Introduction to SQL

‹ Previous
Next ›

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;
```

Try it Yourself »

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.

‹ Previous Next ›

SQL Syntax

‹ Previous Next ›

Database Tables

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:

CustomerID	CustomerName	ContactName	Address	City	Posta
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958

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;
```

Try it Yourself »

In this tutorial we will teach you all about the different SQL statements.

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

‹ Previous Next ›

SQL SELECT Statement

‹ Previous Next ›

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;
```

Demo Database

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

CustomerID	CustomerName	ContactName	Address	City	Posta
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958

SELECT Column Example

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

Example

```
SELECT CustomerName, City FROM Customers;
```

Try it Yourself »

SELECT * Example

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

Example

```
SELECT * FROM Customers;
```

Try it Yourself »

Test Yourself With Exercises

Exercise:

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

```
* FROM Customers;
```

SQL SELECT DISTINCT Statement

‹ Previous Next ›

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;
```

Demo Database

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

CustomerID	CustomerName	ContactName	Address	City	Posta
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209
2	Ana Trujillo	Ana Trujillo	Avda. de la	México	05021

	Emparedados y helados		Constitución 2222	D.F.	
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958

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;
```

Try it Yourself »

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;
```

Try it Yourself »

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

Example

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

Try it Yourself »

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);
```

Try it Yourself »

SQL WHERE Clause

‹ Previous Next ›

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.!

Demo Database

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

CustomerID	CustomerName	ContactName	Address	City	Posta
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958

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';
```

Try it Yourself »

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

```
SELECT * FROM Customers
WHERE CustomerID=1;
```

Try it Yourself »

Operators in The WHERE Clause

The following operators can be used in the `WHERE` clause:

Operator	Description	Example
=	Equal	Try it
>	Greater than	Try it
<	Less than	Try it
>=	Greater than or equal	Try it
<=	Less than or equal	Try it
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=	Try it
BETWEEN	Between a certain range	Try it

LIKE	Search for a pattern	Try it
IN	To specify multiple possible values for a column	Try it

Test Yourself With Exercises

Exercise:

Select all records where the `City` column has the value "Berlin".

```
SELECT * FROM Customers
          =          ;
```

Submit Answer »

Start the Exercise

‹ Previous

Next ›

SQL AND, OR and NOT Operators

‹ Previous Next ›

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;
```

Demo Database

The table below shows the complete "Customers" table from the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim
7	Blondel père et	Frédérique	24, place	Strasbourg

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';
```

Try it Yourself »

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';
```

Try it Yourself »

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';
```

Try it Yourself »

NOT Example

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

Example

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

Try it Yourself »

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');
```

Try it Yourself »

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';
```

Try it Yourself »

Test Yourself With Exercises

Exercise:

Select all records where the `City` column has the value 'Berlin' and the `PostalCode` column has the value 12209.

```
 * FROM Customers
City = 'Berlin'
         = 12209;
```

Submit Answer »

Start the Exercise

SQL ORDER BY Keyword

‹ Previous Next ›

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;
```

Demo Database

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

CustomerID	CustomerName	ContactName	Address	City	Posta
1	Alfreds	Maria Anders	Obere Str. 57	Berlin	12209

	Futterkiste				
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958

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;
```

Try it Yourself »

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;
```

Try it Yourself »

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;
```

Try it Yourself »

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;
```

Try it Yourself »

Test Yourself With Exercises

Exercise:

Select all records from the `Customers` table, sort the result alphabetically by the column `City`.

```
SELECT * FROM Customers
              ;
```

Submit Answer »

Start the Exercise

‹ Previous

Next ›

SQL INSERT INTO Statement

‹ Previous Next ›

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" table in the Northwind sample database:

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

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');
```

Try it Yourself »

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

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

Did you notice that we did not insert any number into the CustomerID field?
The CustomerID column is an auto-increment 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');
```

Try it Yourself »

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

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

Test Yourself With Exercises

Exercise:

Insert a new record in the `Customers` table.

```
            Customers
CustomerName,
Address,
City,
PostalCode,
Country
```

```
'Hekkan Burger',
'Gateveien 15',
'Sandnes',
'4306',
'Norway' ;
```

Submit Answer »

Start the Exercise

‹ Previous

Next ›

SQL NULL Values

‹ Previous

Next ›

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_names
FROM table_name
```

IS NOT NULL Syntax

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

Demo Database

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

CustomerID	CustomerName	ContactName	Address	City	Posta
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958

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;
```

Try it Yourself »

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;
```

Try it Yourself »

Test Yourself With Exercises

Exercise:

Select all records from the `Customers` where the `PostalCode` column is empty.

```
SELECT * FROM Customers
WHERE                    ;
```

Submit Answer »

Start the Exercise

‹ Previous Next ›

SQL UPDATE Statement

‹ Previous Next ›

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" table in the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City	Posta
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958

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;
```

Try it Yourself »

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

CustomerID	CustomerName	ContactName	Address	City	Post
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	122(
2	Ana Trujillo	Ana Trujillo	Avda. de la	México	0502

	Emparedados y helados		Constitución 2222	D.F.	
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	0502
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-95

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';
```

Try it Yourself »

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

CustomerID	CustomerName	ContactName	Address	City	Post

1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	122(
2	Ana Trujillo Emparedados y helados	Juan	Avda. de la Constitución 2222	México D.F.	0502
3	Antonio Moreno Taquería	Juan	Mataderos 2312	México D.F.	0502
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-95

Update Warning!

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

Example

```
UPDATE Customers
SET ContactName='Juan';
```

Try it Yourself »

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

CustomerID	CustomerName	ContactName	Address	City	Post
1	Alfreds Futterkiste	Juan	Obere Str. 57	Frankfurt	122(
2	Ana Trujillo	Juan	Avda. de la	México	0502

	Emparedados y helados		Constitución 2222	D.F.	
3	Antonio Moreno Taquería	Juan	Mataderos 2312	México D.F.	0502
4	Around the Horn	Juan	120 Hanover Sq.	London	WA1
5	Berglunds snabbköp	Juan	Berguvsvägen 8	Luleå	S-95

Test Yourself With Exercises

Exercise:

Update the `City` column of all records in the `Customers` table.

```
   Customers
City = 'Oslo';
```

Submit Answer »

Start the Exercise

‹ Previous Next ›

SQL DELETE Statement

‹ Previous Next ›

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" table in the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City	Posta

1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958

SQL DELETE Example

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

Example

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

Try it Yourself »

The "Customers" table will now look like this:

CustomerID	CustomerName	ContactName	Address	City	Posta
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958

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;
```

Try it Yourself »

SQL TOP, LIMIT, FETCH FIRST or ROWNUM Clause

‹ Previous
Next ›

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_name
WHERE condition;
```

MySQL Syntax:

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

Oracle 12 Syntax:

```
SELECT column_name(s)
FROM table_name
ORDER 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

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

CustomerID	CustomerName	ContactName	Address	City	Posta
1	Alfreds	Maria Anders	Obere Str. 57	Berlin	12209

	Futterkiste				
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958

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;
```

Try it Yourself »

The following SQL statement shows the equivalent example for MySQL:

Example

```
SELECT * FROM Customers
LIMIT 3;
```

Try it Yourself »

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;
```

Try it Yourself »

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';
```

Try it Yourself »

The following SQL statement shows the equivalent example for MySQL:

Example

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

Try it Yourself »

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

‹ Previous Next ›

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;
```

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

MIN() Example

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

Example

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

Try it Yourself »

MAX() Example

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

Example

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

Try it Yourself »

Test Yourself With Exercises

Exercise:

Use the `MIN` function to select the record with the smallest value of the `Price` column.

```
SELECT
FROM Products;
```

Submit Answer »

Start the Exercise

‹ Previous

Next ›

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

‹ Previous Next ›

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;
```

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

COUNT() Example

The following SQL statement finds the number of products:

Example

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

Try it Yourself »

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;
```

Try it Yourself »

Note: NULL values are ignored.

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

SUM() Example

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

Example

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

Try it Yourself »

Note: NULL values are ignored.

Test Yourself With Exercises

SQL LIKE Operator

‹ Previous Next ›

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"

Demo Database

The table below shows the complete "Customers" table from the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F
3	Antonio Moreno	Antonio Moreno	Mataderos	México D.F

	Taquería		2312	
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim

SQL LIKE Examples

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

Example

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

Try it Yourself »

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

Example

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

Try it Yourself »

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

Example

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

Try it Yourself »

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%';
```

Try it Yourself »

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__%';
```

Try it Yourself »

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';
```

Try it Yourself »

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%';
```

Try it Yourself »

Test Yourself With Exercises

Exercise:

Select all records where the value of the `City` column starts with the letter "a".

```
SELECT * FROM Customers
                    ;
```

SQL Wildcards

‹ Previous Next ›

SQL Wildcard Characters

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

Wildcard characters are used with the `LIKE` 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"

Demo Database

The table below shows the complete "Customers" table from the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim
7	Blondel père et	Frédérique	24, place	Strasbourg

Using the % Wildcard

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

Example

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

Try it Yourself »

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

Example

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

Try it Yourself »

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';
```

Try it Yourself »

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';
```

Try it Yourself »

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]%';
```

Try it Yourself »

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]%';
```

Try it Yourself »

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]%';
```

Try it Yourself »

Or:

Example

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

Try it Yourself »

Test Yourself With Exercises

Exercise:

Select all records where the second letter of the `City` is an "a".

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

Submit Answer »

SQL IN Operator

‹ Previous Next ›

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);
```

Demo Database

10	Bottom-Dollar Marketse	Elizabeth Lincoln	23 Tsawassen Blvd.	Tsawassen
11	B's Beverages	Victoria Ashworth	Fauntleroy Circus	London
12	Cactus Comidas para llevar	Patricio Simpson	Cerrito 333	Buenos Air
13	Centro comercial Moctezuma	Francisco Chang	Sierras de Granada 9993	México D.F
14	Chop-suey Chinese	Yang Wang	Hauptstr. 29	Bern
15	Comércio Mineiro	Pedro Afonso	Av. dos Lusíadas, 23	São Paulo
16	Consolidated Holdings	Elizabeth Brown	Berkeley Gardens 12 Brewery	London

IN Operator Examples

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

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

Try it Yourself »

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');
```

Try it Yourself »

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);
```

Try it Yourself »

Test Yourself With Exercises

Exercise:

SQL BETWEEN Operator

‹ Previous Next ›

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;
```

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	18

				bags	
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	1	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	1	2	36 boxes	21.35

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;
```

Try it Yourself »

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;
```

Try it Yourself »

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);
```

Try it Yourself »

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;
```

Try it Yourself »

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;
```

Try it Yourself »

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;
```

Try it Yourself »

Sample Table

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

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

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#;
```

Try it Yourself »

OR:

Example

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

Try it Yourself »

SQL Aliases

‹ Previous Next ›

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;
```

Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalC
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1D

And a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10354	58	8	1996-11-14	3
10355	4	6	1996-11-15	1
10356	86	6	1996-11-18	2

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;
```

Try it Yourself »

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;
```

Try it Yourself »

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;
```

Try it Yourself »

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;
```

Try it Yourself »

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;
```

Try it Yourself »

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

Test Yourself With Exercises

Exercise:

When displaying the `Customers` table, make an ALIAS of the `PostalCode` column, the column should be called `Pno` instead.

```
SELECT CustomerName,
Address,
```

SQL Joins

‹ Previous Next ›

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.

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;
```

Try it Yourself »

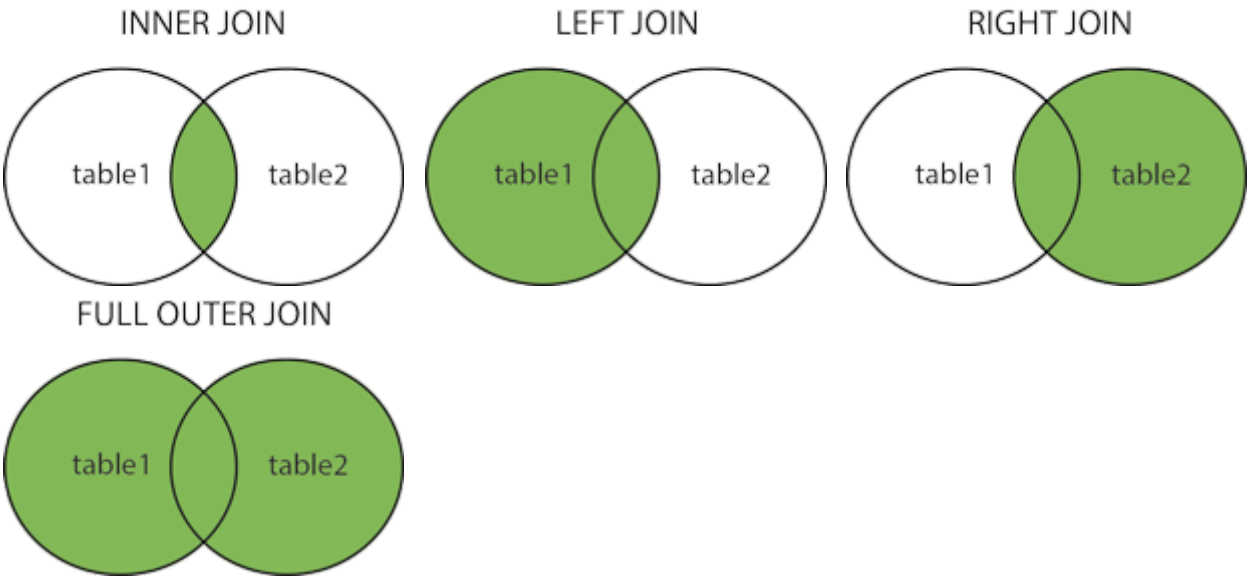
and it will produce something like this:

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



Test Yourself With Exercises

Exercise:

Insert the missing parts in the `JOIN` clause to join the two tables `Orders` and `Customers` , using the `CustomerID` field in both tables as the relationship between the two tables.

```
SELECT *
FROM Orders
LEFT JOIN Customers
                    =                    ;
```

Submit Answer »

Start the Exercise

‹ Previous

Next ›

SQL INNER JOIN Keyword

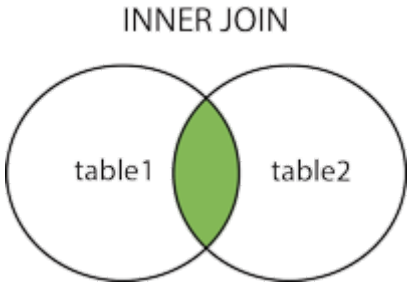
‹ Previous Next ›

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 table2
ON table1.column_name = table2.column_name;
```



Demo Database

In this tutorial 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
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

And a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalC
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023

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;
```

Try it Yourself »

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);
```

Try it Yourself »

Test Yourself With Exercises

Exercise:

Choose the correct `JOIN` clause to select all records from the two tables where there is a match in both tables.

```
SELECT *
FROM Orders

ON Orders.CustomerID=Customers.CustomerID;
```

Submit Answer »

Start the Exercise

‹ Previous

Next ›

SQL LEFT JOIN Keyword

‹ Previous Next ›

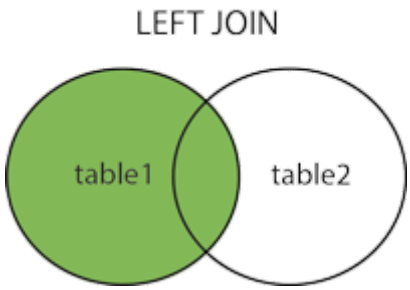
SQL LEFT JOIN Keyword

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 table2
ON table1.column_name = table2.column_name;
```

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



Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalC
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023

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;
```

Try it Yourself »

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

‹ Previous Next ›

SQL RIGHT JOIN Keyword

‹ Previous Next ›

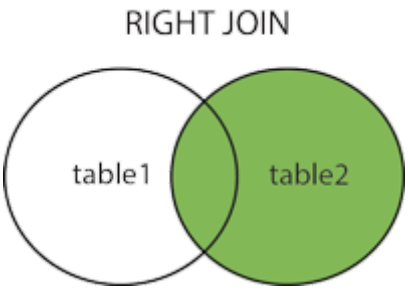
SQL RIGHT JOIN Keyword

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 table2
ON table1.column_name = table2.column_name;
```

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



Demo Database

In this tutorial 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
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

And a selection from the "Employees" table:

EmployeeID	LastName	FirstName	BirthDate	Photo
1	Davolio	Nancy	12/8/1968	EmpID1.pic
2	Fuller	Andrew	2/19/1952	EmpID2.pic
3	Leverling	Janet	8/30/1963	EmpID3.pic

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;
```

Try it Yourself »

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

Test Yourself With Exercises

Exercise:

Choose the correct `JOIN` clause to select all the records from the `Customers` table plus all the matches in the `Orders` table.

```
SELECT *
FROM Orders

ON Orders.CustomerID=Customers.CustomerID;
```

Submit Answer »

Start the Exercise

SQL FULL OUTER JOIN Keyword

‹ Previous Next ›

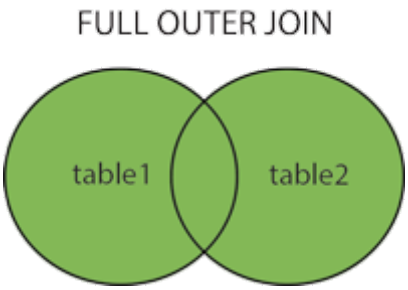
SQL FULL OUTER JOIN Keyword

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 table2
ON table1.column_name = table2.column_name
WHERE condition;
```



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

Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalC
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023

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.

‹ Previous Next ›

SQL Self Join

‹ Previous Next ›

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

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalC
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209

2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023

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;
```

Try it Yourself »

‹ Previous

Next ›

SQL UNION Operator

‹ Previous Next ›

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;
```

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

Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalC
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023

And a selection from the "Suppliers" table:

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

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;
```

Try it Yourself »

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;
```

Try it Yourself »

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;
```

Try it Yourself »

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;
```

Try it Yourself »

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;
```

Try it Yourself »

Notice the "AS Type" above - it is an alias. SQL Aliases 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".

‹ Previous Next ›

SQL GROUP BY Statement

‹ Previous Next ›

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	Posta

1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958

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;
```

Try it Yourself »

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;
```

Try it Yourself »

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;
```

Try it Yourself »

Test Yourself With Exercises

Exercise:

List the number of customers in each country.

```
SELECT       (CustomerID),
Country
FROM Customers
                  ;
```

Submit Answer »

Start the Exercise

SQL HAVING Clause

‹ Previous Next ›

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);
```

Demo Database

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

CustomerID	CustomerName	ContactName	Address	City	Posta
1	Alfreds	Maria Anders	Obere Str. 57	Berlin	12209

	Futterkiste				
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958

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;
```

Try it Yourself »

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;
```

Try it Yourself »

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;
```

Try it Yourself »

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;
```

Try it Yourself »

SQL EXISTS Operator

‹ Previous Next ›

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	18

				bags	
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
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	London	EC1 4SD
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104
4	Tokyo Traders	Yoshi Nagase	9-8 Sekimai Musashino-shi	Tokyo	100

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);
```

Try it Yourself »

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);
```

Try it Yourself »

SQL ANY and ALL Operators

‹ Previous

Next ›

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 <=).

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	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 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
```

Try it Yourself »

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);
```

Try it Yourself »

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);
```

Try it Yourself »

The following SQL statement lists ALL the product names:

Example

```
SELECT ALL ProductName
FROM Products
WHERE TRUE;
```

Try it Yourself »

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);
```

Try it Yourself »

‹ Previous Next ›

SQL CASE Statement

‹ Previous

Next ›

The SQL CASE Statement

The `CASE` statement 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 s

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;
```

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);
```

Try it Yourself »

‹ Previous Next ›

SQL CREATE DATABASE Statement

‹ Previous Next ›

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;`

Test Yourself With Exercises

Exercise:

Write the correct SQL statement to create a new database called `testDB`.

;

Submit Answer »

Start the Exercise

‹ Previous

Next ›

SQL DROP DATABASE Statement

‹ Previous Next ›

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;
```

SQL CREATE TABLE Statement

‹ Previous

Next ›

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.

SQL CREATE TABLE Example

Example

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

Try it Yourself »

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 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

‹ Previous

Next ›

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;
```

Test Yourself With Exercises

Exercise:

Write the correct SQL statement to delete a table called `Persons`.

```
Persons;
```

Submit Answer »

Start the Exercise

SQL ALTER TABLE Statement

‹ Previous

Next ›

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;
```

Try it Yourself »

ALTER TABLE - ALTER/MODIFY COLUMN

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.

```
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.

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

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

Test Yourself With Exercises

Exercise:

Add a column of type `DATE` called `Birthday`.

```
Persons
;
```

Submit Answer »

Start the Exercise

SQL Constraints

‹ Previous

Next ›

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.

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` - Ensures that a column cannot have a NULL value
- `UNIQUE` - Ensures that all values in a column are different
- `PRIMARY KEY` - A combination of a `NOT NULL` and `UNIQUE`. Uniquely identifies each row in a table
- `FOREIGN KEY` - Prevents actions that would destroy links between tables
- `CHECK` - Ensures that the values in a column satisfies a specific condition
- `DEFAULT` - Sets a default value for a column if no value is specified
- `CREATE INDEX` - Used to create and retrieve data from the database very quickly

‹ Previous Next ›

SQL NOT NULL Constraint

‹ Previous Next ›

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
);
```

Try it Yourself »

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:

```
ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

‹ Previous Next ›

SQL UNIQUE Constraint

‹ Previous Next ›

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

‹ Previous Next ›

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;
```

‹ Previous Next ›

SQL FOREIGN KEY Constraint

‹ Previous Next ›

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` 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;
```

‹ Previous Next ›

SQL CHECK Constraint

‹ Previous Next ›

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

‹ Previous

Next ›

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()`:

```
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;
```

‹ Previous Next ›

SQL CREATE INDEX Statement

‹ Previous Next ›

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_name
DROP INDEX index_name;
```

‹ Previous Next ›

SQL AUTO INCREMENT Field

‹ Previous

Next ›

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.

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.

SQL Working With Dates

‹ Previous Next ›

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!

‹ Previous Next ›

SQL Views

‹ Previous Next ›

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';
```

Try it Yourself »

We can query the view above as follows:

Example

```
SELECT * FROM [Brazil Customers];
```

Try it Yourself »

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);
```

Try it Yourself »

We can query the view above as follows:

Example

```
SELECT * FROM [Products Above Average Price];
```

Try it Yourself »

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';
```

Try it Yourself »

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];
```

Try it Yourself »

‹ Previous Next ›

SQL Data Types for MySQL, SQL Server, and MS Access

‹ Previous Next ›

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 column 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	5-17 bytes

	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	
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

❮ Previous Next ❯

SQL Keywords Reference

‹ Previous Next ›

This SQL keywords reference contains the reserved words in SQL.

SQL Keywords

Keyword	Description
ADD	Adds a column in an existing table
ADD CONSTRAINT	Adds a constraint after a table is already created
ALL	Returns true if all of the subquery values meet the condition
ALTER	Adds, deletes, or modifies columns in a table, or changes the data type of a column in a table
ALTER COLUMN	Changes the data type of a column in a table
ALTER TABLE	Adds, deletes, or modifies columns in a table
AND	Only includes rows where both conditions is true
ANY	Returns true if any of the subquery values meet the condition
AS	Renames a column or table with an alias
ASC	Sorts the result set in ascending order

BACKUP DATABASE	Creates a back up of an existing database
BETWEEN	Selects values within a given range
CASE	Creates different outputs based on conditions
CHECK	A constraint that limits the value that can be placed in a column
COLUMN	Changes the data type of a column or deletes a column in a table
CONSTRAINT	Adds or deletes a constraint
CREATE	Creates a database, index, view, table, or procedure
CREATE DATABASE	Creates a new SQL database
CREATE INDEX	Creates an index on a table (allows duplicate values)
CREATE OR REPLACE VIEW	Updates a view
CREATE TABLE	Creates a new table in the database
CREATE PROCEDURE	Creates a stored procedure
CREATE UNIQUE INDEX	Creates a unique index on a table (no duplicate values)
CREATE VIEW	Creates a view based on the result set of a SELECT statement
DATABASE	Creates or deletes an SQL database
DEFAULT	A constraint that provides a default value for a column
DELETE	Deletes rows from a table
DESC	Sorts the result set in descending order
DISTINCT	Selects only distinct (different) values
DROP	Deletes a column, constraint, database, index, table, or view
DROP COLUMN	Deletes a column in a table
DROP CONSTRAINT	Deletes a UNIQUE, PRIMARY KEY, FOREIGN KEY, or CHECK constraint

DROP DATABASE	Deletes an existing SQL database
DROP DEFAULT	Deletes a DEFAULT constraint
DROP INDEX	Deletes an index in a table
DROP TABLE	Deletes an existing table in the database
DROP VIEW	Deletes a view
EXEC	Executes a stored procedure
EXISTS	Tests for the existence of any record in a subquery
FOREIGN KEY	A constraint that is a key used to link two tables together
FROM	Specifies which table to select or delete data from
FULL OUTER JOIN	Returns all rows when there is a match in either left table or right table
GROUP BY	Groups the result set (used with aggregate functions: COUNT, MAX, MIN, SUM, AVG)
HAVING	Used instead of WHERE with aggregate functions
IN	Allows you to specify multiple values in a WHERE clause
INDEX	Creates or deletes an index in a table
INNER JOIN	Returns rows that have matching values in both tables
INSERT INTO	Inserts new rows in a table
INSERT INTO SELECT	Copies data from one table into another table
IS NULL	Tests for empty values
IS NOT NULL	Tests for non-empty values
JOIN	Joins tables
LEFT JOIN	Returns all rows from the left table, and the matching rows from the right table
LIKE	Searches for a specified pattern in a column
LIMIT	Specifies the number of records to return in the result set
NOT	Only includes rows where a condition is not true

NOT NULL	A constraint that enforces a column to not accept NULL values
OR	Includes rows where either condition is true
ORDER BY	Sorts the result set in ascending or descending order
OUTER JOIN	Returns all rows when there is a match in either left table or right table
PRIMARY KEY	A constraint that uniquely identifies each record in a database table
PROCEDURE	A stored procedure
RIGHT JOIN	Returns all rows from the right table, and the matching rows from the left table
ROWNUM	Specifies the number of records to return in the result set
SELECT	Selects data from a database
SELECT DISTINCT	Selects only distinct (different) values
SELECT INTO	Copies data from one table into a new table
SELECT TOP	Specifies the number of records to return in the result set
SET	Specifies which columns and values that should be updated in a table
TABLE	Creates a table, or adds, deletes, or modifies columns in a table, or deletes a table or data inside a table
TOP	Specifies the number of records to return in the result set
TRUNCATE TABLE	Deletes the data inside a table, but not the table itself
UNION	Combines the result set of two or more SELECT statements (only distinct values)
UNION ALL	Combines the result set of two or more SELECT statements (allows duplicate values)
UNIQUE	A constraint that ensures that all values in a column are unique
UPDATE	Updates existing rows in a table
VALUES	Specifies the values of an INSERT INTO statement

VIEW	Creates, updates, or deletes a view
WHERE	Filters a result set to include only records that fulfill a specified condition

‹ Previous

Next ›

MySQL Functions

‹ Previous Next ›

MySQL has many built-in functions.

This reference contains string, numeric, date, and some advanced functions in MySQL.

MySQL String Functions

Function	Description
ASCII	Returns the ASCII value for the specific character
CHAR_LENGTH	Returns the length of a string (in characters)
CHARACTER_LENGTH	Returns the length of a string (in characters)
CONCAT	Adds two or more expressions together
CONCAT_WS	Adds two or more expressions together with a separator
FIELD	Returns the index position of a value in a list of values
FIND_IN_SET	Returns the position of a string within a list of strings
FORMAT	Formats a number to a format like "#,###,###.##", rounded to a specified number of decimal places
INSERT	Inserts a string within a string at the specified position and for a certain number of characters

INSTR	Returns the position of the first occurrence of a string in another string
LCASE	Converts a string to lower-case
LEFT	Extracts a number of characters from a string (starting from left)
LENGTH	Returns the length of a string (in bytes)
LOCATE	Returns the position of the first occurrence of a substring in a string
LOWER	Converts a string to lower-case
LPAD	Left-pads a string with another string, to a certain length
LTRIM	Removes leading spaces from a string
MID	Extracts a substring from a string (starting at any position)
POSITION	Returns the position of the first occurrence of a substring in a string
REPEAT	Repeats a string as many times as specified
REPLACE	Replaces all occurrences of a substring within a string, with a new substring
REVERSE	Reverses a string and returns the result
RIGHT	Extracts a number of characters from a string (starting from right)
RPAD	Right-pads a string with another string, to a certain length
RTRIM	Removes trailing spaces from a string
SPACE	Returns a string of the specified number of space characters
STRCMP	Compares two strings
SUBSTR	Extracts a substring from a string (starting at any position)
SUBSTRING	Extracts a substring from a string (starting at any position)
SUBSTRING_INDEX	Returns a substring of a string before a specified number

	of delimiter occurs
TRIM	Removes leading and trailing spaces from a string
UCASE	Converts a string to upper-case
UPPER	Converts a string to upper-case

MySQL Numeric Functions

Function	Description
ABS	Returns the absolute value of a number
ACOS	Returns the arc cosine of a number
ASIN	Returns the arc sine of a number
ATAN	Returns the arc tangent of one or two numbers
ATAN2	Returns the arc tangent of two numbers
AVG	Returns the average value of an expression
CEIL	Returns the smallest integer value that is >= to a number
CEILING	Returns the smallest integer value that is >= to a number
COS	Returns the cosine of a number
COT	Returns the cotangent of a number
COUNT	Returns the number of records returned by a select query
DEGREES	Converts a value in radians to degrees
DIV	Used for integer division
EXP	Returns e raised to the power of a specified number

FLOOR	Returns the largest integer value that is <= to a number
GREATEST	Returns the greatest value of the list of arguments
LEAST	Returns the smallest value of the list of arguments
LN	Returns the natural logarithm of a number
LOG	Returns the natural logarithm of a number, or the logarithm of a number to a specified base
LOG10	Returns the natural logarithm of a number to base 10
LOG2	Returns the natural logarithm of a number to base 2
MAX	Returns the maximum value in a set of values
MIN	Returns the minimum value in a set of values
MOD	Returns the remainder of a number divided by another number
PI	Returns the value of PI
POW	Returns the value of a number raised to the power of another number
POWER	Returns the value of a number raised to the power of another number
RADIANS	Converts a degree value into radians
RAND	Returns a random number
ROUND	Rounds a number to a specified number of decimal places
SIGN	Returns the sign of a number
SIN	Returns the sine of a number
SQRT	Returns the square root of a number
SUM	Calculates the sum of a set of values
TAN	Returns the tangent of a number
TRUNCATE	Truncates a number to the specified number of decimal places

MySQL Date Functions

Function	Description
ADDDATE	Adds a time/date interval to a date and then returns the date
ADDTIME	Adds a time interval to a time/datetime and then returns the time/datetime
CURDATE	Returns the current date
CURRENT_DATE	Returns the current date
CURRENT_TIME	Returns the current time
CURRENT_TIMESTAMP	Returns the current date and time
CURTIME	Returns the current time
DATE	Extracts the date part from a datetime expression
DATEDIFF	Returns the number of days between two date values
DATE_ADD	Adds a time/date interval to a date and then returns the date
DATE_FORMAT	Formats a date
DATE_SUB	Subtracts a time/date interval from a date and then returns the date
DAY	Returns the day of the month for a given date
DAYNAME	Returns the weekday name for a given date
DAYOFMONTH	Returns the day of the month for a given date
DAYOFWEEK	Returns the weekday index for a given date
DAYOFYEAR	Returns the day of the year for a given date
EXTRACT	Extracts a part from a given date
FROM_DAYS	Returns a date from a numeric datevalue
HOUR	Returns the hour part for a given date
LAST_DAY	Extracts the last day of the month for a given date

LOCALTIME	Returns the current date and time
LOCALTIMESTAMP	Returns the current date and time
MAKEDATE	Creates and returns a date based on a year and a number of days value
MAKETIME	Creates and returns a time based on an hour, minute, and second value
MICROSECOND	Returns the microsecond part of a time/datetime
MINUTE	Returns the minute part of a time/datetime
MONTH	Returns the month part for a given date
MONTHNAME	Returns the name of the month for a given date
NOW	Returns the current date and time
PERIOD_ADD	Adds a specified number of months to a period
PERIOD_DIFF	Returns the difference between two periods
QUARTER	Returns the quarter of the year for a given date value
SECOND	Returns the seconds part of a time/datetime
SEC_TO_TIME	Returns a time value based on the specified seconds
STR_TO_DATE	Returns a date based on a string and a format
SUBDATE	Subtracts a time/date interval from a date and then returns the date
SUBTIME	Subtracts a time interval from a datetime and then returns the time/datetime
SYSDATE	Returns the current date and time
TIME	Extracts the time part from a given time/datetime
TIME_FORMAT	Formats a time by a specified format
TIME_TO_SEC	Converts a time value into seconds
TIMEDIFF	Returns the difference between two time/datetime expressions
TIMESTAMP	Returns a datetime value based on a date or datetime value

TO_DAYS	Returns the number of days between a date and date "0000-00-00"
WEEK	Returns the week number for a given date
WEEKDAY	Returns the weekday number for a given date
WEEKOFYEAR	Returns the week number for a given date
YEAR	Returns the year part for a given date
YEARWEEK	Returns the year and week number for a given date

MySQL Advanced Functions

Function	Description
BIN	Returns a binary representation of a number
BINARY	Converts a value to a binary string
CASE	Goes through conditions and return a value when the first condition is met
CAST	Converts a value (of any type) into a specified datatype
COALESCE	Returns the first non-null value in a list
CONNECTION_ID	Returns the unique connection ID for the current connection
CONV	Converts a number from one numeric base system to another
CONVERT	Converts a value into the specified datatype or character set
CURRENT_USER	Returns the user name and host name for the MySQL account that the server used to authenticate the current client
DATABASE	Returns the name of the current database
IF	Returns a value if a condition is TRUE, or another value if a condition is FALSE
IFNULL	Return a specified value if the expression is NULL, otherwise return the expression
ISNULL	Returns 1 or 0 depending on whether an expression is NULL

LAST_INSERT_ID	Returns the AUTO_INCREMENT id of the last row that has been inserted or updated in a table
NULLIF	Compares two expressions and returns NULL if they are equal. Otherwise, the first expression is returned
SESSION_USER	Returns the current MySQL user name and host name
SYSTEM_USER	Returns the current MySQL user name and host name
USER	Returns the current MySQL user name and host name
VERSION	Returns the current version of the MySQL database

‹ Previous Next ›