**SQL (Structured Query Language)**

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

Our SQL tutorial will teach you how to use SQL in: MySQL, SQL Server, MS Access, Oracle, Sybase, Informix, Postgres, and other database systems.

Example

SELECT \* FROM Customers;

Bottom of Form

# **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

## 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.

# **SQL Syntax**

## Database Tables

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

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

In this tutorial we will use the well-known Northwind sample database (included in MS Access and MS SQL Server).

Below is a selection from the "Customers" table:

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

## SQL Statements

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

The following SQL statement selects all the records in the "Customers" table:

### Example

SELECT \* FROM Customers;

Keep in Mind That...

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

In this tutorial we will write all SQL keywords in upper-case.

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.

In this tutorial, we will use semicolon at the end of each SQL statement.

Some of The Most Important SQL Commands

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

# **SQL SELECT Statement**

## The SQL SELECT Statement

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

The data returned is stored in a result table, called the result-set.

### SELECT Syntax

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

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

SELECT \* FROM table\_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:

## SELECT Column Example

The following SQL statement selects the "CustomerName" and "City" columns from the "Customers" table:

### Example

SELECT CustomerName, City FROM Customers;

## SELECT \* Example

The following SQL statement selects all the columns from the "Customers" table:

### Example

SELECT \* FROM Customers;

Top of Form

## Exercise:

Insert the missing statement to get all the columns from the Customers table.

 \* FROM Customers;

Bottom of Form

# **SQL SELECT DISTINCT Statement**

## The SQL SELECT DISTINCT Statement

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

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

### SELECT DISTINCT

### Syntax

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

## SELECT Example Without DISTINCT

The following SQL statement selects all (including the duplicates) values from the "Country" column in the "Customers" table:

### Example

SELECT Country FROM Customers;

Now, let us use the SELECT DISTINCT statement and see the result.

## SELECT DISTINCT Examples

The following SQL statement selects only the DISTINCT values from the "Country" column in the "Customers" table:

### Example

SELECT DISTINCT Country FROM Customers;

The following SQL statement lists the number of different (distinct) customer countries:

### Example

SELECT COUNT(DISTINCT Country) FROM Customers;

**Note: The example above will not work in Firefox!** Because COUNT(DISTINCT column\_name) is not supported in Microsoft Access databases. Firefox is using Microsoft Access in our examples.

Here is the workaround for MS Access:

### Example

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

# **SQL WHERE Clause**

## 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.

### WHERE Syntax

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.!

## WHERE Clause Example

The following SQL statement selects all the customers from the country "Mexico", in the "Customers" table:

### Example

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

**ext 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

SELECT \* FROM Customers  
WHERE CustomerID=1;

Operators in The WHERE Clause

The following operators can be used in the WHERE clause:

**Operator** **Description**

= Equal

> Greater than

< Less than

>= Greater than or equal

<= Less than or equal

<> Not equal.

BETWEEN Between a certain range

LIKE Search for a pattern

IN To specify multiple possible values for a column

**Note: In some versions of SQL this operator may be written as !=**

# **SQL AND, OR and NOT Operators**

## The SQL AND, OR and NOT Operators

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

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

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

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

### AND Syntax

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

### OR Syntax

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

### NOT Syntax

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

**AND Example**

The following SQL statement selects all fields from "Customers" where country is "Germany" AND city is "Berlin":

Example

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

**OR Example**

The following SQL statement selects all fields from "Customers" where city is "Berlin" OR "München":

Example

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

The following SQL statement selects all fields from "Customers" where country is "Germany" OR "Spain":

Example

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

NOT Example

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

Example

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

**Combining AND, OR and NOT**

You can also combine the AND, OR and NOT operators.

The following SQL statement selects all fields from "Customers" where country is "Germany" AND city must be "Berlin" OR "München" (use parenthesis to form complex expressions):

Example

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

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

Example

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

# **SQL ORDER BY Keyword**

## The SQL ORDER BY Keyword

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

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

### ORDER BY Syntax

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

ORDER BY Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" column:

Example

SELECT \* FROM Customers  
ORDER BY Country;

ORDER BY DESC Example

The following SQL statement selects all customers from the "Customers" table, sorted DESCENDING by the "Country" column:

Example

SELECT \* FROM Customers  
ORDER BY Country DESC;

ORDER BY Several Columns Example

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

SELECT \* FROM Customers  
ORDER BY Country, CustomerName;

ORDER BY Several Columns Example 2

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

Example

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

# **SQL INSERT INTO Statement**

## 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, ...);

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');

**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');

# **SQL NULL Values**

## 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;

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;

**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

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

# **SQL UPDATE Statement**

## The SQL UPDATE Statement

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

### UPDATE Syntax

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

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

UPDATE 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;

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';

Update Warning!

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

Example

UPDATE Customers  
SET ContactName='Juan';

# **SQL DELETE Statement**

## The SQL DELETE Statement

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

### DELETE Syntax

DELETE FROM table\_name WHERE condition;

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

## SQL DELETE Example

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

### Example

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

## Delete All Records

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

DELETE FROM table\_name;

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

### Example

DELETE FROM Customers;

# **SQL TOP, LIMIT, FETCH FIRST or ROWNUM Clause**

## The SQL SELECT TOP Clause

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

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

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

## SQL TOP, LIMIT and FETCH FIRST Examples

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

### Example

SELECT TOP 3 \* FROM Customers;

The following SQL statement shows the equivalent example for MySQL:

### Example

SELECT \* FROM Customers  
LIMIT 3;

The following SQL statement shows the equivalent example for Oracle:

### Example

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;

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';

The following SQL statement shows the equivalent example for MySQL:

### Example

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

The following SQL statement shows the equivalent example for Oracle:

### Example

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

# **SQL MIN() and MAX() Functions**

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

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

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

### MIN() Syntax

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

### MAX() Syntax

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

## MIN() Example

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

### Example

SELECT MIN(Price) AS SmallestPrice  
FROM Products;

## MAX() Example

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

### Example

SELECT MAX(Price) AS LargestPrice  
FROM Products;

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

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

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

### COUNT() Syntax

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

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

### AVG() Syntax

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

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

### SUM() Syntax

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

## COUNT() Example

The following SQL statement finds the number of products:

### Example

SELECT COUNT(ProductID)  
FROM Products;

**Note:** NULL values are not counted.

## AVG() Example

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

### Example

SELECT AVG(Price)  
FROM Products;

**Note:** NULL values are ignored.

## SUM() Example

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

### Example

SELECT SUM(Quantity)  
FROM OrderDetails;

**Note:** NULL values are ignored.

# **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

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

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

### LIKE Syntax

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

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

Here are some examples showing different LIKE operators with '%' and '\_' wildcards:

**LIKE Operator** **Description**

WHERE CustomerName LIKE 'a%' Finds any values that start with "a"

WHERE CustomerName LIKE '%a' Finds any values that end with "a"

WHERE CustomerName LIKE '%or% ' Finds any values that have "or" in any position

WHERE CustomerName LIKE '\_r%' Finds any values that have "r" in the second position

WHERE CustomerName LIKE 'a\_%' Finds any values that start with "a" and are at least 2 characters in length

WHERE CustomerName LIKE 'a\_\_%' Finds any values that start with "a" and are at least 3 characters in length

WHERE ContactName LIKE 'a%o' Finds any values that start with "a" and ends with "o"

## SQL LIKE Examples

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

### Example

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

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

### Example

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

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

### Example

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

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

### Example

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

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

### Example

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

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

### Example

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

The following SQL statement selects all customers with a CustomerName that does NOT start with "a":

### Example

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

# **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.

### Wildcard Characters in MS Access

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

### Wildcard Characters in SQL Server

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

All the wildcards can also be used in combinations!

Here are some examples showing different LIKE operators with '%' and '\_' wildcards:

|  |  |
| --- | --- |
| **LIKE Operator** | **Description** |
| WHERE CustomerName LIKE 'a%' | Finds any values that starts with "a" |
| WHERE CustomerName LIKE '%a' | Finds any values that ends with "a" |
| WHERE CustomerName LIKE '%or%' | Finds any values that have "or" in any position |
| WHERE CustomerName LIKE '\_r%' | Finds any values that have "r" in the second position |
| WHERE CustomerName LIKE 'a\_\_%' | Finds any values that starts with "a" and are at least 3 characters in length |
| WHERE ContactName LIKE 'a%o' | Finds any values that starts with "a" and ends with "o |

## Using the % Wildcard

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

### Example

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

The following SQL statement selects all customers with a City containing the pattern "es":

### Example

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

## Using the \_ Wildcard

The following SQL statement selects all customers with a City starting with any character, followed by "ondon":

### Example

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

The following SQL statement selects all customers with a City starting with "L", followed by any character, followed by "n", followed by any character, followed by "on":

### Example

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

## Using the [charlist] Wildcard

The following SQL statement selects all customers with a City starting with "b", "s", or "p":

### Example

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

The following SQL statement selects all customers with a City starting with "a", "b", or "c":

### Example

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

## Using the [!charlist] Wildcard

The two following SQL statements select all customers with a City NOT starting with "b", "s", or "p":

### Example

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

Or:

### Example

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

# **SQL IN Operator**

## The SQL IN Operator

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

The IN operator is a shorthand for multiple OR conditions.

### IN Syntax

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

or:

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

## IN Operator Examples

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

### Example

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

The following SQL statement selects all customers that are NOT located in "Germany", "France" or "UK":

### Example

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

The following SQL statement selects all customers that are from the same countries as the suppliers:

### Example

SELECT \* FROM Customers  
WHERE Country IN (SELECT Country FROM Suppliers);

# **SQL BETWEEN Operator**

## The SQL BETWEEN Operator

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

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

### BETWEEN Syntax

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

## BETWEEN Example

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

### Example

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

## NOT BETWEEN Example

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 Example

The following SQL statement selects all products with a price between 10 and 20. In addition; do not show products with a CategoryID of 1,2, or 3:

### Example

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

## BETWEEN Text Values Example

The following SQL statement selects all products with a ProductName 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 Example

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 Example

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';

# **SQL Aliases**

## 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.

### Alias Column Syntax

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

### Alias Table Syntax

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

## Alias for Columns Examples

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;

The following SQL statement creates two aliases, one for the CustomerName column and one for the ContactName column. **Note:** It requires double quotation marks or square brackets if the alias name contains spaces:

### Example

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

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:

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:

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

## Alias for Tables Example

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 Joins**

## SQL JOIN

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

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

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

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

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

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.

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

### Example

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

## Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

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

# **SQL INNER JOIN Keyword**

## SQL INNER JOIN Keyword

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

### INNER JOIN Syntax

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





## SQL INNER JOIN Example

The following SQL statement selects all orders with customer information:

### Example

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

**Note:** The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are records in the "Orders" table that do not have matches in "Customers", these orders will not be shown!

## JOIN Three Tables

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

### Example

SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName  
FROM ((Orders  
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)  
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);

# **SQL LEFT JOIN Keyword**

## SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (table1), and the 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.



## 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;

**Note:** The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

# **SQL RIGHT JOIN Keyword**

## 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.



## 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;

**Note:** The RIGHT JOIN keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders).

# **SQL FULL OUTER JOIN Keyword**

## SQL FULL OUTER JOIN Keyword

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

**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;



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

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:

**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**

## 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.

## 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;

# **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;

## SQL UNION Example

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

### Example

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

**Note:** If some customers or suppliers have the same city, each city will only be listed once, because UNION selects only distinct values. Use UNION ALL to also select duplicate values!

## SQL UNION ALL Example

The following SQL statement returns the cities (duplicate values also) from both the "Customers" and the "Suppliers" table:

### Example

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

## 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;

## SQL UNION ALL With WHERE

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

### Example

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

## Another UNION Example

The following SQL statement lists all customers and suppliers:

### Example

SELECT 'Customer' AS Type, ContactName, City, Country  
FROM Customers  
UNION  
SELECT 'Supplier', ContactName, City, Country  
FROM Suppliers;

Notice the "AS Type" above - it is an alias. [SQL Aliases](https://www.w3schools.com/sql/sql_alias.asp) are used to give a table or a column a temporary name. An alias only exists for the duration of the query. So, here we have created a temporary column named "Type", that list whether the contact person is a "Customer" or a "Supplier".

# **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);

## 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;

The following SQL statement lists the number of customers in each country, sorted high to low:

### Example

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

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;

# **SQL HAVING Clause**

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 *condition*ORDER BY *column\_name(s);*

## 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;

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;

The following SQL statement lists if the employees "Davolio" or "Fuller" have registered more than 25 orders:

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;

# **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);

## 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);

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);

# **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 <=).

## SQL ANY Examples

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);

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);

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);

## SQL ALL Examples

The following SQL statement lists ALL the product names:

### Example

SELECT ALL ProductName  
FROM Products  
WHERE TRUE;

The following SQL statement lists the ProductName if ALL the records in the OrderDetails table has Quantity equal to 10. This will of course return FALSE because the Quantity column has many different values (not only the value of 10):

### Example

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

# **SQL SELECT INTO Statement**

## 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;

# **SQL INSERT INTO SELECT Statement**

## 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;

## SQL INSERT INTO SELECT Examples

The following SQL statement copies "Suppliers" into "Customers" (the columns that are not filled with data, will contain NULL):

### Example

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

The following SQL statement copies "Suppliers" into "Customers" (fill all columns):

### Example

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

The following SQL statement copies only the German suppliers into "Customers":

### Example

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

# **SQL CASE Expression**

## 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;

## 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;

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);

# **SQL NULL Functions**

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

Look at the following "Products" table:

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

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;

# **SQL Stored Procedures for SQL Server**

## 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;

## Stored Procedure Example

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

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

CREATE PROCEDURE SelectAllCustomers  
AS  
SELECT \* FROM Customers  
GO;

Execute the stored procedure above as follows:

### Example

EXEC SelectAllCustomers;

ADVERTISEMENT

## 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**

## SQL Comments

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

**Note: The examples in this chapter will not work in Firefox and Microsoft Edge!**

Comments are not supported in Microsoft Access databases. Firefox and Microsoft Edge are using Microsoft Access database in our examples.

## 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;

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

### Example

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

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

### Example

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

## Multi-line Comments

Multi-line comments start with /\* and end with \*/.

Any text between /\* and \*/ will be ignored.

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

### Example

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

The following example uses a multi-line comment to ignore many statements:

### Example

/\*SELECT \* FROM Customers;  
SELECT \* FROM Products;  
SELECT \* FROM Orders;  
SELECT \* FROM Categories;\*/

SELECT \* FROM Suppliers;

To ignore just a part of a statement, also use the /\* \*/ comment.

The following example uses a comment to ignore part of a line:

### Example

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

The following example uses a comment to ignore part of a statement:

### Example

SELECT \* FROM Customers WHERE (CustomerName LIKE 'L%'  
OR CustomerName LIKE 'R%' /\*OR CustomerName LIKE 'S%'  
OR CustomerName LIKE 'T%'\*/ OR CustomerName LIKE 'W%')  
AND Country='USA'  
ORDER BY CustomerName;

# **SQL Operators**

## SQL Arithmetic Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** |  |
| + | Add |  |
| - | Subtract |  |
| \* | Multiply |  |
| / | Divide |  |
| % | Modulo |  |

## SQL Bitwise Operators

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

## SQL Comparison Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** |  |
| = | Equal to |  |
| > | Greater than |  |
| < | Less than |  |
| >= | Greater than or equal to |  |
| <= | Less than or equal to |  |
| <> | Not equal to |  |

## SQL Compound Operators

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

## SQL Logical Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** |  |
| ALL | TRUE if all of the subquery values meet the condition |  |
| AND | TRUE if all the conditions separated by AND is TRUE |  |
| ANY | TRUE if any of the subquery values meet the condition |  |
| BETWEEN | TRUE if the operand is within the range of comparisons |  |
| EXISTS | TRUE if the subquery returns one or more records |  |
| IN | TRUE if the operand is equal to one of a list of expressions |  |
| LIKE | TRUE if the operand matches a pattern |  |
| NOT | Displays a record if the condition(s) is NOT TRUE |  |
| OR | TRUE if any of the conditions separated by OR is TRUE |  |
| SOME | TRUE if any of the subquery values meet the condition |  |