SQL Tutorial

December 3, 2019

Contents

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

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

1 Introduction to SQL

SQL is a standard language for accessing and manipulating databases.

1.1 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

1.2 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

1.3 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!

1.4 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

1.5 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:

```
SELECT * FROM Customers;
```

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

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

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

2 SQL Syntax

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.

2.1 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:

```
SELECT * FROM Customers;
```

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

2.2 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.

2.3 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.

2.4 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

3 SQL SELECT Statement

3.1 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;
```

3.2 SELECT Column Example

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

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

3.3 SELECT * Example

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

```
SELECT * FROM Customers;
```

4 SQL SELECT DISTINCT Statement

4.1 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;
```

4.2 SELECT Example Without DISTINCT

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

```
SELECT Country FROM Customers;
```

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

4.3 SELECT DISTINCT Examples

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

```
SELECT DISTINCT Country FROM Customers;
```

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

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

Here is the workaround for MS Access:

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

5 SQL WHERE Clause

The WHERE clause is used to filter records.

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

WHERE Syntax :

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

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

5.1 WHERE Clause Example

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

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

5.2 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:

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

5.3 Operators in The WHERE Clause

- = - Equal
- > - Greater than
- < - Less than
- >= - Greater than or equal
- <= - Less than or equal
- <> - Not equal. Note: In some versions of SQL this operator may be written as !=
- BETWEEN - Between a certain range
- LIKE - Search for a pattern
- IN - To specify multiple possible values for a column

6 SQL AND, OR and NOT Operators

6.1 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;
```

6.2 AND Example

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

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

6.3 OR Example

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

```
SELECT * FROM Customers
WHERE City='Berlin' OR City='Munchen';
```

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

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

6.4 NOT Example

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

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

6.5 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 "Munchen" (use parenthesis to form complex expressions):

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

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

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

7 SQL ORDER BY Keyword

7.1 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;
```

7.2 ORDER BY Example

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

```
SELECT * FROM Customers
ORDER BY Country;
```

7.3 ORDER BY DESC Example

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

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

7.4 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:

```
SELECT * FROM Customers
ORDER BY Country, CustomerName;
```

7.5 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:

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

8 SQL INSERT INTO Statement

8.1 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.

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

```
INSERT INTO table_name (column1, column2, column3,
   ...)
VALUES (value1, value2, value3, ...);
```

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

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

8.2 INSERT INTO Example

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

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

Did you notice that we did not insert any number into the CustomerID field? The CustomerID column is an auto-increment field and will be generated automatically when a new record is inserted into the table.

8.3 Insert Data Only in Specified Columns

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

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

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

9 SQL NULL Values

9.1 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!

9.2 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
WHERE column_name IS NULL;
```

IS NOT NULL Syntax :

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

9.3 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:

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

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

9.4 The IS NOT NULL Operator

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

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

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

10 SQL UPDATE Statement