

DBMS – Database Management System

SQL – Structured Query Language

Content

- 1.** SQL HOME
- 2.** SQL Intro
- 3.** SQL Syntax
- 4.** SQL Select
- 5.** SQL Distinct
- 6.** SQL Where
- 7.** SQL And & Or
- 8.** SQL Order By
- 9.** SQL Insert Into
- 10.** SQL Update
- 11.** SQL Delete
- 12.** SQL Injection
- 13.** SQL Select Top
- 14.** SQL Like
- 15.** SQL Wildcards
- 16.** SQL In
- 17.** SQL Between
- 18.** SQL Aliases
- 19.** SQL Joins
- 20.** SQL Inner Join
- 21.** SQL Left Join
- 22.** SQL Right Join
- 23.** SQL Full Join
- 24.** SQL Union
- 25.** SQL Select Into
- 26.** SQL Into Select
- 27.** SQL Create DB
- 28.** SQL Create Table
- 29.** SQL Constraints
- 30.** SQL Not Null
- 31.** SQL Unique
- 32.** SQL Primary Key
- 33.** SQL Foreign Key
- 34.** SQL Check
- 35.** SQL Default
- 36.** SQL Create Index
- 37.** SQL Drop
- 38.** SQL Alter
- 39.** SQL Auto Increment

DBMS – Database Management System

SQL – Structured Query Language

- 40.** SQL Views
- 41.** SQL Dates
- 42.** SQL Null Values
- 43.** SQL Null Functions
- 44.** SQL Data Types
- 45.** SQL DB Data Types

- 46.** SQL Functions
- 47.** SQL Avg()
- 48.** SQL Count()
- 49.** SQL First()
- 50.** SQL Last()
- 51.** SQL Max()
- 52.** SQL Min()
- 53.** SQL Sum()
- 54.** SQL Group By
- 55.** SQL Having
- 56.** SQL Ucase()
- 57.** SQL Lcase()
- 58.** SQL Mid()
- 59.** SQL Len()
- 60.** SQL Round()
- 61.** SQL Now()
- 62.** SQL Format()
- 63.** Summary

DBMS – Database Management System

SQL – Structured Query Language

Lesson 01. SQL Tutorial

SQL is a standard language for accessing databases.

Our SQL tutorial will teach you how to use SQL to access and manipulate data in: MySQL & MS-SQL Server.

Lesson 02. Introduction to SQL

SQL is a standard language for accessing and manipulating databases.

What is SQL?

- SQL stands for Structured Query Language
 - SQL lets you access and manipulate databases
 - SQL is an ANSI (American National Standards Institute) standard
-

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
-

DBMS – Database Management System

SQL – Structured Query Language

SQL is a Standard - BUT....

Although SQL is an ANSI (American National Standards Institute) 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

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.

DBMS – Database Management System

SQL – Structured Query Language

Lesson 03. SQL Syntax

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 SDS_V14 sample database (included in MS Access and MS SQL Server, MySQL).

Below is a selection from the "tblCustomers" table:

custID	custName	ContactCompany	Address	City	PostalCode	Country
1	Abdirahman Samadon	Sahalssoftware	2147483647	Nairobi	254	Kenya
2	Ahmed Kayd	Sahalssoftware	2147483647	Buhodle	254	Somalia
3	Abdiwasic Ducaysane	Sahalssoftware	2147483647	Khartum	254	Sudan
4	Fahad Abdirahman	Sahalssoftware	2147483647	Borama	254	Somalia
5	Abdirahman Yusuf	Sahalssoftware	2147483647	Laascaanood	254	Somalia
6	Abdifatah Abdilahi	Sahalssoftware	2147483647	Laascaanood	252	Somalia
7	Mohamud A Artan	Sahalssoftware	2147483647	Laascaanood	254	Somalia
8	Ahmed Abdiweli	Sahalssoftware	2147483647	Laascaanood	252	Somalia

The table above contains eight records (one for each customer) and seven columns (CustID, CustName, ContactCompany, 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 "tblCustomers" table:

DBMS – Database Management System

SQL – Structured Query Language

Example

```
SELECT * FROM tblCustomers;
```

Result:

custID	custName	ContactCompany	Address	City	PostalCode	Country
1	Abdirahman Samadon	Sahalsoftware	2147483647	Nairobi	254	Kenya
2	Ahmed Kayd	Sahalsoftware	2147483647	Buhodle	254	Somalia
3	Abdiwasic Ducaysane	Sahalsoftware	2147483647	Khartum	254	Sudan
4	Fahad Abdirahman	Sahalsoftware	2147483647	Borama	254	Somalia
5	Abdirahman Yusuf	Sahalsoftware	2147483647	Laascaanood	254	Somalia
6	Abdifatah Abdilahi	Sahalsoftware	2147483647	Laascaanood	252	Somalia
7	Mohamud A Artan	Sahalsoftware	2147483647	Laascaanood	254	Somalia
8	Ahmed Abdiweli	Sahalsoftware	2147483647	Laascaanood	252	Somalia

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

Keep in Mind That...

- SQL is 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

DBMS – Database Management System

SQL – Structured Query Language

- **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

Lesson 04. SQL **CREATE DATABASE** Statement

Note:

- Install XAMP
- Run XAMP
- Start

The SQL **CREATE DATABASE** Statement

The CREATE DATABASE statement is used to create a database.

SQL **CREATE DATABASE** Syntax

```
CREATE DATABASE dbname;
```

SQL **CREATE DATABASE** Example

The following SQL statement creates a database called "my_db":

```
CREATE DATABASE my_db;
```

Database tables can be added with the CREATE TABLE statement.

Lesson 05. SQL **CREATE TABLE** Statement

The SQL **CREATE TABLE** Statement

The CREATE TABLE statement is used to create a table in a database.

DBMS – Database Management System

SQL – Structured Query Language

Tables are organized into rows and columns; and each table must have a name.

SQL CREATE TABLE Syntax

```
CREATE TABLE table_name
(
  column_name1 data_type(size),
  column_name2 data_type(size),
  column_name3 data_type(size),
  ....
);
```

The *column_name* parameters specify the names of the columns of the table.

The *data_type* parameter specifies what type of data the column can hold (e.g. varchar, integer, decimal, date, etc.).

The *size* parameter specifies the maximum length of the column of the table.

SQL CREATE TABLE Example

Now we want to create a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City.

We use the following CREATE TABLE statement:

Example

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

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:

DBMS – Database Management System

SQL – Structured Query Language

PersonID	LastName	FirstName	Address

Tip: The empty table can be filled with data with the INSERT INTO statement.

Lesson 06. SQL **INSERT INTO** Statement

The INSERT INTO statement is used to insert new records in a table.

The SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

SQL INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two forms.

The first form does not specify the column names where the data will be inserted, only their values:

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

The second form specifies both the column names and the values to be inserted:

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

Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

Below is a selection from the "tblCustomers" table:

DBMS – Database Management System

SQL – Structured Query Language

INSERT INTO Example

Assume we wish to insert a new row in the "tblCustomers" table.

We can use the following SQL statement:

Example

```
INSERT INTO tblCustomers (CustName, ContactName, Address, City, PostalCode, Country)
VALUES ('Samadoon Saleebaan Xuseen', 'Sahalsoftware Burco', '635009', 'Stavanger', '45', 'Denmark');
```

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

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalsoftware	635000	Laascaanood	252	Somalia
2	Faarax Jaamac Xuseen	Sahalsoftware	635001	Garoowe	252	Somalia
4	Aamina Xasan Jaamac	Sahalsoftware	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalsoftware	635007	Nayroobi	254	Kenya
6	Faadumo Siciid Jaamac	Sahalsoftware	635004	New York	1	United State
7	Samadoon Saleebaan Xuseen	Sahalsoftware Burco	635009	Stavanger	45	Denmark



Did you notice that we did not insert any number into the CustID field?
The CustomerID column is automatically updated with a unique number for each record in 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 row, but only insert data in the "CustName", "City", and "Country" columns (and the CustID field will of course also be updated automatically):

DBMS – Database Management System

SQL – Structured Query Language

Example

```
INSERT INTO Customers (CustName, City, Country)
VALUES ('Abdifatah', 'Baladweyne', 'Somalia');
```

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

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalsoftware	635000	Laascaanood	252	Somalia
2	Faarax Jaamac Xuseen	Sahalsoftware	635001	Garoowe	252	Somalia
4	Aamina Xasan Jaamac	Sahalsoftware	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalsoftware	635007	Nayroobi	254	Kenya
6	Faadumo Siciid Jaamac	Sahalsoftware	635004	New York	1	United State
7	Samadoon Saleebaan Xuseen	Sahalsoftware Burco	635009	Stavanger	45	Denmark
8	Abdifatah	NULL	NULL	Baladweyne	NULL	Somalia

Lesson 07. SQL **SELECT** Statement

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

The SQL SELECT Statement

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

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

SQL SELECT Syntax

```
SELECT column_name, column_name
FROM table_name;
```

and

```
SELECT * FROM table_name;
```

DBMS – Database Management System

SQL – Structured Query Language

Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

Below is a selection from the "tblCustomers" table:

custID	custName	ContactCompany	Address	City	PostalCode	Country
1	Abdirahman Samadon	Sahalssoftware	2147483647	Nairobi	254	Kenya
2	Ahmed Kayd	Sahalssoftware	2147483647	Buhodle	254	Somalia
3	Abdiwasic Ducaysane	Sahalssoftware	2147483647	Khartum	254	Sudan
4	Fahad Abdirahman	Sahalssoftware	2147483647	Borama	254	Somalia
5	Abdirahman Yusuf	Sahalssoftware	2147483647	Laascaanood	254	Somalia
6	Abdifatah Abdilahi	Sahalssoftware	2147483647	Laascaanood	252	Somalia
7	Mohamud A Artan	Sahalssoftware	2147483647	Laascaanood	254	Somalia
8	Ahmed Abdiweli	Sahalssoftware	2147483647	Laascaanood	252	Somalia

SELECT Column Example

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

Example

```
SELECT CustName,City FROM tblCustomers;
```

SELECT * Example

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

DBMS – Database Management System

SQL – Structured Query Language

Example

```
SELECT * FROM tblCustomers;
```

Lesson 08. SQL **SELECT DISTINCT** Statement

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

The SQL **SELECT DISTINCT** Statement

In a table, a column may contain many duplicate values; and sometimes you only want to list the different (distinct) values.

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

SQL **SELECT DISTINCT** Syntax

```
SELECT DISTINCT column_name, column_name  
FROM table_name;
```

Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

Below is a selection from the "tblCustomers" table:

custID	custName	ContactCompany	Address	City	PostalCode	Country
1	Abdirahman Samadon	Sahalssoftware	2147483647	Nairobi	254	Kenya
2	Ahmed Kayd	Sahalssoftware	2147483647	Buhodle	254	Somalia
3	Abdiwasic Ducaysane	Sahalssoftware	2147483647	Khartum	254	Sudan
4	Fahad Abdirahman	Sahalssoftware	2147483647	Borama	254	Somalia
5	Abdirahman Yusuf	Sahalssoftware	2147483647	Laascaanood	254	Somalia
6	Abdifatah Abdilahi	Sahalssoftware	2147483647	Laascaanood	252	Somalia
7	Mohamud A Artan	Sahalssoftware	2147483647	Laascaanood	254	Somalia
8	Ahmed Abdiweli	Sahalssoftware	2147483647	Laascaanood	252	Somalia

DBMS – Database Management System

SQL – Structured Query Language

SELECT DISTINCT Example

The following SQL statement selects only the distinct values from the "City" columns from the "Customers" table:

Example

```
SELECT DISTINCT City FROM tblCustomers;
```

Lesson 09. SQL **WHERE** Clause

The WHERE clause is used to filter records.

The SQL WHERE Clause

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

SQL WHERE Syntax

```
SELECT column_name, column_name  
FROM table_name  
WHERE column_name operator value;
```

Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

Below is a selection from the "tblCustomers" table:

DBMS – Database Management System

SQL – Structured Query Language

custID	custName	ContactCompany	Address	City	PostalCode	Country
1	Abdirahman Samadon	Sahalsoftware	2147483647	Nairobi	254	Kenya
2	Ahmed Kayd	Sahalsoftware	2147483647	Buhodle	254	Somalia
3	Abdiwasic Ducaysane	Sahalsoftware	2147483647	Khartum	254	Sudan
4	Fahad Abdirahman	Sahalsoftware	2147483647	Borama	254	Somalia
5	Abdirahman Yusuf	Sahalsoftware	2147483647	Laascaanood	254	Somalia
6	Abdifatah Abdilahi	Sahalsoftware	2147483647	Laascaanood	252	Somalia
7	Mohamud A Artan	Sahalsoftware	2147483647	Laascaanood	254	Somalia
8	Ahmed Abdiweli	Sahalsoftware	2147483647	Laascaanood	252	Somalia

WHERE Clause Example

The following SQL statement selects all the customers from the country "Somalia", in the "tblCustomers" table:

Example

```
SELECT * FROM tblCustomers  
WHERE Country='Somalia';
```

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 tblCustomers  
WHERE CustID=1;
```

Operators in The WHERE Clause

The following operators can be used in the WHERE clause:

Operator Description

DBMS – Database Management System

SQL – Structured Query Language

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

Lesson 10. SQL AND & OR Operators

The AND & OR operators are used to filter records based on more than one condition.

The SQL AND & OR Operators

The AND operator displays a record if both the first condition AND the second condition are true.

The OR operator displays a record if either the first condition OR the second condition is true.

Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

DBMS – Database Management System

SQL – Structured Query Language

Below is a selection from the "tblCustomers" table:

custID	custName	ContactCompany	Address	City	PostalCode	Country
1	Abdirahman Samadon	Sahalssoftware	2147483647	Nairobi	254	Kenya
2	Ahmed Kayd	Sahalssoftware	2147483647	Buhodle	254	Somalia
3	Abdiwasic Ducaysane	Sahalssoftware	2147483647	Khartum	254	Sudan
4	Fahad Abdirahman	Sahalssoftware	2147483647	Borama	254	Somalia
5	Abdirahman Yusuf	Sahalssoftware	2147483647	Laascaanood	254	Somalia
6	Abdifatah Abdilahi	Sahalssoftware	2147483647	Laascaanood	252	Somalia
7	Mohamud A Artan	Sahalssoftware	2147483647	Laascaanood	254	Somalia
8	Ahmed Abdiweli	Sahalssoftware	2147483647	Laascaanood	252	Somalia

AND Operator Example

The following SQL statement selects all customers from the country "Somalia" AND the city "Laascaanood", in the "tblCustomers" table:

Example

```
SELECT * FROM tblCustomers
WHERE Country='Somalia'
AND City='Laascaanood';
```

OR Operator Example

The following SQL statement selects all customers from the city "Laascaanood" OR "Garoowe", in the "tblCustomers" table:

Example

```
SELECT * FROM tblCustomers
WHERE City='Laascaanood'
OR City='Borama';
```

DBMS – Database Management System

SQL – Structured Query Language

Combining AND & OR

You can also combine AND and OR (use parenthesis to form complex expressions).

The following SQL statement selects all customers from the country "Somalia" AND the city must be equal to "Laascaanood" OR "Garowe", in the "Customers" table:

Example

```
SELECT * FROM tblCustomers  
WHERE Country='Somalia'  
AND (City='Laascaanood' OR City='Borama');
```

Lesson 11. SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set.

The SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set by one or more columns.

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

DBMS – Database Management System

SQL – Structured Query Language

SQL ORDER BY Syntax

```
SELECT column_name, column_name  
FROM table_name  
ORDER BY column_name, column_name ASC|DESC;
```

Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

Below is a selection from the "tblCustomers" table:

custID	custName	ContactCompany	Address	City	PostalCode	Country
1	Abdirahman Samadon	Sahalsoftware	2147483647	Nairobi	254	Kenya
2	Ahmed Kayd	Sahalsoftware	2147483647	Buhodle	254	Somalia
3	Abdiwasic Ducaysane	Sahalsoftware	2147483647	Khartum	254	Sudan
4	Fahad Abdirahman	Sahalsoftware	2147483647	Borama	254	Somalia
5	Abdirahman Yusuf	Sahalsoftware	2147483647	Laascaanood	254	Somalia
6	Abdifatah Abdilahi	Sahalsoftware	2147483647	Laascaanood	252	Somalia
7	Mohamud A Artan	Sahalsoftware	2147483647	Laascaanood	254	Somalia
8	Ahmed Abdiweli	Sahalsoftware	2147483647	Laascaanood	252	Somalia

ORDER BY Example

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

Example

```
SELECT * FROM tblCustomers  
ORDER BY CustName;
```

DBMS – Database Management System

SQL – Structured Query Language

ORDER BY DESC Example

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

Example

```
SELECT * FROM tblCustomers  
ORDER BY CustName DESC;
```

ORDER BY Several Columns Example

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

Example

```
SELECT * FROM tblCustomers  
ORDER BY CustName, City;
```

Lesson 12. SQL UPDATE Statement

The UPDATE statement is used to update records in a table.

The SQL UPDATE Statement

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

SQL UPDATE Syntax

```
UPDATE table_name  
SET column1=value1, column2=value2, ...  
WHERE some_column=some_value;
```

DBMS – Database Management System

SQL – Structured Query Language



Notice the WHERE clause in the SQL UPDATE statement!

The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

Below is a selection from the "tblCustomers" table:

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalsoftware	635000	Laascaanood	252	Somalia
2	Faarax Jaamac Xuseen	Sahalsoftware	635001	Garoowe	252	Somalia
4	Aamina Xasan Jaamac	Sahalsoftware	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalsoftware	635007	Nayroobi	254	Kenya
6	Faadumo Siciid Jaamac	Sahalsoftware	635004	New York	1	United State
7	Samadoon Saleebaan Xuseen	Sahalsoftware Burco	635009	Stavanger	45	Denmark
8	Abdifatah	NULL	NULL	Baladweyne	NULL	Somalia

SQL UPDATE Example

Assume we wish to update the customer "Abdifatah" with a new contact person and Address.

We use the following SQL statement:

Example

```
UPDATE tblCustomers  
SET ContactName='Sahalsoftware', Address='635008'  
WHERE CustName='Abdifatah';
```

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

DBMS – Database Management System

SQL – Structured Query Language

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalsoftware	635000	Laascaanood	252	Somalia
2	Faarax Jaamac Xuseen	Sahalsoftware	635001	Garoowe	252	Somalia
4	Aamina Xasan Jaamac	Sahalsoftware	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalsoftware	635007	Nayroobi	254	Kenya
6	Faadumo Siciid Jaamac	Sahalsoftware	635004	New York	1	United State
7	Samadoon Saleebaan Xuseen	Sahalsoftware Burco	635009	Stavanger	45	Denmark
8	Abdifatah	Sahalsoftware	635008	Baladweyne	NULL	Somalia

Update Warning!

Be careful when updating records. If we had omitted the WHERE clause, in the example above, like this:

```
UPDATE tblCustomers  
SET ContactName='Sahalsoftware College';
```

The "tblCustomers" table would have looked like this:

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalsoftware College	635000	Laascaanood	252	Somalia
2	Faarax Jaamac Xuseen	Sahalsoftware College	635001	Garoowe	252	Somalia
4	Aamina Xasan Jaamac	Sahalsoftware College	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalsoftware College	635007	Nayroobi	254	Kenya
6	Faadumo Siciid Jaamac	Sahalsoftware College	635004	New York	1	United State
7	Samadoon Saleebaan Xuseen	Sahalsoftware College	635009	Stavanger	45	Denmark
8	Abdifatah	Sahalsoftware College	635008	Baladweyne	NULL	Somalia

Lesson 13. SQL DELETE Statement

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

The SQL DELETE Statement

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

DBMS – Database Management System

SQL – Structured Query Language

SQL DELETE Syntax

```
DELETE FROM table_name  
WHERE some_column=some_value;
```



Notice the WHERE clause in the SQL DELETE statement!

The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

Below is a selection from the "tblCustomers" table:

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalssoftware College	635000	Laascaanood	252	Somalia
2	Faarax Jaamac Xuseen	Sahalssoftware College	635001	Garowe	252	Somalia
4	Aamina Xasan Jaamac	Sahalssoftware College	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalssoftware College	635007	Nayroobi	254	Kenya
6	Faadumo Siciid Jaamac	Sahalssoftware College	635004	New York	1	United State
7	Samadoon Saleebaan Xuseen	Sahalssoftware College	635009	Stavanger	45	Denmark
8	Abdifatah	Sahalssoftware College	635008	Baladweyne	NULL	Somalia

SQL DELETE Example

Assume we wish to delete the customer "Faadumo Siciid Jaamac" from the "tblCustomers" table.

We use the following SQL statement:

Example

```
DELETE FROM tblCustomers  
WHERE CustID='6';
```

DBMS – Database Management System

SQL – Structured Query Language

The "tblCustomers" table will now look like this:

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalssoftware College	635000	Laascaanood	252	Somalia
2	Faarax Jaamac Xuseen	Sahalssoftware College	635001	Garoowe	252	Somalia
4	Aamina Xasan Jaamac	Sahalssoftware College	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalssoftware College	635007	Nayroobi	254	Kenya
7	Samadoon Saleebaan Xuseen	Sahalssoftware College	635009	Stavanger	45	Denmark
8	Abdifatah	Sahalssoftware College	635008	Baladweyne	NULL	Somalia

Delete All Data

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;
```

Note: Be very careful when deleting records. You cannot undo this statement!

Lesson 14. SQL **SELECT TOP** Clause

The **SQL SELECT TOP** Clause

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

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

DBMS – Database Management System

SQL – Structured Query Language

SQL SELECT TOP in MySQL

MySQL Syntax

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

Example

```
SELECT *  
FROM Persons  
LIMIT 5;
```

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Customers" table:

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalsoftware College	635000	Laascaanood	252	Somalia
2	Faarax Jaamac Xuseen	Sahalsoftware College	635001	Garoowe	252	Somalia
4	Aamina Xasan Jaamac	Sahalsoftware College	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalsoftware College	635007	Nayroobi	254	Kenya
7	Samadoon Saleebaan Xuseen	Sahalsoftware College	635009	Stavanger	45	Denmark
8	Abdifatah	Sahalsoftware College	635008	Baladweyne	NULL	Somalia

Lesson 15. SQL LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

The SQL LIKE Operator

The LIKE operator is used to search for a specified pattern in a column.

DBMS – Database Management System

SQL – Structured Query Language

SQL LIKE Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern;
```

Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

Below is a selection from the "tblCustomers" table:

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalsoftware College	635000	Laascaanood	252	Somalia
2	Faarax Jaamac Xuseen	Sahalsoftware College	635001	Garoowe	252	Somalia
4	Aamina Xasan Jaamac	Sahalsoftware College	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalsoftware College	635007	Nayroobi	254	Kenya
7	Samadoon Saleebaan Xuseen	Sahalsoftware College	635009	Stavanger	45	Denmark
8	Abdifatah	Sahalsoftware College	635008	Baladweyne	NULL	Somalia

SQL LIKE Operator Examples

The following SQL statement selects all customers with a City starting with the letter "s":

Example

```
SELECT * FROM tblCustomers
WHERE City LIKE 's%';
```

Tip: The "%" sign is used to define wildcards (missing letters) both before and after the pattern. You will learn more about wildcards in the next chapter.

The following SQL statement selects all customers with a City ending with the letter "s":

Example

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

DBMS – Database Management System

SQL – Structured Query Language

The following SQL statement selects all customers with a Country containing the pattern "mal":

Example

```
SELECT * FROM tblCustomers  
WHERE Country LIKE '%mal%';
```

Using the NOT keyword allows you to select records that does NOT match the pattern.

The following SQL statement selects all customers with a Country NOT containing the pattern "mal":

Example

```
SELECT * FROM tblCustomers  
WHERE Country NOT LIKE '%mal%';
```

Lesson 16. SQL Wildcards

A wildcard character can be used to substitute for any other character(s) in a string.

SQL Wildcard Characters

In SQL, wildcard characters are used with the SQL LIKE operator.

SQL wildcards are used to search for data within a table.

With SQL, the wildcards are:

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalssoftware College	635000	Laascaanood	252	Somalia
2	Faarax Jaamac Xuseen	Sahalssoftware College	635001	Garoowe	252	Somalia
4	Aamina Xasan Jaamac	Sahalssoftware College	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalssoftware College	635007	Nayroobi	254	Kenya
7	Samadoon Saleebaan Xuseen	Sahalssoftware College	635009	Stavanger	45	Denmark
8	Abdifatah	Sahalssoftware College	635008	Baladweyne	NULL	Somalia

DBMS – Database Management System

SQL – Structured Query Language

Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

Below is a selection from the "tblCustomers" table:

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalssoftware College	635000	Laascaanood	252	Somalia
2	Faarax Jaamac Xuseen	Sahalssoftware College	635001	Garoowe	252	Somalia
4	Aamina Xasan Jaamac	Sahalssoftware College	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalssoftware College	635007	Nayroobi	254	Kenya
7	Samadoon Saleebaan Xuseen	Sahalssoftware College	635009	Stavanger	45	Denmark
8	Abdifatah	Sahalssoftware College	635008	Baladweyne	NULL	Somalia

Using the SQL % Wildcard

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

Example

```
SELECT * FROM tblCustomers
WHERE City LIKE 'las%';
```

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

Example

```
SELECT * FROM tblCustomers
WHERE City LIKE '%caa%';
```

Using the SQL _ Wildcard

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

DBMS – Database Management System

SQL – Structured Query Language

Example

```
SELECT * FROM tblCustomers  
WHERE City LIKE '_aascaanood';
```

- Student Assignment -

Using the SQL [charlist] Wildcard

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

Example

```
SELECT * FROM tblCustomers  
WHERE City LIKE '[lgb]%';
```

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

Example

```
SELECT * FROM tblCustomers  
WHERE City LIKE '[a-c]%';
```

The following SQL statement selects all customers with a City NOT starting with "b", "s", or "p":

Example

```
SELECT * FROM tblCustomers  
WHERE City LIKE '[!bsp]%';
```

or

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

Lesson 17. SQL IN Operator

DBMS – Database Management System

SQL – Structured Query Language

The IN Operator

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

SQL IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...);
```

Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

Below is a selection from the "tblCustomers" table:

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalsoftware College	635000	arabsiiyo	252	Somalia
2	Faarax Jaamac Xuseen	Sahalsoftware College	635001	bosaso	252	Somalia
4	Aamina Xasan Jaamac	Sahalsoftware College	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalsoftware College	635007	Nayroobi	254	Kenya
7	Samadoon Saleebaan Xuseen	Sahalsoftware College	635009	Stavanger	45	Denmark
8	Abdifatah	Sahalsoftware College	635008	Baladweyne	NULL	Somalia

IN Operator Example

The following SQL statement selects all customers with a address of "635001" or "635008":

Example

```
SELECT * FROM tblCustomers
WHERE Address IN ('635001','635008');
```

DBMS – Database Management System

SQL – Structured Query Language

Lesson 18. SQL BETWEEN Operator

The BETWEEN operator is used to select values within a range.

The SQL BETWEEN Operator

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

SQL BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

ProdID	ProdName	VendorID	CategoryID	Unit	Price
1	Bariis	1	1	10	20
2	Sonkor	2	1	20	25
3	Bur	1	2	25	20
4	Caleen	1	4	100	10

Below is a selection from the "tblProducts" table:

DBMS – Database Management System

SQL – Structured Query Language

BETWEEN Operator Example

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

Example

```
SELECT * FROM tblProducts  
WHERE Price BETWEEN 10 AND 20;
```

NOT BETWEEN Operator Example

To display the products outside the range of the previous example, use NOT BETWEEN:

Example

```
SELECT * FROM Products  
WHERE Price NOT BETWEEN 10 AND 20;
```

BETWEEN Operator with IN Example

The following SQL statement selects all products with a price BETWEEN 10 and 20, but products with a CategoryID of 1,2, or 3 should not be displayed:

Example

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

BETWEEN Operator with Text Value Example

The following SQL statement selects all products with a ProdName beginning with any of the letter BETWEEN 'C' and 'M':

Example

```
SELECT * FROM tblProducts  
WHERE ProdName BETWEEN 'C' AND 'M';
```

DBMS – Database Management System

SQL – Structured Query Language

NOT BETWEEN Operator with Text Value Example

The following SQL statement selects all products with a ProductName beginning with any of the letter NOT BETWEEN 'C' and 'M':

Example

```
SELECT * FROM Products
WHERE ProductName NOT BETWEEN 'C' AND 'M';
```

Sample Table

Below is a selection from the "Orders" table:

orderID	custID	EmpID	OrderDate	ShipperID
1	1	1	01/01/2014	1
2	2	2	01/02/2014	2
3	3	3	01/03/2014	3
4	4	4	01/04/2014	4
5	5	5	01/04/2014	5

BETWEEN Operator with Date Value Example

The following SQL statement selects all orders with an OrderDate BETWEEN '01-Janaury-2015' and '09-March-2015':

Example

```
SELECT * FROM tblOrders
WHERE OrderDate BETWEEN #01/01/2015# AND #10/03/2015#;
```

Notice that the BETWEEN operator can produce different result in different databases!

In some databases, BETWEEN selects fields that are between and excluding the test values.

In other databases, BETWEEN selects fields that are between and including the test values.

And in other databases, BETWEEN 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 operator!

DBMS – Database Management System

SQL – Structured Query Language

Lesson 19. SQL Aliases

SQL aliases are used to temporarily rename a table or a column heading.

SQL Aliases

SQL aliases are used to give a database table, or a column in a table, a temporary name.

Basically aliases are created to make column names more readable.

SQL Alias Syntax for Columns

```
SELECT column_name AS alias_name
FROM table_name;
```

SQL Alias Syntax for Tables

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Customers" table:

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalsoftware College	635000	arabsiiyo	252	Somalia
2	Faarax Jaamac Xuseen	Sahalsoftware College	635001	bosaso	252	Somalia
4	Aamina Xasan Jaamac	Sahalsoftware College	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalsoftware College	635007	Nayroobi	254	Kenya
7	Samadoon Saleebaan Xuseen	Sahalsoftware College	635009	Stavanger	45	Denmark
8	Abdifatah	Sahalsoftware College	635008	Baladweyne	NULL	Somalia

And a selection from the "tblOrders" table:

DBMS – Database Management System

SQL – Structured Query Language

orderID	custID	EmplID	OrderDate	ShipperID
1	1	1	01/01/2014	1
2	2	2	01/02/2014	2
3	3	3	01/03/2014	3
4	4	4	01/04/2014	4
5	5	5	01/04/2014	5

Alias Example for Table Columns

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

Example

```
SELECT CustName AS Customer, ContactName AS "Contact Person" FROM tblCustomers;
```

In the following SQL statement we combine four columns (Address, City, PostalCode, and Country) and create an alias named "Address":

Example

```
SELECT CustName, Address+', '+City+', '+PostalCode+', '+Country AS Address  
FROM tblCustomers;
```

Note: To get the SQL statement above to work in MySQL use the following:

```
SELECT CustName, CONCAT(Address,', ',City,', ',PostalCode,', ',Country) AS  
Address  
FROM tblCustomers;
```

Aliases can be useful when:

DBMS – Database Management System

SQL – Structured Query Language

- 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

Lesson 20. SQL Joins

SQL joins are used to combine rows from two or more tables.

SQL JOIN

An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

The most common type of join is: **SQL INNER JOIN (simple join)**. An SQL INNER JOIN return all rows from multiple tables where the join condition is met.

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

orderID	custID	EmpID	OrderDate	ShipperID
1	1	1	01/01/2014	1
2	2	2	01/02/2014	2
3	3	3	01/03/2014	3
4	4	4	01/04/2014	4
5	5	5	01/04/2014	5

Then, have a look at a selection from the "tblCustomers" table:

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalsoftware College	635000	arabsiiyo	252	Somalia
2	Faarax Jaamac Xuseen	Sahalsoftware College	635001	bosaso	252	Somalia
4	Aamina Xasan Jaamac	Sahalsoftware College	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalsoftware College	635007	Nayroobi	254	Kenya
7	Samadoon Saleebaan Xuseen	Sahalsoftware College	635009	Stavanger	45	Denmark
8	Abdifatah	Sahalsoftware College	635008	Baladweyne	NULL	Somalia

Notice that the "CustID" column in the "Orders" table refers to the "CustID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

DBMS – Database Management System

SQL – Structured Query Language

Then, if we run the following SQL statement (that contains an INNER JOIN):

Example

```
SELECT tblOrders.orderID, tblCustomers.CustName, tblOrders.orderDate
FROM tblOrders
INNER JOIN tblCustomers
ON tblOrders.CustID=tblCustomers.CustID;
```

it will produce something like this:

orderID	CustName	orderDate
1	Maxamed Jaamac Xuseen	01/01/2014
2	Faarax Jaamac Xuseen	01/02/2014
4	Aamina Xasan Jaamac	01/04/2014
5	Faadumo Siciid Jaamac	01/04/2014

Different SQL JOINS

Before we continue with examples, we will list the types the different SQL JOINS you can use:

- **INNER JOIN:** Returns all rows when there is at least one match in BOTH tables
- **LEFT JOIN:** Return all rows from the left table, and the matched rows from the right table
- **RIGHT JOIN:** Return all rows from the right table, and the matched rows from the left table
- **FULL JOIN:** Return all rows when there is a match in ONE of the tables

Lesson 21. SQL INNER JOIN Keyword

SQL INNER JOIN Keyword

The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns in both tables.

SQL INNER JOIN Syntax

```
SELECT column_name(s)
FROM table1
```

DBMS – Database Management System

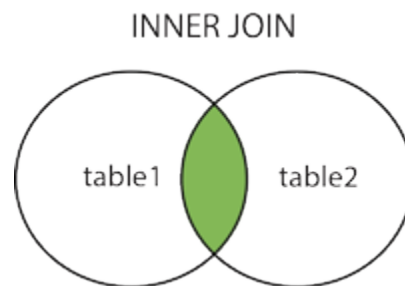
SQL – Structured Query Language

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

or:

```
SELECT column_name(s)  
FROM table1  
JOIN table2  
ON table1.column_name=table2.column_name;
```

PS! INNER JOIN is the same as JOIN.



Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

Below is a selection from the "tblCustomers" table:

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalsoftware College	635000	arabsiiyo	252	Somalia
2	Faarax Jaamac Xuseen	Sahalsoftware College	635001	bosaso	252	Somalia
4	Aamina Xasan Jaamac	Sahalsoftware College	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalsoftware College	635007	Nayroobi	254	Kenya
7	Samadoon Saleebaan Xuseen	Sahalsoftware College	635009	Stavanger	45	Denmark
8	Abdifatah	Sahalsoftware College	635008	Baladweyne	NULL	Somalia

.And a selection from the "tblOrders" table:

DBMS – Database Management System

SQL – Structured Query Language

orderID	custID	EmpID	OrderDate	ShipperID
1	1	1	01/01/2014	1
2	2	2	01/02/2014	2
3	3	3	01/03/2014	3
4	4	4	01/04/2014	4
5	5	5	01/04/2014	5

SQL INNER JOIN Example

The following SQL statement will return all customers with orders:

Example

```
SELECT tblCustomers.CustName, tblOrders.OrderID
FROM tblCustomers
INNER JOIN tblOrders
ON tblCustomers.CustID=Orders.CustID
ORDER BY Customers.CustName;
```

Then Like this

CustName ▲ 1	OrderID
Aamina Xasan Jaamac	4
Faadumo Siciid Jaamac	5
Faarax Jaamac Xuseen	2
Maxamed Jaamac Xuseen	1

Note: The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are rows in the "tblCustomers" table that do not have matches in "tblOrders", these customers will NOT be listed.

Lesson 22. SQL LEFT JOIN Keyword

DBMS – Database Management System

SQL – Structured Query Language

SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.

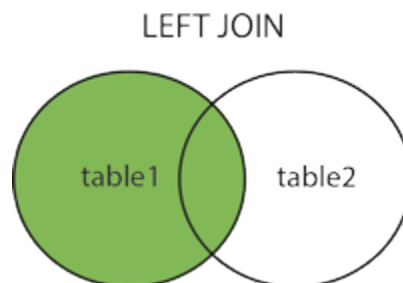
SQL LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name=table2.column_name;
```

or:

```
SELECT column_name(s)
FROM table1
LEFT OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

PS! In some databases LEFT JOIN is called LEFT OUTER JOIN.



Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

Below is a selection from the "tblCustomers" table:

DBMS – Database Management System

SQL – Structured Query Language

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalsoftware College	635000	arabsiiyo	252	Somalia
2	Faarax Jaamac Xuseen	Sahalsoftware College	635001	bosaso	252	Somalia
4	Aamina Xasan Jaamac	Sahalsoftware College	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalsoftware College	635007	Nayroobi	254	Kenya
7	Samadoon Saleebaan Xuseen	Sahalsoftware College	635009	Stavanger	45	Denmark
8	Abdifatah	Sahalsoftware College	635008	Baladweyne	NULL	Somalia

And a selection from the "tblOrders" table:

orderID	custID	EmplID	OrderDate	ShipperID
1	1	1	01/01/2014	1
2	2	2	01/02/2014	2
3	3	3	01/03/2014	3
4	4	4	01/04/2014	4
5	5	5	01/04/2014	5

SQL LEFT JOIN Example

The following SQL statement will return all customers, and any orders they might have:

Example

```
SELECT tblCustomers.CustName, tblOrders.orderID
FROM tblCustomers
LEFT JOIN tblOrders
ON tblCustomers.CustID=tblOrders.CustID
ORDER BY tblCustomers.CustName;
```

Then Like This:

CustName ▲ 1	orderID
Aamina Xasan Jaamac	4
Abdifatah	NULL
Faadumo Siciid Jaamac	5
Faarax Jaamac Xuseen	2
Maxamed Jaamac Xuseen	1
Samadoon Saleebaan Xuseen	NULL

DBMS – Database Management System

SQL – Structured Query Language

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

Lesson 23. SQL RIGHT JOIN Keyword

SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

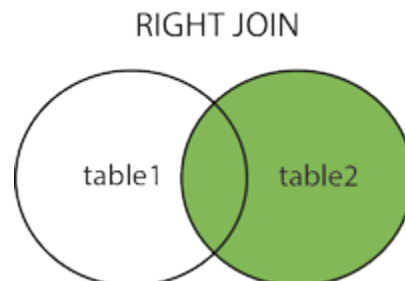
SQL RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name=table2.column_name;
```

or:

```
SELECT column_name(s)
FROM table1
RIGHT OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

PS! In some databases RIGHT JOIN is called RIGHT OUTER JOIN.



Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

DBMS – Database Management System

SQL – Structured Query Language

Below is a selection from the "tblOrders" table:

orderID	custID	EmpID	OrderDate	ShipperID
1	1	1	01/01/2014	1
2	2	2	01/02/2014	2
3	3	3	01/03/2014	3
4	4	4	01/04/2014	4
5	5	5	01/04/2014	5

And a selection from the "Employees" table:

EmpID	LastName	FirstName	BirthDate	Photo	Notes
1	Jaamac	Xasan	12/12/2014	1	IT
2	Siciid	Xuseen	11/12/2014	2	BA
3	Xuseen	Salmaan	10/12/2014	3	Health
4	Maxamuud	Saynab	09/12/2014	4	Social

SQL RIGHT JOIN Example

The following SQL statement will return all employees, and any orders they have placed:

Example

```
SELECT tblOrders.OrderID, tblEmp.FirstName
FROM tblOrders
RIGHT JOIN tblEmp
ON tblOrders.EmpID=tblEmp.EmpID
ORDER BY tblOrders.OrderID;
```

Then Like this:

OrderID	FirstName
1	Xasan
2	Xuseen
3	Salmaan
4	Saynab

DBMS – Database Management System

SQL – Structured Query Language

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

Lesson 24. SQL FULL OUTER JOIN Keyword

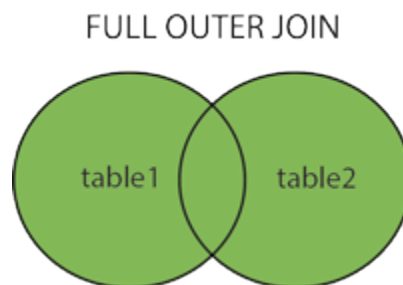
SQL FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).

The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

SQL FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name=table2.column_name;
```



Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

Below is a selection from the "tblcustomers" table:

DBMS – Database Management System

SQL – Structured Query Language

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalssoftware College	635000	arabsiiyo	252	Somalia
2	Faarax Jaamac Xuseen	Sahalssoftware College	635001	bosaso	252	Somalia
4	Aamina Xasan Jaamac	Sahalssoftware College	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalssoftware College	635007	Nayroobi	254	Kenya
7	Samadoon Saleebaan Xuseen	Sahalssoftware College	635009	Stavanger	45	Denmark
8	Abdifatah	Sahalssoftware College	635008	Baladweyne	NULL	Somalia

And a selection from the "Orders" table:

orderID	custID	EmplID	OrderDate	ShipperID
1	1	1	01/01/2014	1
2	2	2	01/02/2014	2
3	3	3	01/03/2014	3
4	4	4	01/04/2014	4
5	5	5	01/04/2014	5

SQL FULL OUTER JOIN Example

The following SQL statement selects all tblcustomers, 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
Alfreds Futterkiste	
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	10365
	10382

DBMS – Database Management System

SQL – Structured Query Language

10351

Note: The FULL OUTER JOIN keyword returns all the rows from the left table (Customers), and all the rows from the right table (Orders). 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.

Warning Full Join:

In MySQL doesn't exist FULL OUTER JOIN keyword. But you can use like this:

```
SELECT * FROM Customers C
```

```
Left join Orders O on C.CustomerID=O.OrderID
```

Union

```
Select * From Customers C Right Join Orders O On C.CustomerID=O.OrderID
```

Lesson 25. SQL UNION Operator

The SQL UNION operator combines the result of two or more SELECT statements.

DBMS – Database Management System

SQL – Structured Query Language

The SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

Notice that each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types. Also, the columns in each SELECT statement must be in the same order.

SQL UNION Syntax

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

Note: The UNION operator selects only distinct values by default. To allow duplicate values, use the ALL keyword with UNION.

SQL UNION ALL Syntax

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

PS: The column names in the result-set of a UNION are usually equal to the column names in the first SELECT statement in the UNION.

Demo Database

In this tutorial we will use the well-known SDS_V14 sample database.

Below is a selection from the "tblCustomers" table:

CustID	CustName	ContactName	Address	City	PostalCode	Country
1	Maxamed Jaamac Xuseen	Sahalsoftware College	635000	arabsiiyo	252	Somalia
2	Faarax Jaamac Xuseen	Sahalsoftware College	635001	bosaso	252	Somalia
4	Aamina Xasan Jaamac	Sahalsoftware College	635003	Laascaanood	252	Somalia
5	Faadumo Siciid Jaamac	Sahalsoftware College	635007	Nayroobi	254	Kenya
7	Samadoon Saleebaan Xuseen	Sahalsoftware College	635009	Stavanger	45	Denmark
8	Abdifatah	Sahalsoftware College	635008	Baladweyne	NULL	Somalia

And a selection from the "Suppliers" table:

DBMS – Database Management System

SQL – Structured Query Language

supplD	suppName	contactName	address	city	postalCode	country
1	Hormuud	Sahalsoftware	635100	Laascaanood	252	Somalia
2	Hillaac	Sahalsoftware	635101	Garoowe	252	Somalia
3	Golis	Sahalsoftware	635102	Bosaso	252	Somalia
4	Daarasalaam	Sahalsoftware	635103	Bosaso	252	Somalia
5	Dabayl	Sahalsoftware	635104	Laascaanood	252	Somalia

SQL UNION Example

The following SQL statement selects all the **different** cities (only distinct values) from the "Customers" and the "Suppliers" tables:

Example

```
SELECT City FROM tblCustomers
UNION
SELECT City FROM tblSupp
ORDER BY City;
```

Note: UNION cannot be used to list ALL cities from the two tables. If several customers and suppliers share the same city, each city will only be listed once. UNION selects only distinct values. Use UNION ALL to also select duplicate values!

SQL UNION ALL Example

The following SQL statement uses UNION ALL to select **all** (duplicate values also) cities from the "Customers" and "Suppliers" tables:

Example

```
SELECT City FROM tblCustomers
UNION ALL
SELECT City FROM tblSupp
ORDER BY City;
```

SQL UNION ALL With WHERE

The following SQL statement uses UNION ALL to select **all** (duplicate values also) **Laascaanood** cities from the "tblCustomers" and "tblSuppliers" tables:

DBMS – Database Management System

SQL – Structured Query Language

Example

```
SELECT City, Country FROM tblCustomers
WHERE Country='Somalia'
UNION ALL
SELECT City, Country FROM tblSupp
WHERE Country='Somalia'
ORDER BY City;
```

Lesson 26. SQL **SELECT INTO** Statement

With SQL, you can copy information from one table into another.

The SELECT INTO statement copies data from one table and inserts it into a new table.

The SQL **SELECT INTO** Statement

The SELECT INTO statement selects data from one table and inserts it into a new table.

SQL **SELECT INTO** Syntax

We can copy all columns into the new table:

```
SELECT *
INTO newtable [IN externaldb]
FROM table1;
```

Or we can copy only the columns we want into the new table:

```
SELECT column_name(s)
INTO newtable [IN externaldb]
FROM table1;
```

The new table will be created with the column-names and types as defined in the SELECT statement. You can apply new names using the AS clause.

DBMS – Database Management System

SQL – Structured Query Language

SQL SELECT INTO Examples

Create a backup copy of Customers:

```
SELECT *  
INTO CustomersBackup2013  
FROM Customers;
```

Use the IN clause to copy the table into another database:

```
SELECT *  
INTO CustomersBackup2013 IN 'Backup.mdb'  
FROM Customers;
```

Copy only a few columns into the new table:

```
SELECT CustomerName, ContactName  
INTO CustomersBackup2013  
FROM Customers;
```

Copy only the German customers into the new table:

```
SELECT *  
INTO CustomersBackup2013  
FROM Customers  
WHERE Country='Germany';
```

Copy data from more than one table into the new table:

```
SELECT Customers.CustomerName, Orders.OrderID  
INTO CustomersOrderBackup2013  
FROM Customers  
LEFT JOIN Orders  
ON Customers.CustomerID=Orders.CustomerID;
```

Tip: The SELECT INTO statement can also be used to create a new, empty table using the schema of another. Just add a WHERE clause that causes the query to return no data:

```
SELECT *  
INTO newtable  
FROM table1  
WHERE 1=0;
```

DBMS – Database Management System

SQL – Structured Query Language

Lesson 27. SQL **INSERT INTO SELECT** Statement

With SQL, you can copy information from one table into another.

The INSERT INTO SELECT statement copies data from one table and inserts it into an existing table.

The SQL INSERT INTO SELECT Statement

The INSERT INTO SELECT statement selects data from one table and inserts it into an existing table. Any existing rows in the target table are unaffected.

SQL INSERT INTO SELECT Syntax

We can copy all columns from one table to another, existing table:

```
INSERT INTO table2
SELECT * FROM table1;
```

Or we can copy only the columns we want to into another, existing table:

```
INSERT INTO table2
(column_name(s))
SELECT column_name(s)
FROM table1;
```

DBMS – Database Management System

SQL – Structured Query Language

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode
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	Postal Code	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	Londona	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA

SQL INSERT INTO SELECT Examples

Copy only a few columns from "Suppliers" into "Customers":

DBMS – Database Management System

SQL – Structured Query Language

Example

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

Copy only the German suppliers into "Customers":

Example

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

Lesson 28. SQL Constraints

SQL Constraints

SQL constraints are used to specify rules for the data in a table.

If there is any violation between the constraint and the data action, the action is aborted by the constraint.

Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

SQL CREATE TABLE + CONSTRAINT Syntax

```
CREATE TABLE table_name
(
  column_name1 data_type(size) constraint_name,
  column_name2 data_type(size) constraint_name,
  column_name3 data_type(size) constraint_name,
  ....
);
```

In SQL, we have the following constraints:

- **NOT NULL** - Indicates that a column cannot store NULL value
- **UNIQUE** - Ensures that each row for a column must have a unique value
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have an unique identity which helps to find a particular record in a table more easily and quickly
- **FOREIGN KEY** - Ensure the referential integrity of the data in one table to match values in another table
- **CHECK** - Ensures that the value in a column meets a specific condition

DBMS – Database Management System

SQL – Structured Query Language

- **DEFAULT** - Specifies a default value when specified none for this column

The next chapters will describe each constraint in detail.

Lesson 29. SQL **NOT NULL** Constraint

By default, a table column can hold NULL values.

SQL **NOT NULL** Constraint

The NOT NULL constraint enforces a column to NOT accept NULL values.

The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.

The following SQL enforces the "P_Id" column and the "LastName" column to not accept NULL values:

Example

```
CREATE TABLE PersonsNotNull
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

DBMS – Database Management System

SQL – Structured Query Language

Lesson 30. SQL **UNIQUE** Constraint

SQL **UNIQUE** Constraint

The **UNIQUE** constraint uniquely identifies each record in a database table.

The **UNIQUE** and **PRIMARY KEY** constraints both provide a guarantee for uniqueness for a column or set of columns.

A **PRIMARY KEY** constraint automatically has a **UNIQUE** constraint defined on it.

Note that 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 "P_Id" column when the "Persons" table is created:

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL UNIQUE,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

MySQL:

DBMS – Database Management System

SQL – Structured Query Language

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
UNIQUE (P_Id)
)
```

To allow naming of a UNIQUE constraint, and for defining a UNIQUE constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName)
)
```

SQL UNIQUE Constraint on ALTER TABLE

To create a UNIQUE constraint on the "P_Id" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD UNIQUE (P_Id)
```

To allow naming of a UNIQUE constraint, and for defining a UNIQUE constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName)
```

DBMS – Database Management System

SQL – Structured Query Language

To DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

MySQL:

```
ALTER TABLE Persons  
DROP INDEX uc_PersonID
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons  
DROP CONSTRAINT uc_PersonID
```

Lesson 31. SQL PRIMARY KEY Constraint

SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a database table.

Primary keys must contain unique values.

A primary key column cannot contain NULL values.

Most tables should have a primary key, and each table can have only ONE primary key.

SQL PRIMARY KEY Constraint on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "P_Id" column when the "Persons" table is created:

MySQL:

```
CREATE TABLE Persons  
(  
P_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255),
```

DBMS – Database Management System

SQL – Structured Query Language

```
PRIMARY KEY (P_Id)
)
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

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
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)
)
```

Note: In the example above there is only ONE PRIMARY KEY (pk_PersonID). However, the value of the pk_PersonID is made up of two columns (P_Id and LastName).

SQL PRIMARY KEY Constraint on ALTER TABLE

To create a PRIMARY KEY constraint on the "P_Id" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD PRIMARY KEY (P_Id)
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

DBMS – Database Management System

SQL – Structured Query Language

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)
```

Note: If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created).

To 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_PersonID
```

Lesson 32. SQL FOREIGN KEY Constraint

SQL FOREIGN KEY Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

Let's illustrate the foreign key with an example. Look at the following two tables:

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

DBMS – Database Management System

SQL – Structured Query Language

3	Pettersen	Kari	Storgt 20	Stavange
---	-----------	------	-----------	----------

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Note that the "P_Id" column in the "Orders" table points to the "P_Id" column in the "Persons" table.

The "P_Id" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "P_Id" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

SQL FOREIGN KEY Constraint on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "P_Id" column when the "Orders" table is created:

MySQL:

```
CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
PRIMARY KEY (O_Id),
```

DBMS – Database Management System

SQL – Structured Query Language

```
FOREIGN KEY (P_Id) REFERENCES Persons(P_Id)
)
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders
(
O_Id int NOT NULL PRIMARY KEY,
OrderNo int NOT NULL,
P_Id int FOREIGN KEY REFERENCES Persons(P_Id)
)
```

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
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
PRIMARY KEY (O_Id),
CONSTRAINT fk_PerOrders FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
)
```

SQL FOREIGN KEY Constraint on ALTER TABLE

To create a FOREIGN KEY constraint on the "P_Id" column when the "Orders" table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
```

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:

DBMS – Database Management System

SQL – Structured Query Language

```
ALTER TABLE Orders
ADD CONSTRAINT fk_PerOrders
FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
```

To DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL:

MySQL:

```
ALTER TABLE Orders
DROP FOREIGN KEY fk_PerOrders
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
DROP CONSTRAINT fk_PerOrders
```

Lesson 33. SQL CHECK Constraint

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 single column it allows 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 Constraint on CREATE TABLE

The following SQL creates a CHECK constraint on the "P_Id" column when the "Persons" table is created. The CHECK constraint specifies that the column "P_Id" must only include integers greater than 0.

MySQL:

DBMS – Database Management System

SQL – Structured Query Language

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CHECK (P_Id>0)
)
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL CHECK (P_Id>0),
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

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
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT chk_Person CHECK (P_Id>0 AND City='Sandnes')
)
```

SQL CHECK Constraint on ALTER TABLE

To create a CHECK constraint on the "P_Id" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

DBMS – Database Management System

SQL – Structured Query Language

```
ALTER TABLE Persons  
ADD CHECK (P_Id>0)
```

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_Person CHECK (P_Id>0 AND City='Sandnes')
```

To DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons  
DROP CONSTRAINT chk_Person
```

MySQL:

```
ALTER TABLE Persons  
DROP CHECK chk_Person
```

Lesson 34. SQL **DEFAULT** Constraint

DBMS – Database Management System

SQL – Structured Query Language

SQL DEFAULT Constraint

The DEFAULT constraint is used to insert a default value into a column.

The default value will be added to all new records, if no other value is specified.

SQL DEFAULT Constraint on CREATE TABLE

The following SQL creates a DEFAULT constraint on the "City" column when the "Persons" table is created:

My SQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255) DEFAULT 'Sandnes'
)
```

The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE():

```
CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
OrderDate date DEFAULT GETDATE()
)
```

SQL DEFAULT Constraint on ALTER TABLE

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

MySQL:

DBMS – Database Management System

SQL – Structured Query Language

```
ALTER TABLE Persons  
ALTER City SET DEFAULT 'SANDNES'
```

SQL Server / MS Access:

```
ALTER TABLE Persons  
ALTER COLUMN City SET DEFAULT 'SANDNES'
```

Oracle:

```
ALTER TABLE Persons  
MODIFY City DEFAULT 'SANDNES'
```

To 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
```

Lesson 35. SQL CREATE INDEX Statement

The CREATE INDEX statement is used to create indexes in tables.

Indexes allow the database application to find data fast; without reading the whole table.

DBMS – Database Management System

SQL – Structured Query Language

Indexes

An index can be created in a table to find data more quickly and efficiently.

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 you should only create indexes on columns (and tables) that will be frequently searched against.

SQL CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

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

SQL CREATE UNIQUE INDEX Syntax

Creates a unique index on a table. Duplicate values are not allowed:

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

Note: The syntax for creating indexes varies amongst different databases. Therefore: Check the syntax for creating indexes in your database.

CREATE INDEX Example

The SQL statement below creates an index named "PIndex" on the "LastName" column in the "Persons" table:

```
CREATE INDEX PIndex  
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 PIndex  
ON Persons (LastName, FirstName)
```

DBMS – Database Management System

SQL – Structured Query Language

Lesson 36. SQL DROP INDEX, DROP TABLE, and DROP DATABASE

Indexes, tables, and databases can easily be deleted/removed with the DROP statement.

The DROP INDEX Statement

The DROP INDEX statement is used to delete an index in a table.

DROP INDEX Syntax for MS Access:

```
DROP INDEX index_name ON table_name
```

DROP INDEX Syntax for MS SQL Server:

```
DROP INDEX table_name.index_name
```

DROP INDEX Syntax for DB2/Oracle:

```
DROP INDEX index_name
```

DROP INDEX Syntax for MySQL:

```
ALTER TABLE table_name DROP INDEX index_name
```

The DROP TABLE Statement

The DROP TABLE statement is used to delete a table.

```
DROP TABLE table_name
```

The DROP DATABASE Statement

The DROP DATABASE statement is used to delete a database.

```
DROP DATABASE database_name
```

DBMS – Database Management System

SQL – Structured Query Language

The TRUNCATE TABLE Statement

What if we only want to delete the data inside the table, and not the table itself?

Then, use the TRUNCATE TABLE statement:

```
TRUNCATE TABLE table_name
```

Lesson 37. SQL **ALTER TABLE** Statement

The ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

SQL ALTER TABLE Syntax

To add a column in a table, use the following syntax:

DBMS – Database Management System

SQL – Structured Query Language

```
ALTER TABLE table_name  
ADD column_name datatype
```

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
```

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:

```
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:

P_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.

We use the following SQL statement:

DBMS – Database Management System

SQL – Structured Query Language

```
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:

P_Id	LastName	FirstName	Address	City	DateOf
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-digit or four-digit format.

DROP COLUMN Example

Next, we want to delete the column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

```
ALTER TABLE Persons  
DROP COLUMN DateOfBirth
```

DBMS – Database Management System

SQL – Structured Query Language

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavange

Lesson 38. SQL **AUTO INCREMENT** Field

Auto-increment allows a unique number to be generated when a new record is inserted into a table.

AUTO INCREMENT a Field

Very often we would like the value of the primary key field to be created automatically every time a new record is inserted.

We would like to create an auto-increment field in a table.

Syntax for MySQL

The following SQL statement defines the "ID" column to be an auto-increment primary key field in the "Persons" table:

DBMS – Database Management System

SQL – Structured Query Language

```
CREATE TABLE Persons
(
  ID int NOT NULL AUTO_INCREMENT,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255),
  PRIMARY KEY (ID)
)
```

MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

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 "ID" 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 "ID" 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 "ID" column to be an auto-increment primary key field in the "Persons" table:

```
CREATE TABLE Persons
(
  ID int IDENTITY(1,1) PRIMARY KEY,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255)
)
```

The MS SQL Server uses the IDENTITY keyword to perform an auto-increment feature.

DBMS – Database Management System

SQL – Structured Query Language

In the example above, the starting value for IDENTITY is 1, and it will increment by 1 for each new record.

Tip: To specify that the "ID" column should start at value 10 and increment by 5, change it to IDENTITY(10,5).

To insert a new record into the "Persons" table, we will NOT have to specify a value for the "ID" 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 "ID" 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 Access

The following SQL statement defines the "ID" column to be an auto-increment primary key field in the "Persons" table:

```
CREATE TABLE Persons
(
ID Integer PRIMARY KEY AUTOINCREMENT,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

The MS Access uses the AUTOINCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTOINCREMENT is 1, and it will increment by 1 for each new record.

Tip: To specify that the "ID" column should start at value 10 and increment by 5, change the autoincrement to AUTOINCREMENT(10,5).

To insert a new record into the "Persons" table, we will NOT have to specify a value for the "ID" column (a unique value will be added automatically):

```
INSERT INTO Persons (FirstName,LastName)
VALUES ('Lars', 'Monsen')
```

DBMS – Database Management System

SQL – Structured Query Language

The SQL statement above would insert a new record into the "Persons" table. The "P_Id" 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 Oracle

In Oracle the code is a little bit more tricky.

You will have to create an auto-increment field with the sequence object (this object generates a number sequence).

Use the following CREATE SEQUENCE syntax:

```
CREATE SEQUENCE seq_person
MINVALUE 1
START WITH 1
INCREMENT BY 1
CACHE 10
```

The code above creates a sequence object called seq_person, that starts with 1 and will increment by 1. It will also cache up to 10 values for performance. The cache option specifies how many sequence values will be stored in memory for faster access.

To insert a new record into the "Persons" table, we will have to use the nextval function (this function retrieves the next value from seq_person sequence):

```
INSERT INTO Persons (ID,FirstName,LastName)
VALUES (seq_person.nextval, 'Lars', 'Monsen')
```

The SQL statement above would insert a new record into the "Persons" table. The "ID" column would be assigned the next number from the seq_person sequence. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

Lesson 39. SQL Views

A view is a virtual table.

This chapter shows how to create, update, and delete a view.

DBMS – Database Management System

SQL – Structured Query Language

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 functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

SQL CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

Note: A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

SQL CREATE VIEW Examples

If you have the SDS_14 database you can see that it has several views installed by default.

The view "Current Product List" lists all active products (products that are not discontinued) from the "Products" table. The view is created with the following SQL:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID,ProductName
FROM Products
WHERE Discontinued=No
```

We can query the view above as follows:

```
SELECT * FROM [Current Product List]
```

Another view in the SDS_14 sample database selects every product in the "Products" table with a unit price higher than the average unit price:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName,UnitPrice
FROM Products
WHERE UnitPrice>(SELECT AVG(UnitPrice) FROM Products)
```

DBMS – Database Management System

SQL – Structured Query Language

We can query the view above as follows:

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

Another view in the SDS_14 database calculates the total sale for each category in 1997. Note that this view selects its data from another view called "Product Sales for 1997":

```
CREATE VIEW [Category Sales For 1997] AS  
SELECT DISTINCT CategoryName,Sum(ProductSales) AS CategorySales  
FROM [Product Sales for 1997]  
GROUP BY CategoryName
```

We can query the view above as follows:

```
SELECT * FROM [Category Sales For 1997]
```

We can also add a condition to the query. Now we want to see the total sale only for the category "Beverages":

```
SELECT * FROM [Category Sales For 1997]  
WHERE CategoryName='Beverages'
```

SQL Updating a View

You can update a view by using the following syntax:

SQL CREATE OR REPLACE VIEW Syntax

```
CREATE OR REPLACE VIEW view_name AS  
SELECT column_name(s)  
FROM table_name  
WHERE condition
```

Now we want to add the "Category" column to the "Current Product List" view. We will update the view with the following SQL:

```
CREATE VIEW [Current Product List] AS  
SELECT ProductID,ProductName,Category  
FROM Products  
WHERE Discontinued=No
```

DBMS – Database Management System

SQL – Structured Query Language

SQL Dropping a View

You can delete a view with the DROP VIEW command.

SQL DROP VIEW Syntax

```
DROP VIEW view_name
```

Lesson 40. SQL Date Functions

SQL Dates



The most difficult part when working with dates is to be sure that the format of the date you a 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 complicated.

Before talking about the complications of querying for dates, we will look at the most important built-in functions for working with dates.

MySQL Date Functions

The following table lists the most important built-in date functions in MySQL:

Function	Description
<u>NOW()</u>	Returns the current date and time
<u>CURDATE()</u>	Returns the current date
<u>CURTIME()</u>	Returns the current time
<u>DATE()</u>	Extracts the date part of a date or date/time expression

DBMS – Database Management System

SQL – Structured Query Language

<u>EXTRACT()</u>	Returns a single part of a date/time
<u>DATE_ADD()</u>	Adds a specified time interval to a date
<u>DATE_SUB()</u>	Subtracts a specified time interval from a date
<u>DATEDIFF()</u>	Returns the number of days between two dates
<u>DATE_FORMAT()</u>	Displays date/time data in different formats

SQL Server Date Functions

The following table lists the most important built-in date functions in SQL Server:

Function	Description
<u>GETDATE()</u>	Returns the current date and time
<u>DATEPART()</u>	Returns a single part of a date/time
<u>DATEADD()</u>	Adds or subtracts a specified time interval from a date
<u>DATEDIFF()</u>	Returns the time between two dates
<u>CONVERT()</u>	Displays date/time data in different formats

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

DBMS – Database Management System

SQL – Structured Query Language

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!

For an overview of all data types available, go to our complete [Data Types reference](#).

SQL Working with Dates



You can compare two dates easily if there is no time component involved!

Assume we have the following "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
---------	-------------	-----------

DBMS – Database Management System

SQL – Structured Query Language

1	Geitost	2008-11-11
3	Mozzarella di Giovanni	2008-11-11

Now, assume that the "Orders" table looks like this (notice the 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: If you want to keep your queries simple and easy to maintain, do not allow time components in your dates!

DBMS – Database Management System

SQL – Structured Query Language

Lesson 41. SQL NULL Values

NULL values represent missing unknown data.

By default, a table column can hold NULL values.

This chapter will explain the IS NULL and IS NOT NULL operators.

SQL NULL Values

If a column in a table is optional, we can insert a new record or update an existing record without adding a value to this column. This means that the field will be saved with a NULL value.

NULL values are treated differently from other values.

NULL is used as a placeholder for unknown or inapplicable values.



Note: It is not possible to compare NULL and 0; they are not equivalent.

SQL Working with NULL Values

Look at the following "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola		Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

DBMS – Database Management System

SQL – Structured Query Language

3	Pettersen	Kari		Stavanger
---	-----------	------	--	-----------

Suppose that the "Address" column in the "Persons" table is optional. This means that if we insert a record with no value for the "Address" column, the "Address" column will be saved with a NULL value.

How can we 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.

SQL IS NULL

How do we select only the records with NULL values in the "Address" column?

We will have to use the IS NULL operator:

```
SELECT LastName,FirstName,Address FROM Persons
WHERE Address IS NULL
```

The result-set will look like this:

LastName	FirstName	Address
Hansen	Ola	
Pettersen	Kari	



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

SQL IS NOT NULL

How do we select only the records with no NULL values in the "Address" column?

DBMS – Database Management System

SQL – Structured Query Language

We will have to use the IS NOT NULL operator:

```
SELECT LastName,FirstName,Address FROM Persons  
WHERE Address IS NOT NULL
```

The result-set will look like this:

LastName	FirstName	Address
Svendson	Tove	Borgvn 23

In the next chapter we will look at the ISNULL(), NVL(), IFNULL() and COALESCE() functions.

Lesson 42. SQL NULL Functions

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

Look at the following "Products" table:

P_Id	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder
1	Jarlsberg	10.45	16	15
2	Mascarpone	32.56	23	
3	Gorgonzola	15.67	9	20

Suppose that the "UnitsOnOrder" column is optional, and may contain NULL values.

We have the following SELECT statement:

```
SELECT ProductName,UnitPrice*(UnitsInStock+UnitsOnOrder)  
FROM Products
```

In the example above, if any of the "UnitsOnOrder" values are NULL, the result is NULL.

Microsoft's ISNULL() function is used to specify how we want to treat NULL values.

DBMS – Database Management System

SQL – Structured Query Language

The NVL(), IFNULL(), and COALESCE() functions can also be used to achieve the same result.

In this case we want NULL values to be zero.

Below, if "UnitsOnOrder" is NULL it will not harm the calculation, because ISNULL() returns a zero if the value is NULL:

MS Access

```
SELECT
ProductName,UnitPrice*(UnitsInStock+IIF(ISNULL(UnitsOnOrder),0,UnitsOnOrder))
FROM Products
```

SQL Server

```
SELECT ProductName,UnitPrice*(UnitsInStock+ISNULL(UnitsOnOrder,0))
FROM Products
```

Oracle

Oracle does not have an ISNULL() function. However, we can use the NVL() function to achieve the same result:

```
SELECT ProductName,UnitPrice*(UnitsInStock+NVL(UnitsOnOrder,0))
FROM Products
```

MySQL

MySQL does have an ISNULL() function. However, it works a little bit different from Microsoft's ISNULL() function.

In MySQL we can use the IFNULL() function, like this:

```
SELECT ProductName,UnitPrice*(UnitsInStock+IFNULL(UnitsOnOrder,0))
FROM Products
```

or we can use the COALESCE() function, like this:

```
SELECT ProductName,UnitPrice*(UnitsInStock+COALESCE(UnitsOnOrder,0))
FROM Products
```

Lesson 43. SQL General Data Types

DBMS – Database Management System

SQL – Structured Query Language

A data type defines what kind of value a column can contain.

SQL General Data Types

Each column in a database table is required to have a name and a data type.

SQL developers have to decide what types of data will be stored inside each and every table column when creating a SQL table. The data type is a label and 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.

The following table lists the general data types in SQL:

Data type	Description
CHARACTER(n)	Character string. Fixed-length n
VARCHAR(n) or CHARACTER VARYING(n)	Character string. Variable length. Maximum length n
BINARY(n)	Binary string. Fixed-length n
BOOLEAN	Stores TRUE or FALSE values
VARBINARY(n) or BINARY VARYING(n)	Binary string. Variable length. Maximum length n
INTEGER(p)	Integer numerical (no decimal). Precision p
SMALLINT	Integer numerical (no decimal). Precision 5
INTEGER	Integer numerical (no decimal). Precision 10
BIGINT	Integer numerical (no decimal). Precision 19
DECIMAL(p,s)	Exact numerical, precision p, scale s. Example: decimal(5,2) is a number with 5 digits before the decimal and 2 digits after the decimal
NUMERIC(p,s)	Exact numerical, precision p, scale s. (Same as DECIMAL)

DBMS – Database Management System

SQL – Structured Query Language

FLOAT(p)	Approximate numerical, mantissa precision p. A floating number in base exponential notation. The size argument for this type consists of a single specifying the minimum precision
REAL	Approximate numerical, mantissa precision 7
FLOAT	Approximate numerical, mantissa precision 16
DOUBLE PRECISION	Approximate numerical, mantissa precision 16
DATE	Stores year, month, and day values
TIME	Stores hour, minute, and second values
TIMESTAMP	Stores year, month, day, hour, minute, and second values
INTERVAL	Composed of a number of integer fields, representing a period of time, de the type of interval
ARRAY	A set-length and ordered collection of elements
MULTISET	A variable-length and unordered collection of elements
XML	Stores XML data

SQL Data Type Quick Reference

However, different databases offer different choices for the data type definition.

The following table shows some of the common names of data types between the various database platforms:

Data type	Access	SQLServer	Oracle	MySQL	PostgreSQL
<i>boolean</i>	Yes/No	Bit	Byte	N/A	Bo

DBMS – Database Management System

SQL – Structured Query Language

<i>integer</i>	Number (integer)	Int	Number	Int Integer	In In
<i>float</i>	Number (single)	Float Real	Number	Float	NU
<i>currency</i>	Currency	Money	N/A	N/A	Me
<i>string (fixed)</i>	N/A	Char	Char	Char	Ch
<i>string (variable)</i>	Text (<256) Memo (65k+)	Varchar	Varchar Varchar2	Varchar	Va
<i>binary object</i>	OLE Object Memo	Binary (fixed up to 8K) Varbinary (<8K) Image (<2GB)	Long Raw	Blob Text	Bi Va



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!**

Lesson 44. SQL Data Types for Various DBs

Data types and ranges for Microsoft Access, MySQL and SQL Server.

DBMS – Database Management System

SQL – Structured Query Language

Microsoft Access Data Types

Data type	Description
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
Integer	Allows whole numbers between -32,768 and 32,767
Long	Allows whole numbers between -2,147,483,648 and 2,147,483,647
Single	Single precision floating-point. Will handle most decimals
Double	Double precision floating-point. Will handle most decimals
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
AutoNumber	AutoNumber fields automatically give each record its own number, usually starting at 1
Date/Time	Use for dates and times
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
Ole Object	Can store pictures, audio, video, or other BLOBs (Binary Large Objects)
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

DBMS – Database Management System

SQL – Structured Query Language

MySQL Data Types

In MySQL there are three main types : text, number, and Date/Time types.

Text types:

Data type	Description
CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). maximum size is specified in parenthesis. Can store up to 255 characters. Note: a greater value than 255 it will be converted to a TEXT type
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 characters
BLOB	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
LOB	For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data
ENUM(x,y,z,etc.)	Let you enter a list of possible values. You can list up to 65535 values in an ENUM. If a value is inserted that is not in the list, a blank value will be inserted. Note: The values are sorted in the order you enter them. You enter the possible values in this format: ENUM('X','Y','Z')
SET	Similar to ENUM except that SET may contain up to 64 list items and can store one choice

Number types:

DBMS – Database Management System

SQL – Structured Query Language

Data type	Description
TINYINT(size)	-128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis
SMALLINT(size)	-32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis
MEDIUMINT(size)	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis
INT(size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
BIGINT(size)	-9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED*. The maximum number of digits may be specified in parenthesis
FLOAT(size,d)	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DOUBLE(size,d)	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DECIMAL(size,d)	A DOUBLE stored as a string, allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

*The integer types have an extra option called UNSIGNED. Normally, the integer goes from an negative to positive value. Adding the UNSIGNED attribute will move that range up so it starts at zero instead of a negative number.

Date types:

Data type	Description
DATE()	A date. Format: YYYY-MM-DD Note: The supported range is from '1000-01-01' to '9999-12-31'

DBMS – Database Management System

SQL – Structured Query Language

DATETIME()	*A date and time combination. Format: YYYY-MM-DD HH:MI:SS Note: The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP()	*A timestamp. TIMESTAMP values are stored as the number of seconds since the epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MI:SS Note: The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:00:00' UTC
TIME()	A time. Format: HH:MI:SS Note: The supported range is from '-838:59:59' to '838:59:59'
YEAR()	A year in two-digit or four-digit format. Note: Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

*Even if DATETIME and TIMESTAMP return the same format, they work very differently. In an INSERT or UPDATE query, the TIMESTAMP automatically set itself to the current date and time. TIMESTAMP also accepts various formats, like YYYYMMDDHHMISS, YYMMDDHHMISS, YYYYMMDD, or YYMMDD.

SQL Server Data Types

String types:

Data type	Description	Storage
char(n)	Fixed width character string. Maximum 8,000 characters	Defined width
varchar(n)	Variable width character string. Maximum 8,000 characters	2 bytes + n chars
varchar(max)	Variable width character string. Maximum 1,073,741,824 characters	2 bytes + n chars
Text	Variable width character string. Maximum 2GB of text data	4 bytes + n chars

DBMS – Database Management System

SQL – Structured Query Language

Nchar	Fixed width Unicode string. Maximum 4,000 characters	Defined width
Nvarchar	Variable width Unicode string. Maximum 4,000 characters	
nvarchar(max)	Variable width Unicode string. Maximum 536,870,912 characters	
Ntext	Variable width Unicode string. Maximum 2GB of text data	
Bit	Allows 0, 1, or NULL	
binary(n)	Fixed width binary string. Maximum 8,000 bytes	
Varbinary	Variable width binary string. Maximum 8,000 bytes	
varbinary(max)	Variable width binary string. Maximum 2GB	
Image	Variable width binary string. Maximum 2GB	

Number types:

Data type	Description
Tinyint	Allows whole numbers from 0 to 255
Smallint	Allows whole numbers between -32,768 and 32,767
Int	Allows whole numbers between -2,147,483,648 and 2,147,483,647
Bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807
decimal(p,s)	<p>Fixed precision and scale numbers.</p> <p>Allows numbers from $-10^{38} + 1$ to $10^{38} - 1$.</p> <p>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.</p>

DBMS – Database Management System

SQL – Structured Query Language

	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)	<p>Fixed precision and scale numbers.</p> <p>Allows numbers from $-10^{38} + 1$ to $10^{38} - 1$.</p> <p>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.</p> <p>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</p>
Smallmoney	Monetary data from -214,748.3648 to 214,748.3647
Money	Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807
float(n)	<p>Floating precision number data from $-1.79E + 308$ to $1.79E + 308$.</p> <p>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.</p>
Real	Floating precision number data from $-3.40E + 38$ to $3.40E + 38$

Date types:

Data type	Description
Datetime	From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds
datetime2	From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds
Smalldatetime	From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute
Date	Store a date only. From January 1, 0001 to December 31, 9999
Time	Store a time only to an accuracy of 100 nanoseconds

DBMS – Database Management System

SQL – Structured Query Language

Datetimeoffset	The same as datetime2 with the addition of a time zone offset
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
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

Lesson 45. SQL Functions

SQL has many built-in functions for performing calculations on data.

SQL Aggregate Functions

SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- FIRST() - Returns the first value
- LAST() - Returns the last value
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum

DBMS – Database Management System

SQL – Structured Query Language

SQL Scalar functions

SQL scalar functions return a single value, based on the input value.

Useful scalar functions:

- UCASE() - Converts a field to upper case
- LCASE() - Converts a field to lower case
- MID() - Extract characters from a text field
- LEN() - Returns the length of a text field
- ROUND() - Rounds a numeric field to the number of decimals specified
- NOW() - Returns the current system date and time
- FORMAT() - Formats how a field is to be displayed

Tip: The aggregate functions and the scalar functions will be explained in details in the next chapters.

Lesson 46. SQL **AVG()** Function

The **AVG()** Function

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

SQL **AVG()** Syntax

```
SELECT AVG(column_name) FROM table_name
```

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Products" table:

ProductID	ProductName	SupplierID	CategoryID	Unit
1	Chais	1	1	10 boxes x 20 bag

DBMS – Database Management System

SQL – Structured Query Language

2	Chang	1	1	24 - 12 oz bottles
3	Aniseed Syrup	1	2	12 - 550 ml bottle
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars
5	Chef Anton's Gumbo Mix	2	2	36 boxes

SQL AVG() Example

The following SQL statement gets the average value of the "Price" column from the "Products" table:

Example

```
SELECT AVG(Price) AS PriceAverage FROM Products;
```

The following SQL statement selects the "ProductName" and "Price" records that have an above average price:

Example

```
SELECT ProductName, Price FROM Products  
WHERE Price > (SELECT AVG(Price) FROM Products);
```

Lesson 47. SQL COUNT() Function

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

SQL COUNT(column_name) Syntax

The COUNT(column_name) function returns the number of values (NULL values will not be counted) of the specified column:

```
SELECT COUNT(column_name) FROM table_name;
```

SQL COUNT(*) Syntax

The COUNT(*) function returns the number of records in a table:

DBMS – Database Management System

SQL – Structured Query Language

```
SELECT COUNT(*) FROM table_name;
```

SQL COUNT(DISTINCT column_name) Syntax

The COUNT(DISTINCT column_name) function returns the number of distinct values of the specified column:

```
SELECT COUNT(DISTINCT column_name) FROM table_name;
```

Note: COUNT(DISTINCT) works with ORACLE and Microsoft SQL Server, but not with Microsoft Access.

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	Shippe
10265	7	2	1996-07-25	1
10266	87	3	1996-07-26	3
10267	25	4	1996-07-29	1

SQL COUNT(column_name) Example

The following SQL statement counts the number of orders from "CustomerID"=7 from the "Orders" table:

Example

```
SELECT COUNT(CustomerID) AS OrdersFromCustomerID7 FROM Orders  
WHERE CustomerID=7;
```

DBMS – Database Management System

SQL – Structured Query Language

SQL COUNT(*) Example

The following SQL statement counts the total number of orders in the "Orders" table:

Example

```
SELECT COUNT(*) AS NumberOfOrders FROM Orders;
```

SQL COUNT(DISTINCT column_name) Example

The following SQL statement counts the number of unique customers in the "Orders" table:

Example

```
SELECT COUNT(DISTINCT CustomerID) AS NumberOfCustomers FROM Orders;
```

Lesson 48. SQL FIRST() Function

The FIRST() Function

The FIRST() function returns the first value of the selected column.

SQL FIRST() Syntax

```
SELECT FIRST(column_name) FROM table_name;
```

Note: The FIRST() function is only supported in MS Access.

SQL FIRST() Workaround in SQL Server, MySQL and Oracle

SQL Server Syntax

```
SELECT TOP 1 column_name FROM table_name  
ORDER BY column_name ASC;
```

DBMS – Database Management System

SQL – Structured Query Language

Example

```
SELECT TOP 1 CustomerName FROM Customers  
ORDER BY CustomerID ASC;
```

MySQL Syntax

```
SELECT column_name FROM table_name  
ORDER BY column_name ASC  
LIMIT 1;
```

Example

```
SELECT CustomerName FROM Customers  
ORDER BY CustomerID ASC  
LIMIT 1;
```

Oracle Syntax

```
SELECT column_name FROM table_name  
ORDER BY column_name ASC  
WHERE ROWNUM <=1;
```

Example

```
SELECT CustomerName FROM Customers  
ORDER BY CustomerID ASC  
WHERE ROWNUM <=1;
```

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode
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

DBMS – Database Management System

SQL – Structured Query Language

4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DF
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22

SQL FIRST() Example

The following SQL statement selects the first value of the "CustomerName" column from the "Customers" table:

Example

```
SELECT FIRST(CustomerName) AS FirstCustomer FROM Customers;
```

Lesson 49. SQL LAST() Function

The LAST() Function

The LAST() function returns the last value of the selected column.

SQL LAST() Syntax

```
SELECT LAST(column_name) FROM table_name;
```

Note: The LAST() function is only supported in MS Access.

SQL LAST() Workaround in SQL Server, MySQL and Oracle

SQL Server Syntax

```
SELECT TOP 1 column_name FROM table_name  
ORDER BY column_name DESC;
```

Example

```
SELECT TOP 1 CustomerName FROM Customers  
ORDER BY CustomerID DESC;
```

DBMS – Database Management System

SQL – Structured Query Language

MySQL Syntax

```
SELECT column_name FROM table_name  
ORDER BY column_name DESC  
LIMIT 1;
```

Example

```
SELECT CustomerName FROM Customers  
ORDER BY CustomerID DESC  
LIMIT 1;
```

Oracle Syntax

```
SELECT column_name FROM table_name  
ORDER BY column_name DESC  
WHERE ROWNUM <=1;
```

Example

```
SELECT CustomerName FROM Customers  
ORDER BY CustomerID DESC  
WHERE ROWNUM <=1;
```

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode
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 1DP
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22

DBMS – Database Management System

SQL – Structured Query Language

SQL LAST() Example

The following SQL statement selects the last value of the "CustomerName" column from the "Customers" table:

Example

```
SELECT LAST(CustomerName) AS LastCustomer FROM Customers;
```

Lesson 50. SQL MAX() Function

The MAX() Function

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

SQL MAX() Syntax

```
SELECT MAX(column_name) FROM table_name;
```

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Products" table:

ProductID	ProductName	SupplierID	CategoryID	Unit
1	Chais	1	1	10 boxes x 20 bag
2	Chang	1	1	24 - 12 oz bottles

DBMS – Database Management System

SQL – Structured Query Language

3	Aniseed Syrup	1	2	12 - 550 ml bottle
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars
5	Chef Anton's Gumbo Mix	2	2	36 boxes

SQL MAX() Example

The following SQL statement gets the largest value of the "Price" column from the "Products" table:

Example

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

Lesson 51. SQL MIN() Function

The MIN() Function

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

SQL MIN() Syntax

```
SELECT MIN(column_name) FROM table_name;
```

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Products" table:

ProductID	ProductName	SupplierID	CategoryID	Unit
1	Chais	1	1	10 boxes x 20 bag

DBMS – Database Management System

SQL – Structured Query Language

2	Chang	1	1	24 - 12 oz bottles
3	Aniseed Syrup	1	2	12 - 550 ml bottle
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars
5	Chef Anton's Gumbo Mix	2	2	36 boxes

SQL MIN() Example

The following SQL statement gets the smallest value of the "Price" column from the "Products" table:

Example

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

Lesson 52. SQL SUM() Function

The SUM() Function

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

SQL SUM() Syntax

```
SELECT SUM(column_name) FROM table_name;
```

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "OrderDetails" table:

OrderDetailID	OrderID	ProductID	Quantity
---------------	---------	-----------	----------

DBMS – Database Management System

SQL – Structured Query Language

1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40

SQL SUM() Example

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

Example

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

Lesson 53. SQL GROUP BY Statement

Aggregate functions often need an added GROUP BY statement.

The GROUP BY Statement

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

SQL GROUP BY Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
```

DBMS – Database Management System

SQL – Structured Query Language

```
WHERE column_name operator value  
GROUP BY column_name;
```

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Orders" table:

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	Phone
1	Speedy Express	(503) 555-9831
2	United Package	(503) 555-3199
3	Federal Shipping	(503) 555-9931

And a selection from the "Employees" table:

EmployeeID	LastName	FirstName	BirthDate	Photo	Notes
1	Davolio	Nancy	1968-12-08	EmpID1.pic	Education includes a
2	Fuller	Andrew	1952-02-19	EmpID2.pic	Andrew received his

DBMS – Database Management System

SQL – Structured Query Language

3	Leverling	Janet	1963-08-30	EmpID3.pic	Janet has a BS degree
---	-----------	-------	------------	------------	-----------------------

SQL GROUP BY Example

Now we want to find the number of orders sent by each shipper.

The following SQL statement counts as orders grouped by shippers:

Example

```
SELECT Shippers.ShipperName,COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders
LEFT JOIN Shippers
ON Orders.ShipperID=Shippers.ShipperID
GROUP BY ShipperName;
```

GROUP BY More Than One Column

We can also use the GROUP BY statement on more than one column, like this:

Example

```
SELECT Shippers.ShipperName, Employees.LastName,
COUNT(Orders.OrderID) AS NumberOfOrders
FROM ((Orders
INNER JOIN Shippers
ON Orders.ShipperID=Shippers.ShipperID)
INNER JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID)
GROUP BY ShipperName,LastName;
```

Lesson 54. SQL HAVING Clause

The HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

DBMS – Database Management System

SQL – Structured Query Language

SQL HAVING Syntax

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value;
```

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Orders" table:

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
2	Fuller	Andrew	1952-02-19	EmpID2.pic	Andrew received his
3	Leverling	Janet	1963-08-30	EmpID3.pic	Janet has a BS degree

SQL HAVING Example

Now we want to find if any of the customers have a total order of less than 2000.

DBMS – Database Management System

SQL – Structured Query Language

We use the following SQL statement:

The following SQL statement finds if any of the employees has 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;
```

Now we want to find if the employees "Davolio" or "Fuller" have more than 25 orders.

We add an ordinary WHERE clause to the SQL statement:

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;
```

Lesson 55. SQL UCASE() Function

The UCASE() Function

The UCASE() function converts the value of a field to uppercase.

DBMS – Database Management System

SQL – Structured Query Language

SQL UCASE() Syntax

```
SELECT UCASE(column_name) FROM table_name;
```

Syntax for SQL Server

```
SELECT UPPER(column_name) FROM table_name;
```

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode
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 1DP
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22

SQL UCASE() Example

The following SQL statement selects the "CustomerName" and "City" columns from the "Customers" table, and converts the "CustomerName" column to uppercase:

Example

```
SELECT UCASE(CustomerName) AS Customer, City  
FROM Customers;
```

DBMS – Database Management System

SQL – Structured Query Language

Lesson 56. SQL **LCASE()** Function

The **LCASE()** Function

The LCASE() function converts the value of a field to lowercase.

SQL **LCASE()** Syntax

```
SELECT LCASE(column_name) FROM table_name;
```

Syntax for SQL Server

```
SELECT LOWER(column_name) FROM table_name;
```

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode
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 1DP
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22

DBMS – Database Management System

SQL – Structured Query Language

SQL LCASE() Example

The following SQL statement selects the "CustomerName" and "City" columns from the "Customers" table, and converts the "CustomerName" column to lowercase:

Example

```
SELECT LCASE(CustomerName) AS Customer, City  
FROM Customers;
```

Lesson 57. SQL MID() Function

The MID() Function

The MID() function is used to extract characters from a text field.

SQL MID() Syntax

```
SELECT MID(column_name,start[,length]) AS some_name FROM table_name;
```

Parameter	Description
column_name	Required. The field to extract characters from
Start	Required. Specifies the starting position (starts at 1)

DBMS – Database Management System

SQL – Structured Query Language

Length	Optional. The number of characters to return. If omitted, the MID() function returns all characters of the text
--------	---

Note: The equivalent function for SQL Server is SUBSTRING():

```
SELECT SUBSTRING(column_name,start,length) AS some_name FROM table_name;
```

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode
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 1DP
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22

SQL MID() Example

The following SQL statement selects the first four characters from the "City" column from the "Customers" table:

DBMS – Database Management System

SQL – Structured Query Language

Example

```
SELECT MID(City,1,4) AS ShortCity  
FROM Customers;
```

Lesson 58. SQL **LEN()** Function

The **LEN()** Function

The LEN() function returns the length of the value in a text field.

SQL **LEN()** Syntax

```
SELECT LEN(column_name) FROM table_name;
```

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode
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 1DP
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22

DBMS – Database Management System

SQL – Structured Query Language

SQL LEN() Example

The following SQL statement selects the "CustomerName" and the length of the values in the "Address" column from the "Customers" table:

Example

```
SELECT CustomerName,LEN(Address) as LengthOfAddress  
FROM Customers;
```

Lesson 59. SQL ROUND() Function

The ROUND() Function

The ROUND() function is used to round a numeric field to the number of decimals specified.

Note: Some database systems do rounding differently than would typically be considered. Most people assume that the ROUND() function would round to the nearest whole number. However, many DBMS's do "Bankers Rounding". This means that the number being rounded is rounded to the nearest EVEN whole number. I.E. if the number being rounded was 11.3, the logical rounding to most people would be to 11. However, since 11 is odd, "Bankers Rounding" would round this number to 12 instead.

SQL ROUND() Syntax

```
SELECT ROUND(column_name,decimals) FROM table_name;
```

DBMS – Database Management System

SQL – Structured Query Language

Parameter	Description
column_name	Required. The field to round.
Decimals	Required. Specifies the number of decimals to be returned.

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Products" table:

ProductID	ProductName	SupplierID	CategoryID	Unit
1	Chais	1	1	10 boxes x 20 bag
2	Chang	1	1	24 - 12 oz bottles
3	Aniseed Syrup	1	2	12 - 550 ml bottle
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars
5	Chef Anton's Gumbo Mix	2	2	36 boxes

SQL ROUND() Example

The following SQL statement selects the product name and rounds the price in the "Products" table:

Example

```
SELECT ProductName, ROUND(Price,0) AS RoundedPrice  
FROM Products;
```

DBMS – Database Management System

SQL – Structured Query Language

Lesson 60. SQL **NOW()** Function

The **NOW()** Function

The NOW() function returns the current system date and time.

SQL **NOW()** Syntax

```
SELECT NOW() FROM table_name;
```

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Products" table:

ProductID	ProductName	SupplierID	CategoryID	Unit
1	Chais	1	1	10 boxes x 20 bag
2	Chang	1	1	24 - 12 oz bottles
3	Aniseed Syrup	1	2	12 - 550 ml bottle
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars
5	Chef Anton's Gumbo Mix	2	2	36 boxes

SQL **NOW()** Example

The following SQL statement selects the product name, and price for today from the "Products" table:

Example

```
SELECT ProductName, Price, Now() AS PerDate  
FROM Products;
```

DBMS – Database Management System

SQL – Structured Query Language

Lesson 61. SQL **FORMAT()** Function

The **FORMAT()** Function

The **FORMAT()** function is used to format how a field is to be displayed.

SQL **FORMAT() Syntax**

```
SELECT FORMAT(column_name,format) FROM table_name;
```

Parameter	Description
column_name	Required. The field to be formatted.
Format	Required. Specifies the format.

Demo Database

In this tutorial we will use the well-known SDS_14 sample database.

Below is a selection from the "Products" table:

DBMS – Database Management System

SQL – Structured Query Language

ProductID	ProductName	SupplierID	CategoryID	Unit
1	Chais	1	1	10 boxes x 20 bag
2	Chang	1	1	24 - 12 oz bottles
3	Aniseed Syrup	1	2	12 - 550 ml bottle
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars
5	Chef Anton's Gumbo Mix	2	2	36 boxes

SQL FORMAT() Example

The following SQL statement selects the product name, and price for today (formatted like YYYY-MM-DD) from the "Products" table:

Example

```
SELECT ProductName, Price, FORMAT(Now(), 'YYYY-MM-DD') AS PerDate  
FROM Products;
```

Lesson 62. SQL Injection

An SQL Injection can destroy your database.

SQL in Web Pages

In the previous chapters, you have learned to retrieve (and update) database data, using SQL.

When SQL is used to display data on a web page, it is common to let web users input their own search values.

Since SQL statements are text only, it is easy, with a little piece of computer code, to dynamically change SQL statements to provide the user with selected data:

DBMS – Database Management System

SQL – Structured Query Language

Server Code

```
txtUserId = getRequestString("UserId");  
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

The example above, creates a select statement by adding a variable (txtUserId) to a select string. The variable is fetched from the user input (Request) to the page.

The rest of this chapter describes the potential dangers of using user input in SQL statements.

SQL Injection

SQL injection is a technique where malicious users can inject SQL commands into an SQL statement, via web page input.

Injected SQL commands can alter SQL statement and compromise the security of a web application.

SQL Injection Based on 1=1 is Always True

Look at the example above, one more time.

Let's say that the original purpose of the code was to create an SQL statement to select a user with a given user id.

If there is nothing to prevent a user from entering "wrong" input, the user can enter some "smart" input like this:

UserId:

Server Result

```
SELECT * FROM Users WHERE UserId = 105 or 1=1
```

The SQL above is valid. It will return all rows from the table Users, since **WHERE 1=1** is always true.

Does the example above seem dangerous? What if the Users table contains names and passwords?

The SQL statement above is much the same as this:

DBMS – Database Management System

SQL – Structured Query Language

```
SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1
```

A smart hacker might get access to all the user names and passwords in a database by simply inserting 105 or 1=1 into the input box.

SQL Injection Based on ""="" is Always True

Here is a common construction, used to verify user login to a web site:

User Name:

Password:

Server Code

```
uName = getRequestString("UserName");  
uPass = getRequestString("UserPass");
```

```
sql = "SELECT * FROM Users WHERE Name ='" + uName + "' AND Pass ='" + uPass + "'"
```

A smart hacker might get access to user names and passwords in a database by simply inserting " or ""="" into the user name or password text box.

The code at the server will create a valid SQL statement like this:

Result

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

The result SQL is valid. It will return all rows from the table Users, since **WHERE ""=""** is always true.

SQL Injection Based on Batched SQL Statements

Most databases support batched SQL statement, separated by semicolon.

Example

```
SELECT * FROM Users; DROP TABLE Suppliers
```

DBMS – Database Management System

SQL – Structured Query Language

The SQL above will return all rows in the Users table, and then delete the table called Suppliers.

If we had the following server code:

Server Code

```
txtUserId = getRequestString("UserId");  
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

And the following input:

User id:

The code at the server would create a valid SQL statement like this:

Result

```
SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers
```

Parameters for Protection

Some web developers use a "blacklist" of words or characters to search for in SQL input, to prevent SQL injection attacks.

This is not a very good idea. Many of these words (like delete or drop) and characters (like semicolons and quotation marks), are used in common language, and should be allowed in many types of input.

(In fact it should be perfectly legal to input an SQL statement in a database field.)

The only proven way to protect a web site from SQL injection attacks, is to use SQL parameters.

SQL parameters are values that are added to an SQL query at execution time, in a controlled manner.

ASP.NET Razor Example

```
txtUserId = getRequestString("UserId");  
txtSQL = "SELECT * FROM Users WHERE UserId = @0";  
db.Execute(txtSQL,txtUserId);
```

DBMS – Database Management System

SQL – Structured Query Language

Note that parameters are represented in the SQL statement by a @ marker.

The SQL engine checks each parameter to ensure that it is correct for its column and are treated literally, and not as part of the SQL to be executed.

Another Example

```
txtNam = getRequestString("CustomerName");
txtAdd = getRequestString("Address");
txtCit = getRequestString("City");
txtSQL = "INSERT INTO Customers (CustomerName,Address,City) Values(@0,@1,@2)";
db.Execute(txtSQL,txtNam,txtAdd,txtCit);
```



You have just learned to avoid SQL injection. One of the top website vulnerabilities.

Examples

The following examples shows how to build parameterized queries in some common web languages.

ASP.NET SELECT

```
txtUserId = getRequestString("UserId");
sql = "SELECT * FROM Customers WHERE CustomerId = @0";
command = new SqlCommand(sql);
command.Parameters.AddWithValue("@0",txtUserID);
command.ExecuteReader();
```

ASP.NET INSERT INTO

```
txtNam = getRequestString("CustomerName");
txtAdd = getRequestString("Address");
txtCit = getRequestString("City");
txtSQL = "INSERT INTO Customers (CustomerName,Address,City) Values(@0,@1,@2)";
command = new SqlCommand(txtSQL);
command.Parameters.AddWithValue("@0",txtNam);
command.Parameters.AddWithValue("@1",txtAdd);
command.Parameters.AddWithValue("@2",txtCit);
command.ExecuteNonQuery();
```

DBMS – Database Management System

SQL – Structured Query Language

PHP INSERT INTO

```
$stmt = $dbh->prepare("INSERT INTO Customers (CustomerName,Address,City)
VALUES (:nam, :add, :cit)");
$stmt->bindParam(':nam', $txtNam);
$stmt->bindParam(':add', $txtAdd);
$stmt->bindParam(':cit', $txtCit);
$stmt->execute();
```

Lesson 63. Summary – Try

END