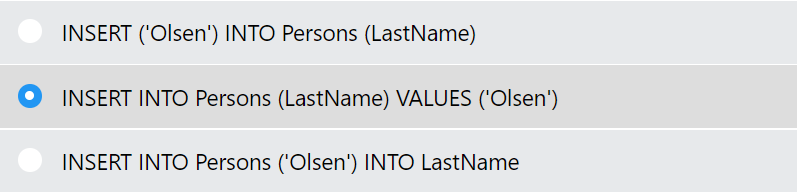
### Question 16 of 25:

With SQL, how can you insert a new record into the "Persons" table?



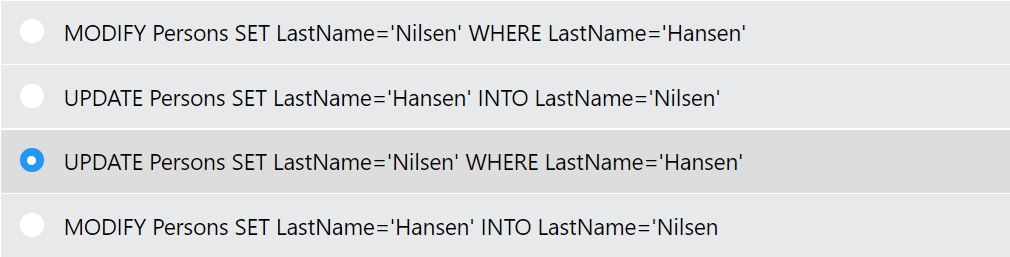
### Question 17 of 25:

With SQL, how can you insert "Olsen" as the "LastName" in the "Persons" table?



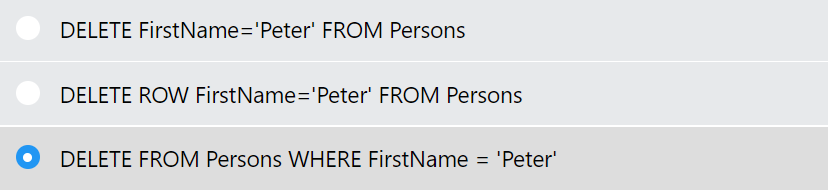
### Question 18 of 25:

How can you change "Hansen" into "Nilsen" in the "LastName" column in the Persons table?



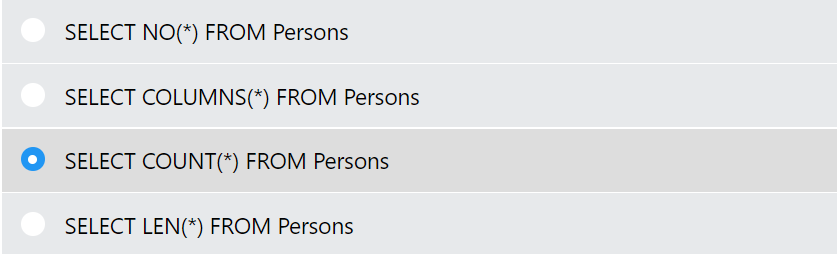
### Question 19 of 25:

With SQL, how can you delete the records where the "FirstName" is "Peter" in the Persons Table?



### Question 20 of 25:

With SQL, how can you return the number of records in the "Persons" table?



### Question 21 of 25:

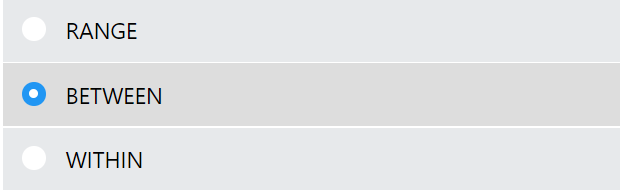
What is the most common type of join?



Question 22 of 25:

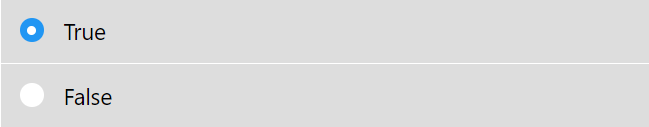
Which operator is used to select values within a range?

Top of Form



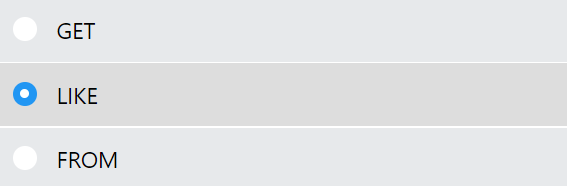
### Question 23 of 25:

The NOT NULL constraint enforces a column to not accept NULL values.



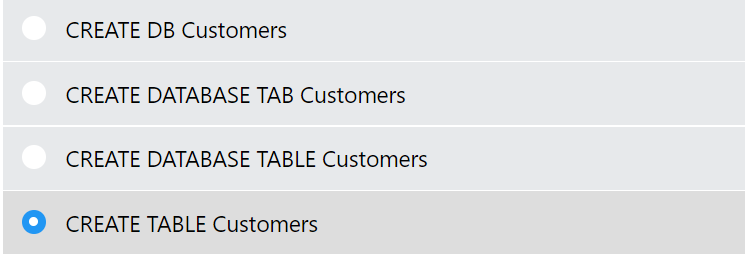
### Question 24 of 25:

Which operator is used to search for a specified pattern in a column?



### Question 25 of 25:

Which SQL statement is used to create a database table called 'Customers'?



Top of Form

Top of Form

Top of Form