

9. SQL FUNDAMENTALS

What is SQL?

- SQL stands for **Structured Query Language**
- SQL allows you to access a database
- SQL is an ANSI standard computer 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 an ANSI (American National Standards Institute) standard computer language for accessing and manipulating 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. There are many different versions of the SQL language, but to be in compliance with the ANSI standard, they must support the same major keywords in a similar manner (such as SELECT, UPDATE, DELETE, INSERT, WHERE, and others).

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

SQL 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. 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 Language types:

Structured Query Language(SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database. SQL uses certain commands like Create, Drop, Insert, etc. to carry out the required tasks.

These SQL commands are mainly categorized into four categories as:

1. DDL – Data Definition Language
2. DQL – Data Query Language
3. DML – Data Manipulation Language
4. DCL – Data Control Language

Though many resources claim there to be another category of SQL clauses **TCL – Transaction Control Language**. So we will see in detail about TCL as well.

1. **DDL(Data Definition Language)** : DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

Examples of DDL commands:

- **CREATE** – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).

- **DROP** – is used to delete objects from the database.
- **ALTER**-is used to alter the structure of the database.
- **TRUNCATE**–is used to remove all records from a table, including all spaces allocated for the records are removed.
- **COMMENT** –is used to add comments to the data dictionary.
- **RENAME** –is used to rename an object existing in the database.

DQL (Data Query Language) :

1. **DQL** statements are used for performing queries on the data within schema objects. The purpose of the DQL Command is to get some schema relation based on the query passed to it.

Example of DQL:

- **SELECT** – is used to retrieve data from the database.

2. **DML(Data Manipulation Language):** The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

Examples of DML:

- **INSERT** – is used to insert data into a table.
- **UPDATE** – is used to update existing data within a table.
- **DELETE** – is used to delete records from a database table.

3. **DCL(Data Control Language):** DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions and other controls of the database system.

Examples of DCL commands:

- **GRANT**-gives user's access privileges to the database.
- **REVOKE**-withdraw user's access privileges given by using the GRANT command.

4. **TCL(transaction Control Language):** TCL commands deal with the transaction within the database.

Examples of TCL commands:

- **COMMIT**– commits a Transaction.
- **ROLLBACK**– rolls back a transaction in case of any error occurs.
- **SAVEPOINT**–sets a savepoint within a transaction.
- **SET TRANSACTION**–specify characteristics for the transaction.

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


SQL-DML Query examples

The SQL 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
```

 **Note:** SQL statements are not case sensitive. SELECT is the same as select.

SQL SELECT Example

To select the content of columns named "LastName" and "FirstName", from the database table called "Persons", use a SELECT statement like this:

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

The database table "Persons":

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

The 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 SELECT DISTINCT Statement

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

The 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:

Syntax

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

To select ALL values from the column named "Company" we use a SELECT statement like this:

```
SELECT Company FROM Orders
```

"Orders" table

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

Result

Company
Sega
W3Schools
Trio
W3Schools

```
SELECT DISTINCT Company FROM Orders
```

Result:

Company
Sega
W3Schools
Trio

Now "W3Schools" is listed only once in the result-set.

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'
```

LIKE Condition**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%'
```

The following SQL statement will return persons with first names that end with an 'a':

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

The following SQL statement will return persons with first names that contain the pattern 'la':

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

INSERT INTO

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	

UPDATE

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

DELETE

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
Drop

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
```

ORDER BY

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 company names in alphabetical order:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company ASC (ascending)
```

Result:

Company	OrderNumber
ABC Shop	5678
Sega	3412
W3Schools	6798
W3Schools	2312

Example

To display the company names in alphabetical order AND the OrderNumber in numerical order:

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

Result:

Company	OrderNumber
ABC Shop	5678
Sega	3412
W3Schools	2312
W3Schools	6798

Aggregate functions in MS Access and SQL server

- Aggregate functions operate against a collection of values, but return a single value.

Function	Description
AVG(column)	Returns the average value of a column
COUNT(column)	Returns the number of rows (without a NULL value) of a column

COUNT(*)	Returns the number of selected rows
FIRST(column)	Returns the value of the first record in a specified field
LAST(column)	Returns the value of the last record in a specified field
MAX(column)	Returns the highest value of a column
MIN(column)	Returns the lowest value of a column
STDEV(column)	This function is used to compute statistical standard deviation from sample data.
STDEVP(column)	This function is used to compute statistical standard deviation for an entire population data.

Scalar functions

Scalar functions operate against a single value, and return a single value based on the input value.

Useful Scalar Functions in MS Access

Function	Description
UCASE(c)	Converts a field to upper case
LCASE(c)	Converts a field to lower case
MID(c,start[,end])	Extract characters from a text field
LEN(c)	Returns the length of a text field
INSTR(c,char)	Returns the numeric position of a named character within a text field
LEFT(c,number_of_char)	Return the left part of a text field requested
RIGHT(c,number_of_char)	Return the right part of a text field requested
ROUND(c,decimals)	Rounds a numeric field to the number of decimals specified
MOD(x,y)	Returns the remainder of a division operation

GROUP BY

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...

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 example :

Consider the employee database , where the primary keys are Underlined.

Employee(empname,street,city)

Works(empname,companyname,salary)

Company(companyname,city)

Manages(empname,management)

Write the SQL query for the following requests

1) Find the names of all employees who work for Company A

Ans:

Select empname from Works Where companyname='Company A'

2) Find the names, street addresses and cities of residence of all employees who work for Company A and earn more than 200000 per annum.

Ans:

select * from Employee where empname in

(select empname from Works where companyname='company A' and salary>200000)

3)Find the names of all employees in this database who live in the same city as the company for which they work.

Ans:

select E.empname from Employee as E, Works as W, Company as C where

E.empname=W.empname and E.city=C.city and

W.company_name=C.company_name

4)Find the names of all employees who earn more than every employees of Company B

B

Ans:

select empname from Works where salary > all

(select salary from Works where companyname='Company B')

5. Delete all tuples in the works relation for employees of Company B

Ans

Delete from Works where companyname='Company B'

