SQL TUTORIAL

Version 1.9, Date 5.3.2014

B. Studer based on Source: http://www.w3schools.com

Introduction to SQL

What is SQL?

- SQL stands for **S**tructured **Q**uery **L**anguage
- SQL allows you to access a database
- SQL is an ANSI standard language
- SQL can execute queries against a database
- SQL can retrieve data from a database

- SQL can insert new records in a database
- SQL can delete records from a database
- SQL can update records in a database
- SQL is easy to learn

SQL is a Standard

SQL is an ANSI (American National Standards Institute) standard for accessing database systems. SQL statements are used to retrieve and update data in a database.

SQL works with database programs like MS Access, DB2, Informix, MS SQL Server, Oracle, Sybase, etc.

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

Database Tables

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

Below is an example of a table called "Persons":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

The table above contains three records (one for each person) and four columns (LastName, FirstName, Address, and City).

SQL Queries

With SQL, we can query a database and have a result set returned.

A query like this:

```
SELECT LastName FROM Persons
```

Gives a result set like this:

LastName
Hansen
Svendson
Pettersen

Note: Some database systems require a semicolon at the end of the SQL statement. We don't use the semicolon in our tutorials.

SQL Data Manipulation Language (DML)

SQL (Structured Query Language) is a syntax for executing queries. But the SQL language also includes a syntax to update, insert, and delete records.

These query and update commands together form the Data Manipulation Language (DML) part of SQL:

- SELECT - extracts data from a database table
- UPDATE - updates data in a database table
- DELETE - deletes data from a database table
- INSERT INTO - inserts new data into a database table

SQL Data Definition Language (DDL)

The Data Definition Language (DDL) part of SQL permits database tables to be created or deleted. We can also define indexes (keys), specify links between tables, and impose constraints between database tables.

The most important DDL statements in SQL are:

- CREATE TABLE - creates a new database table
- ALTER TABLE - alters (changes) a database table
- DROP TABLE - deletes a database table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

The SELECT Statement

The SELECT statement is used to select data from a table. The tabular result is stored in a result table (called the result set).

Syntax

```
SELECT column_name(s)
FROM table_name
```

Select Some Columns

To select the columns named "LastName" and "FirstName", use a SELECT statement like this:

```
SELECT LastName,FirstName FROM Persons
```

"Persons" table

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Result

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Select All Columns

To select all columns from the "Persons" table, use a * symbol instead of column names, like this:

```
SELECT * FROM Persons
```

Result

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

The Result Set

The result from a SQL query is stored in a result set. Most database software systems allow navigation of the result set with programming functions, like: Move-To-First-Record, Get-Record-Content, Move-To-Next-Record, etc.

Semicolon after SQL Statements?

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

Some SQL tutorials ends each SQL statement with a semicolon. Is this necessary? We are using MS Access and SQL Server 2000 and we do not have to put a semicolon after each SQL statement, but some database programs force you to use it.

SQL The WHERE Clause

The WHERE clause is used to specify a selection criterion.

The WHERE Clause

To conditionally select data from a table, a WHERE clause can be added to the SELECT statement.

Syntax

```
SELECT column FROM table
WHERE column operator value
```

With the WHERE clause, the following operators can be used:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern

Note: In some versions of SQL the <> operator may be written as !=

Using the WHERE Clause

To select only the persons living in the city "Sandnes", we add a WHERE clause to the SELECT statement:

```
SELECT * FROM Persons
WHERE City='Sandnes'
```

"Persons" table

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980
Pettersen	Kari	Storgt 20	Stavanger	1960

Result

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980

Using Quotes

Note that we have used single quotes around the conditional values in the examples.

SQL uses single quotes around text values (most database systems will also accept double quotes). Numeric values should not be enclosed in quotes.

For text values:

```
This is correct:
SELECT * FROM Persons WHERE FirstName='Tove'
This is wrong:
SELECT * FROM Persons WHERE FirstName=Tove
```

For numeric values:

```
This is correct:
SELECT * FROM Persons WHERE Year>1965
This is wrong:
SELECT * FROM Persons WHERE Year>'1965'
```

The LIKE Condition

The LIKE condition is used to specify a search for a pattern in a column.

Syntax

```
SELECT column FROM table
WHERE column LIKE pattern
```

A "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

Using LIKE

The following SQL statement will return persons with first names that start with an 'O':

```
SELECT * FROM Persons
WHERE FirstName LIKE 'O%'
```

SQL And & Or

AND & OR

AND and OR join two or more conditions in a WHERE clause.

The AND operator displays a row if ALL conditions listed are true. The OR operator displays a row if ANY of the conditions listed are true.

Original Table (used in the examples)

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

Example

Use AND to display each person with the first name equal to "Tove", and the last name equal to "Svendson":

```
SELECT * FROM Persons
WHERE FirstName='Tove'
AND LastName='Svendson'
```

Result:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes

Example

Use OR to display each person with the first name equal to "Tove", or the last name equal to "Svendson":

```
SELECT * FROM Persons
WHERE firstname='Tove'
OR lastname='Svendson'
```

Result:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

Example

You can also combine AND and OR (use parentheses to form complex expressions):

```
SELECT * FROM Persons WHERE
(FirstName='Tove' OR FirstName='Stephen')
AND LastName='Svendson'
```

Result:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

SQL Between...And

BETWEEN ... AND

The BETWEEN ... AND operator selects a range of data between two values. These values can be numbers, text, or dates.

```
SELECT column_name FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

Original Table (used in the examples)

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Nordmann	Anna	Neset 18	Sandnes
Pettersen	Kari	Storgt 20	Stavanger
Svendson	Tove	Borgvn 23	Sandnes

Example 1

To display the persons alphabetically between (and including) "Hansen" and exclusive "Pettersen", use the following SQL:

```
SELECT * FROM Persons WHERE LastName
BETWEEN 'Hansen' AND 'Pettersen'
```

Result:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Nordmann	Anna	Neset 18	Sandnes

IMPORTANT! The BETWEEN...AND operator is treated differently in different databases. With some databases a person with the LastName of "Hansen" or "Pettersen" will not be listed (BETWEEN..AND only selects fields that are between and excluding the test values). With some databases a person with the last name of "Hansen" or "Pettersen" will be listed (BETWEEN..AND selects fields that are between and including the test values). With other databases a person with the last name of "Hansen" will be listed, but "Pettersen" will not be listed (BETWEEN..AND selects fields between the test values, including the first test value

and excluding the last test value). Therefore: Check how your database treats the BETWEEN....AND operator!

Example 2

To display the persons outside the range used in the previous example, use the NOT operator:

```
SELECT * FROM Persons WHERE LastName
NOT BETWEEN 'Hansen' AND 'Pettersen'
```

Result:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Svendson	Tove	Borgvn 23	Sandnes

SQL Select Distinct

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

The DISTINCT Keyword

The SQL SELECT statement returns information from table columns. But what if we only want to select distinct elements?

With SQL, all we need to do is to add a DISTINCT keyword to the SELECT statement with the following syntax:

```
SELECT DISTINCT column-name(s) FROM table-name
```

Example: Select Companies from Order Table

Example: Simple Table of Purchase Orders:

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

This SQL statement:

```
SELECT Company FROM Orders
```

Will return this result:

Company
Sega
W3Schools
Trio
W3Schools

Note that the company W3Schools is listed twice in the result. Sometimes we don't want that.

Example: Select Distinct Companies from Orders

This SQL statement:

```
SELECT DISTINCT Company FROM Orders
```

Will return this result:

Company
Sega
W3Schools
Trio

Now the company W3Schools is listed only once in the result. Sometimes that is what we want.

SQL Order By

The ORDER BY keywords are used to sort-order the result.

Sort the Rows

The ORDER BY clause is used to sort the rows.

Orders:

Company	OrderNumber
Sega	3412
ABC Shop	5678
W3Schools	2312
W3Schools	6798

Example

To display the companies in alphabetical order:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company
```

Result:

Company	OrderNumber
ABC Shop	5678
Sega	3412
W3Schools	6798
W3Schools	2312

Example

To display the companies in alphabetical order AND the orders in numerical order:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company, OrderNumber
```

Result:

Company	OrderNumber
ABC Shop	5678
Sega	3412
W3Schools	2312
W3Schools	6798

Example

To display the companies in reverse alphabetical order:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company DESC
```

Result:

Company	OrderNumber
W3Schools	6798
W3Schools	2312
Sega	3412
ABC Shop	5678

SQL The INSERT INTO Statement

The INSERT INTO Statement

The INSERT INTO statement is used to insert new rows into a table.

Syntax

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

You can also specify the columns for which you want to insert data:

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

Insert a New Row

This "Persons" table:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger

And this SQL statement:

```
INSERT INTO Persons
VALUES ('Hetland', 'Camilla', 'Hagabakka 24', 'Sandnes')
```

Will give this result:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

Insert Data in Specified Columns

This "Persons" table:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

And This SQL statement:

```
INSERT INTO Persons (LastName, Address)
VALUES ('Rasmussen', 'Storgt 67')
```

Will give this result:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes
Rasmussen		Storgt 67	

SQL The Update Statement

The Update Statement

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

Syntax

```
UPDATE table_name
SET column_name = new_value
WHERE column_name = some_value
```

Person:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen		Storgt 67	

Update one Column in a Row

We want to add a first name to the person with a last name of "Rasmussen":

```
UPDATE Person SET FirstName = 'Nina'
WHERE LastName = 'Rasmussen'
```

Result:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Storgt 67	

Update several Columns in a Row

We want to change the address and add the name of the city:

```
UPDATE Person
SET Address = 'Stien 12', City = 'Stavanger'
WHERE LastName = 'Rasmussen'
```

Result:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

SQL The Delete Statement

The Delete Statement

The DELETE statement is used to delete rows in a table.

Syntax

```
DELETE FROM table_name
WHERE column_name = some_value
```

Person:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

Delete a Row

"Nina Rasmussen" is going to be deleted:

```
DELETE FROM Person WHERE LastName = 'Rasmussen'
```

Result

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger

Delete All Rows

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
or
DELETE * FROM table_name
```

SQL Count Functions

SQL has built-in functions for counting database records.

Count Function Syntax

The syntax for the built-in COUNT functions is:

```
SELECT COUNT(column) FROM table
```

Function COUNT(*)

The COUNT(*) function returns the number of selected rows in a selection.

With this "Persons" Table:

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

This example returns the number of rows in the table:

```
SELECT COUNT(*) FROM Persons
```

Result:

```
3
```

This example returns the number of persons that are older than 20 years:

```
SELECT COUNT(*) FROM Persons WHERE Age>20
```

Result:

```
2
```

Function COUNT(column)

The COUNT(column) function returns the number of rows without a NULL value in the specified column.

With this "Persons" Table:

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	

This example finds the number of persons with a value in the "Age" field in the "Persons" table:

```
SELECT COUNT(Age) FROM Persons
```

Result:

```
2
```

The COUNT(column) function is handy for finding columns without a value. Note that the result is one less than the number of rows in the original table because one of the persons does not have an age value stored.

COUNT DISTINCT

Note: The following example works with ORACLE and Microsoft SQL server but not with Microsoft Access.

The keyword DISTINCT and COUNT can be used together to count the number of distinct results.

The syntax is:

```
SELECT COUNT(DISTINCT column(s)) FROM table
```

With this "Orders" Table:

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

This SQL statement:

```
SELECT COUNT(distinct Company) FROM Orders
```

Will return this result:

```
3
```

SQL Functions

SQL has a lot of built-in functions for counting and calculations.

Function Syntax

The syntax for built-in SQL functions is:

```
SELECT function(column) FROM table
```

Original Table (used in the examples)

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

Function AVG(column)

The AVG function returns the average value of a column in a selection. NULL values are not included in the calculation.

Example

This example returns the average age of the persons in the "Persons" table:

```
SELECT AVG(Age) FROM Persons
```

Result

```
32.67
```

Example

This example returns the average age for persons that are older than 20 years:

```
SELECT AVG(Age) FROM Persons WHERE Age>20
```

Result

```
39.5
```

Function MAX(column)

The MAX function returns the highest value in a column. NULL values are not included in the calculation.

Example

```
SELECT MAX(Age) FROM Persons
```

Result:

```
45
```

Function MIN(column)

The MIN function returns the lowest value in a column. NULL values are not included in the calculation.

Example

```
SELECT MIN(Age) FROM Persons
```

Result:

```
19
```

Note: The MIN and MAX functions can also be used on text columns, to find the highest or lowest value in alphabetical order.

Function SUM(column)

The SUM function returns the total sum of a column in a given selection. NULL values are not included in the calculation.

Example

This example returns the sum of all ages in the "person" table:

```
SELECT SUM(Age) FROM Persons
```

Result:

```
98
```

Example

This example returns the sum of ages for persons that are more than 20 years old:

```
SELECT SUM(Age) FROM Persons WHERE Age>20
```

Result:

```
79
```

SQL Group By and Having

Aggregate functions (like SUM) often need an added GROUP BY functionality.

GROUP BY...

GROUP BY... was added to SQL because aggregate functions (like SUM) return the aggregate of all column values every time they are called, and without the GROUP BY function it was impossible to find the sum for each individual group of column values.

The syntax for the GROUP BY function is:

```
SELECT column,SUM(column) FROM table GROUP BY column
```

GROUP BY Example

This "Sales" Table:

Company	Amount
W3Schools	5500
IBM	4500
W3Schools	7100

And This SQL:

```
SELECT Company, SUM(Amount) FROM Sales
```

Returns this result:

Company	SUM(Amount)
W3Schools	17100
IBM	17100
W3Schools	17100

The above code is invalid because the column returned is not part of an aggregate. A GROUP BY clause will solve this problem:

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company
```

Returns this result:

Company	SUM(Amount)
W3Schools	12600
IBM	4500

HAVING...

HAVING... was added to SQL because the WHERE keyword could not be used against aggregate functions (like SUM), and without HAVING... it would be impossible to test for result conditions.

The syntax for the HAVING function is:

```
SELECT column,SUM(column) FROM table
GROUP BY column
HAVING SUM(column) condition value
```

This "Sales" Table:

Company	Amount
W3Schools	5500
IBM	4500
W3Schools	7100

This SQL:

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company
HAVING SUM(Amount)>10000
```

Returns this result

Company	SUM(Amount)
W3Schools	12600

SQL Join

Joins and Keys

Sometimes we have to select data from two tables to make our result complete. We have to perform a join.

Tables in a database can be related to each other with keys. A primary key is a column with a unique value for each row. The purpose is to bind data together, across tables, without repeating all of the data in every table.

In the "Employees" table below, the "Employee_ID" column is the primary key, meaning that **no** two rows can have the same Employee_ID. The Employee_ID distinguishes two persons even if they have the same name.

When you look at the example tables below, notice that:

- The "Employee_ID" column is the primary key of the "Employees" table
- The "Prod_ID" column is the primary key of the "Orders" table
- The "Employee_ID" column in the "Orders" table is used to refer to the persons in the "Employees" table without using their names

---

Employees:

Employee_ID	Name
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

Orders:

Prod_ID	Product	Employee_ID
234	Printer	01
657	Table	03
865	Chair	03

Referring to Two Tables

We can select data from two tables by referring to two tables, like this:

Example

Who has ordered a product, and what did they order?

```
SELECT Employees.Name, Orders.Product
FROM Employees, Orders
WHERE Employees.Employee_ID=Orders.Employee_ID
```

Result

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

Example

Who ordered a printer?

```
SELECT Employees.Name
FROM Employees, Orders
WHERE Employees.Employee_ID=Orders.Employee_ID
AND Orders.Product='Printer'
```

Result

Name
Hansen, Ola

SQL Create Table

Create a Table

To create a table in a database:

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
.......
)
```

Example

This example demonstrates how you can create a table named "Person", with four columns. The column names will be "LastName", "FirstName", "Address", and "Age":

```
CREATE TABLE Person
(
LastName varchar,
FirstName varchar,
Address varchar,
Age int
)
```

This example demonstrates how you can specify a maximum length for some columns:

```
CREATE TABLE Person
(
LastName varchar(30),
FirstName varchar,
Address varchar,
Age int(3)
);
```

The data type specifies what type of data the column can hold. The table below contains the most common data types in SQL:

Data Type	Description
integer(size) int(size) smallint(size) tinyint(size)	Hold integers only. The maximum number of digits are specified in parenthesis.
decimal(size,d) numeric(size,d)	Hold numbers with fractions. The maximum number of digits are specified in "size". The maximum number of digits to the right of the decimal is specified in "d".
char(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis.
varchar(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis.
date(yyyymmdd)	Holds a date

Create Index

Indices are created in an existing table to locate rows more quickly and efficiently. It is possible to create an index on one or more columns of a table, and each index is given a name. The users can not see the indices, they are just used to speed up queries.

Note: Updating a table containing indices takes more time than updating a table without, this is because the indices also need an update. So, it is a good idea to create indices only on columns that are often used for a search.

A Unique Index

Creates a unique index on a table. A unique index means that two rows cannot have the same index value.

```
CREATE UNIQUE INDEX index_name
ON table_name (column_name)
```

The "column_name" specifies the column you want indexed.

A Simple Index

Creates a simple index on a table. When the UNIQUE keyword is omitted, duplicate values are allowed.

```
CREATE INDEX index_name
ON table_name (column_name)
```

The "column_name" specifies the column you want indexed.

Example

This example creates a simple index, named "PersonIndex", on the LastName field of the Person table:

```
CREATE INDEX PersonIndex
ON Person (LastName)
```

If you want to index the values in a column in **descending** order, you can add the reserved word **DESC** after the column name:

```
CREATE INDEX PersonIndex
ON Person (LastName DESC)
```

If you want to index more than one column you can list the column names within the parentheses, separated by commas:.

```
CREATE INDEX PersonIndex
ON Person (LastName, FirstName)
```

Drop Index

You can delete an existing index in a table with the DROP statement.

```
DROP INDEX table_name.index_name
```

Delete a Table

To delete a table (the table structure, attributes, and indexes will also be deleted):

```
DROP TABLE table_name
```

SQL Alter Table

Alter Table

The ALTER TABLE statement is used to add or drop columns in an existing table.

```
ALTER TABLE table_name
ADD column_name datatype
ALTER TABLE table_name
DROP COLUMN column_name
```

Note: Some database systems don't allow the dropping of a column in a database table (DROP COLUMN column_name).

Person:

LastName	FirstName	Address
Pettersen	Kari	Storgt 20

Example

To add a column named "City" in the "Person" table:

```
ALTER TABLE Person ADD City varchar(30)
```

Result:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	

Example

To drop the "Address" column in the "Person" table:

```
ALTER TABLE Person DROP COLUMN Address
```

Result:

LastName	FirstName	City
Pettersen	Kari	
