SQL SEVER

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 became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

SQL is a Standard - BUT....

Although SQL is an ANSI/ISO standard, there are different versions of the SQL language.

However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.

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

Using SQL in Your Web Site

To build a web site that shows data from a database, you will need:

- An RDBMS database program (i.e. MS Access, SQL Server, MySQL)
- To use a server-side scripting language, like PHP or ASP
- To use SQL to get the data you want

- To use HTML / CSS to style the page

RDBMS

- RDBMS stands for Relational Database Management System.
- RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.
- Look at the "Customers" table:

Example:

SELECT * FROM Customers;

Result:

Number of Records: 10

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
8	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid	28023	Spain
9	Bon app'	Laurence Lebihans	12, rue des Bouchers	Marseille	13008	France
10	Bottom-Dollar Marketse	Elizabeth Lincoln	23 Tsawassen Blvd.	Tsawassen	T2F 8M4	Canada

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

- A record, also called a row, is each individual entry that exists in a table. For example, there are 91 records in the above Customers table. A record is a horizontal entity in a table.
- A column is a vertical entity in a table that contains all information associated with a specific field in a table.

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 Northwind sample database (included in MS Access and MS SQL Server).
- Below is a selection from the "Customers" table:

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

- The table above contains five records (one for each customer) and seven columns (CustomerID, CustomerName, ContactName, Address, City, PostalCode, and Country).

SQL Statements

Most of the actions you need to perform on a database are done with SQL statements.
The following SQL statement selects all the records in the "Customers" table:
Example:

```
SELECT * FROM Customers;
```

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

Keep in Mind That...

- SQL keywords are NOT case sensitive: select is the same as SELECT

In this tutorial we will write all SQL keywords in upper-case.

Semicolon after SQL Statements?

- Some database systems require a semicolon at the end of each SQL statement.
- Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.
- In this tutorial, we will use semicolon at the end of each SQL statement.

Some of the Most Important SQL Commands

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

SQL SELECT STATEMENT

The SQL SELECT Statement

The SELECT statement is used to select data from a database.
The data returned is stored in a result table, called the result-set.

SELECT Syntax:

SELECT *column1*, *column2, ...*
FROM *table_name*;

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

SELECT * FROM *table_name*;

Demo Database

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

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

SELECT Column Example

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

Example:

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

Result:

Number of Records: 9

CustomerName	City
Alfreds Futterkiste	Berlin
Ana Trujillo Emparedados y helados	México D.F.
Antonio Moreno Taquería	México D.F.
Around the Horn	London
Berglunds snabbköp	Luleå
Blauer See Delikatessen	Mannheim
Blondel père et fils	Strasbourg
Bólido Comidas preparadas	Madrid
Bon app'	Marseille

SQL SELECT DISTINCT Statement

The SQL SELECT DISTINCT Statement

- The SELECT DISTINCT statement is used to return only distinct (different) values.
- Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT Syntax:
SELECT DISTINCT *column1*, *column2, ...*
FROM *table_name*;

Demo Database

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

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

SELECT Example Without DISTINCT

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

Example:
SELECT Country FROM Customers;
Result:

Country
Germany
Mexico
Mexico
UK

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

SELECT DISTINCT Examples

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

Example:

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

Result:

Country
Germany
Mexico
UK
Sweden

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

Example:

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

Note: The example above will not work in Firefox! Because COUNT(DISTINCT *column_name*) is not supported in Microsoft Access databases. Firefox is using Microsoft Access in our examples.

Here is the workaround for MS Access:

Example:

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


SQL WHERE CLAUSE

The SQL WHERE Clause

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

WHERE Syntax:

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

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

Demo Database

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

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

WHERE Clause Example

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

Example:

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

Result:

Number of Records: 5

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
13	Centro comercial Moctezuma	Francisco Chang	Sierras de Granada 9993	México D.F.	05022	Mexico
58	Pericles Comidas clásicas	Guillermo Fernández	Calle Dr. Jorge Cash 321	México D.F.	05033	Mexico
80	Tortuga Restaurante	Miguel Angel Paolino	Avda. Azteca 123	México D.F.	05033	Mexico

Text Fields vs. Numeric Fields

SQL requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes:

Example:

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

Operators in The WHERE Clause

The following operators can be used in the WHERE clause:

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

SQL AND, OR and NOT Operators

The SQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax:
SELECT *column1*, *column2, ...*
FROM *table_name*
WHERE *condition1* OR *condition2* OR *condition3 ...*;

NOT Syntax:
SELECT *column1*, *column2, ...*
FROM *table_name*
WHERE NOT *condition*;

Demo Database

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

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France

8	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid	28023	Spain
9	Bon app'	Laurence Lebihans	12, rue des Bouchers	Marseille	13008	France
10	Bottom-Dollar Marketse	Elizabeth Lincoln	23 Tsawassen Blvd.	Tsawassen	T2F 8M4	Canada
11	B's Beverages	Victoria Ashworth	Fauntleroy Circus	London	EC2 5NT	UK
12	Cactus Comidas para llevar	Patricio Simpson	Cerrito 333	Buenos Aires	1010	Argentina
13	Centro comercial Moctezuma	Francisco Chang	Sierras de Granada 9993	México D.F.	05022	Mexico
14	Chop-suey Chinese	Yang Wang	Hauptstr. 29	Bern	3012	Switzerland
15	Comércio Mineiro	Pedro Afonso	Av. dos Lusíadas, 23	São Paulo	05432-043	Brazil
16	Consolidated Holdings	Elizabeth Brown	Berkeley Gardens 12 Brewery	London	WX1 6LT	UK
17	Drachenblut Delikatessend	Sven Ottlieb	Walserweg 21	Aachen	52066	Germany
18	Du monde entier	Janine Labrune	67, rue des Cinquante Otages	Nantes	44000	France

19	Eastern Connection	Ann Devon	35 King George	London	WX3 6FW	UK
20	Ernst Handel	Roland Mendel	Kirchgasse 6	Graz	8010	Austria
21	Familia Arquibaldo	Aria Cruz	Rua Orós, 92	São Paulo	05442-030	Brazil
22	FISSA Fabrica Inter. Salchichas S.A.	Diego Roel	C/ Moralzarzal, 86	Madrid	28034	Spain
23	Folies gourmandes	Martine Rancé	184, chaussée de Tournai	Lille	59000	France
24	Folk och fä HB	Maria Larsson	Åkergatan 24	Bräcke	S-844 67	Sweden
25	Frankenversand	Peter Franken	Berliner Platz 43	München	80805	Germany
26	France restauration	Carine Schmitt	54, rue Royale	Nantes	44000	France
27	Franchi S.p.A.	Paolo Accorti	Via Monte Bianco 34	Torino	10100	Italy
28	Furia Bacalhau e Frutos do Mar	Lino Rodriguez	Jardim das rosas n. 32	Lisboa	1675	Portugal
29	Galería del gastrónomo	Eduardo Saavedra	Rambla de Cataluña, 23	Barcelona	08022	Spain
30	Godos Cocina Típica	José Pedro Freyre	C/ Romero, 33	Sevilla	41101	Spain

31	Gourmet Lanchonetes	André Fonseca	Av. Brasil, 442	Campinas	04876-786	Brazil
32	Great Lakes Food Market	Howard Snyder	2732 Baker Blvd.	Eugene	97403	USA
33	GROSELLA-Restaurante	Manuel Pereira	5ª Ave. Los Palos Grandes	Caracas	1081	Venezuela
34	Hanari Carnes	Mario Pontes	Rua do Paço, 67	Rio de Janeiro	05454-876	Brazil
35	HILARIÓN-Abastos	Carlos Hernández	Carrera 22 con Ave. Carlos Soublette #8-35	San Cristóbal	5022	Venezuela
36	Hungry Coyote Import Store	Yoshi Latimer	City Center Plaza 516 Main St.	Elgin	97827	USA
37	Hungry Owl All-Night Grocers	Patricia McKenna	8 Johnstown Road	Cork		Ireland
38	Island Trading	Helen Bennett	Garden House Crowther Way	Cowes	PO31 7PJ	UK
39	Königlich Essen	Philip Cramer	Maubelstr. 90	Brandenburg	14776	Germany
40	La corne d'abondance	Daniel Tonini	67, avenue de l'Europe	Versailles	78000	France
41	La maison d'Asie	Annette Roulet	1 rue Alsace-Lorraine	Toulouse	31000	France

42	Laughing Bacchus Wine Cellars	Yoshi Tannamuri	1900 Oak St.	Vancouver	V3F 2K1	Canada
43	Lazy K Kountry Store	John Steel	12 Orchestra Terrace	Walla Walla	99362	USA
44	Lehmanns Marktstand	Renate Messner	Magazinweg 7	Frankfurt a.M.	60528	Germany
45	Let's Stop N Shop	Jaime Yorres	87 Polk St. Suite 5	San Francisco	94117	USA
46	LILA-Supermercado	Carlos González	Carrera 52 con Ave. Bolívar #65-98 Llano Largo	Barquisimeto	3508	Venezuela
47	LINO-Delicatesses	Felipe Izquierdo	Ave. 5 de Mayo Porlamar	I. de Margarita	4980	Venezuela
48	Lonesome Pine Restaurant	Fran Wilson	89 Chiaroscuro Rd.	Portland	97219	USA
49	Magazzini Alimentari Riuniti	Giovanni Rovelli	Via Ludovico il Moro 22	Bergamo	24100	Italy
50	Maison Dewey	Catherine Dewey	Rue Joseph-Bens 532	Bruxelles	B-1180	Belgium
51	Mère Paillarde	Jean Fresnière	43 rue St. Laurent	Montréal	H1J 1C3	Canada
52	Morgenstern	Alexander Feuer	Heerstr. 22	Leipzig	04179	Germany

	Gesundkost					
53	North/South	Simon Crowther	South House 300 Queensbridge	London	SW7 1RZ	UK
54	Océano Atlántico Ltda.	Yvonne Moncada	Ing. Gustavo Moncada 8585 Piso 20-A	Buenos Aires	1010	Argentina
55	Old World Delicatessen	Rene Phillips	2743 Bering St.	Anchorage	99508	USA
56	Ottilies Käseladen	Henriette Pfalzheim	Mehrheimerstr. 369	Köln	50739	Germany
57	Paris spécialités	Marie Bertrand	265, boulevard Charonne	Paris	75012	France
58	Pericles Comidas clásicas	Guillermo Fernández	Calle Dr. Jorge Cash 321	México D.F.	05033	Mexico
59	Piccolo und mehr	Georg Pipps	Geislweg 14	Salzburg	5020	Austria
60	Princesa Isabel Vinhoss	Isabel de Castro	Estrada da saúde n. 58	Lisboa	1756	Portugal
61	Que Delícia	Bernardo Batista	Rua da Panificadora, 12	Rio de Janeiro	02389-673	Brazil
62	Queen Cozinha	Lúcia Carvalho	Alameda dos	São Paulo	05487-020	Brazil

			Canàrios, 891			
63	QUICK-Stop	Horst Kloss	Tauchers traße 10	Cunewalde	01307	Germany
64	Rancho grande	Sergio Gutiérrez	Av. del Libertador 900	Buenos Aires	1010	Argentina
65	Rattlesnake Canyon Grocery	Paula Wilson	2817 Milton Dr.	Albuquerque	87110	USA
66	Reggiani Caseifici	Maurizio Moroni	Strada Provinciale 124	Reggio Emilia	42100	Italy
67	Ricardo Adocicados	Janete Limeira	Av. Copacabana, 267	Rio de Janeiro	02389-890	Brazil
68	Richter Supermarkt	Michael Holz	Grenzacherweg 237	Genève	1203	Switzerland
69	Romero y tomillo	Alejandra Camino	Gran Vía, 1	Madrid	28001	Spain
70	Santé Gourmet	Jonas Bergulfsen	Erling Skakkes gate 78	Stavern	4110	Norway
71	Save-a-lot Markets	Jose Pavarotti	187 Suffolk Ln.	Boise	83720	USA
72	Seven Seas Imports	Hari Kumar	90 Wadhurst Rd.	London	OX15 4NB	UK
73	Simons bistro	Jytte Petersen	Vinbæltet 34	København	1734	Denmark
74	Spécialités du monde	Dominique Perrier	25, rue Lauriston	Paris	75016	France
75	Split Rail Beer & Ale	Art Braunschweiger	P.O. Box 555	Lander	82520	USA

76	Suprêmes délices	Pascale Cartrain	Boulevard Tirou, 255	Charleroi	B-6000	Belgium
77	The Big Cheese	Liz Nixon	89 Jefferson Way Suite 2	Portland	97201	USA
78	The Cracker Box	Liu Wong	55 Grizzly Peak Rd.	Butte	59801	USA
79	Toms Spezialitäten	Karin Josephs	Luisenstr. 48	Münster	44087	Germany
80	Tortuga Restaurante	Miguel Angel Paolino	Avda. Azteca 123	México D.F.	05033	Mexico
81	Tradição Hipermercados	Anabela Domingues	Av. Inês de Castro, 414	São Paulo	05634-030	Brazil
82	Trail's Head Gourmet Provisioners	Helvetius Nagy	722 DaVinci Blvd.	Kirkland	98034	USA
83	Vaffeljernet	Palle Ibsen	Smagsløget 45	Århus	8200	Denmark
84	Victuailles en stock	Mary Saveley	2, rue du Commerce	Lyon	69004	France
85	Vins et alcools Chevalier	Paul Henriot	59 rue de l'Abbaye	Reims	51100	France
86	Die Wandernde Kuh	Rita Müller	Adenauerallee 900	Stuttgart	70563	Germany
87	Wartian Herkku	Pirkko Koskitalo	Torikatu 38	Oulu	90110	Finland

88	Wellington Importadora	Paula Parente	Rua do Mercado, 12	Resende	08737-363	Brazil
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland

AND Example

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

Example:

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

OR Example

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

Example:

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

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

Example:

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

NOT Example

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

Example:

```
SELECT * FROM Customers
```

```
WHERE NOT Country='Germany';
```

Combining AND, OR and NOT

- You can also combine the AND, OR and NOT operators.
- The following SQL statement selects all fields from "Customers" where country is "Germany" AND city must be "Berlin" OR "München" (use parenthesis to form complex expressions):

Example

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

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

Example

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

SQL ORDER BY Keyword

The SQL ORDER BY Keyword

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Demo Database

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

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany

2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

ORDER BY Example

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

Example:

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

ORDER BY DESC Example

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

Example:

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

ORDER BY Several Columns Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

Example:

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

ORDER BY Several Columns Example 2

The following SQL statement selects all customers from the "Customers" table, sorted ascending by the "Country" and descending by the "CustomerName"

column:
Example:

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

SQL INSERT INTO Statement

The SQL INSERT INTO Statement

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

INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

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

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

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


Demo Database

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

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland

INSERT INTO Example

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

Example:

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

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

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland
92	Cardinal	Tom B. Erichsen	Skagen 21	Stavanger	4006	Norway

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

Insert Data Only in Specified Columns

- It is also possible to only insert data in specific columns.
- The following SQL statement will insert a new record, but only insert data in the "CustomerName", "City", and "Country" columns (CustomerID will be updated automatically):

Example:

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

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

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland

91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland
92	Cardinal	null	null	Stavanger	null	Norway

SQL NULL Values

What is a NULL Value?

- A field with a NULL value is a field with no value.
- If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

Note: A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

IS NULL Syntax:

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

IS NOT NULL Syntax:

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

Demo Database

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

Customer I D	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany

2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

The IS NULL Operator

The IS NULL operator is used to test for empty values (NULL values).

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

Example:

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

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

The IS NOT NULL Operator

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

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

Example:

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

SQL UPDATE Statement

The SQL UPDATE Statement

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

UPDATE Syntax:

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

Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

Demo Database

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

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

UPDATE Table

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person *and* a new city.

Example:

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

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

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

UPDATE Multiple Records

It is the WHERE clause that determines how many records will be updated.

The following SQL statement will update the ContactName to "Juan" for all records where country is "Mexico":

Example

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```


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

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	12209	Germany
2	Ana Trujillo Emparedados y helados	Juan	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Juan	Mataderos 2312	México D.F.	05023	Mexico

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

Update Warning!

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

Example:

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

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

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Juan	Obere Str. 57	Frankfurt	12209	Germany
2	Ana Trujillo Emparedados y helados	Juan	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Juan	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Juan	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Juan	Berguvsvägen 8	Luleå	S-958 22	Sweden

SQL DELETE Statement

The SQL DELETE Statement

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

DELETE Syntax:

```
DELETE FROM table_name WHERE condition;
```

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be

deleted. If you omit the WHERE clause, all records in the table will be deleted!

Demo Database

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

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

SQL DELETE Example

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

Example:

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

The "Customers" table will now look like this:

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Delete All Records

It is possible to delete all rows in a table without deleting the table. This means

that the table structure, attributes, and indexes will be intact:

DELETE FROM *table_name*;

The following SQL statement deletes all rows in the "Customers" table, without deleting the table:

Example:
DELETE FROM Customers;

SQL TOP, LIMIT, FETCH FIRST or ROWNUM Clause

The SQL SELECT TOP Clause

- The SELECT TOP clause is used to specify the number of records to return.
- The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

Note: Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses FETCH FIRST *n* ROWS ONLY and ROWNUM.

SQL Server / MS Access Syntax:
SELECT TOP *number*|*percent column_name(s)*
FROM *table_name*
WHERE *condition*;

MySQL Syntax:
SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*
LIMIT *number*;

Oracle 12 Syntax:
SELECT *column_name(s)*
FROM *table_name*
ORDER BY *column_name(s)*
FETCH FIRST *number* ROWS ONLY;

Older Oracle Syntax:

SELECT *column_name(s)*
FROM *table_name*
WHERE ROWNUM <= *number*;

Older Oracle Syntax (with ORDER BY):

SELECT *
FROM (SELECT *column_name(s)* FROM *table_name* ORDER BY *column_name(s)*
)
WHERE ROWNUM <= *number*;

Demo Database

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

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

SQL TOP, LIMIT and FETCH FIRST Examples

The following SQL statement selects the first three records from the "Customers" table (for SQL Server/MS Access):

Example:

SELECT TOP 3 * FROM Customers;

The following SQL statement shows the equivalent example for MySQL:

Example:

SELECT * FROM Customers
LIMIT 3;

The following SQL statement shows the equivalent example for Oracle:

Example:
```
SELECT * FROM Customers
FETCH FIRST 3 ROWS ONLY;
```

SQL TOP PERCENT Example

The following SQL statement selects the first 50% of the records from the "Customers" table (for SQL Server/MS Access):

Example:
```
SELECT TOP 50 PERCENT * FROM Customers;
```

The following SQL statement shows the equivalent example for Oracle:

Example:
```
SELECT * FROM Customers
FETCH FIRST 50 PERCENT ROWS ONLY;
```

ADD a WHERE CLAUSE

The following SQL statement selects the first three records from the "Customers" table, where the country is "Germany" (for SQL Server/MS Access):

Example:
```
SELECT TOP 3 * FROM Customers
WHERE Country='Germany';
```


The following SQL statement shows the equivalent example for MySQL:

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

The following SQL statement shows the equivalent example for Oracle:

Example:
```
SELECT * FROM Customers
WHERE Country='Germany'
FETCH FIRST 3 ROWS ONLY;
```

SQL MIN() and MAX() Functions

The SQL MIN() and MAX() Functions

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

MIN() Syntax:

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

MAX() Syntax:

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

Demo Database

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

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

MIN() Example

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

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

MAX() Example

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

Example:
SELECT MAX(Price) AS LargestPrice
FROM Products;

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

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

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

COUNT() Syntax:
SELECT COUNT(*column_name*)
FROM *table_name*
WHERE *condition*;

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

AVG() Syntax:
SELECT AVG(*column_name*)
FROM *table_name*
WHERE *condition*;

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

SUM() Syntax:
SELECT SUM(*column_name*)
FROM *table_name*
WHERE *condition*;

Demo Database

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

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19

3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

COUNT() Example

The following SQL statement finds the number of products:

Example:

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

Note: NULL values are not counted.

AVG() Example

The following SQL statement finds the average price of all products:

Example:

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

Note: NULL values are ignored.

Demo Database

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

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

SUM() Example

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

Example:

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

Note: NULL values are ignored.

SQL BETWEEN Operator

The SQL BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
The BETWEEN operator is inclusive: begin and end values are included.

BETWEEN Syntax:

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

Demo Database

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

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	1	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	1	2	36 boxes	21.35

BETWEEN Example

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

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

NOT BETWEEN Example

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

```
SELECT * FROM Products
```

```
WHERE Price NOT BETWEEN 10 AND 20;
```

BETWEEN with IN Example

The following SQL statement selects all products with a price between 10 and 20. In addition; do not show products with a CategoryID of 1,2, or 3:

Example:

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

BETWEEN Text Values Example

The following SQL statement selects all products with a ProductName between Carnarvon Tigers and Mozzarella di Giovanni:

Example:

```
SELECT * FROM Products
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

The following SQL statement selects all products with a ProductName between Carnarvon Tigers and Chef Anton's Cajun Seasoning:

Example:

```
SELECT * FROM Products
WHERE ProductName BETWEEN "Carnarvon Tigers" AND "Chef Anton's Cajun Seasoning"
ORDER BY ProductName;
```


NOT BETWEEN Text Values Example

The following SQL statement selects all products with a ProductName not between Carnarvon Tigers and Mozzarella di Giovanni:

Example:

```
SELECT * FROM Products
WHERE ProductName NOT BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

Sample Table

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

database:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	7/4/1996	3
10249	81	6	7/5/1996	1
10250	34	4	7/8/1996	2
10251	84	3	7/9/1996	1
10252	76	4	7/10/1996	2

BETWEEN Dates Example

The following SQL statement selects all orders with an OrderDate between '01-July-1996' and '31-July-1996':

Example:

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;
```

OR:

Example:

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

SQL JOINS

SQL JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

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

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Then, look at a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

- Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.
- Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

Example:

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

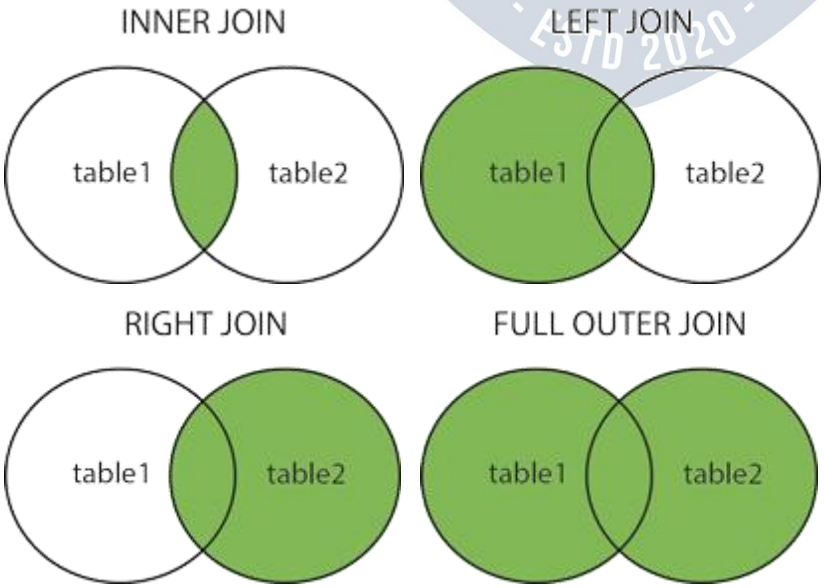
and it will produce something like this:

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- (INNER) JOIN: Returns records that have matching values in both tables
- LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
- RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
- FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table



SQL UNION Operator

The SQL UNION Operator

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

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

UNION Syntax:

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

UNION ALL Syntax:

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

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

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

Demo Database

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

Below is a selection from the "Customers" table:

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

And a selection from the "Suppliers" table:

Supplier ID	Supplier Name	Contact Name	Address	City	PostalCode	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	London	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 UNION Example

The following SQL statement returns the cities (only distinct values) from both the "Customers" and the "Suppliers" table:

Example:

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

Note: If some customers or suppliers have the same city, each city will only be listed once, because UNION selects only distinct values. Use UNION ALL to also select duplicate values!

SQL UNION ALL Example

The following SQL statement returns the cities (duplicate values also) from both the "Customers" and the "Suppliers" table:

Example:

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

SQL UNION With WHERE

The following SQL statement returns the German cities (only distinct values) from both the "Customers" and the "Suppliers" table:

Example:

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

SQL UNION ALL With WHERE

The following SQL statement returns the German cities (duplicate values also) from both the "Customers" and the "Suppliers" table:

Example:

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION ALL
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

Another UNION Example

The following SQL statement lists all customers and suppliers:

Example:

```
SELECT 'Customer' AS Type, ContactName, City, Country
FROM Customers
UNION
SELECT 'Supplier', ContactName, City, Country
FROM Suppliers;
```


SQL EXISTS Operator

The SQL EXISTS Operator

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

EXISTS Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

Demo Database:
Below is a selection from the "Products" table in the Northwind sample database:

Product ID	Product Name	Supplier ID	Category ID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

And a selection from the "Suppliers" table:

Supplier ID	Supplier Name	Contact Name	Address	City	Postal Code	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	London	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
4	Tokyo Traders	Yoshi Nagase	9-8 Sekimai Musashino-shi	Tokyo	100	Japan

SQL EXISTS Examples

The following SQL statement returns TRUE and lists the suppliers with a product price less than 20:
Example:

```
SELECT SupplierName
FROM Suppliers
```

WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price < 20);

The following SQL statement returns TRUE and lists the suppliers with a product price equal to 22:

Example:
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price = 22);

SQL ANY and ALL Operators

The SQL ANY and ALL Operators

The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

The SQL ANY Operator

The ANY operator:

- returns a boolean value as a result
- returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

ANY Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

Note: The *operator* must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

The SQL ALL Operator

The ALL operator:

- returns a boolean value as a result
- returns TRUE if ALL of the subquery values meet the condition
- is used with SELECT, WHERE and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

ALL Syntax With SELECT:

```
SELECT ALL column_name(s)
FROM table_name
WHERE condition;
```

ALL Syntax With WHERE or HAVING:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

Note: The *operator* must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

Demo Database

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

Product ID	Product Name	Supplier ID	Category ID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25

7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97

And a selection from the **"OrderDetails"** table:

OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40
6	10250	41	10
7	10250	51	35
8	10250	65	15
9	10251	22	6
10	10251	57	15

SQL ANY Examples

The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity equal to 10 (this will return TRUE because the Quantity column has some values of 10):

Example:

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity larger than 99 (this will return TRUE because the Quantity column has some values larger than 99):

Example:

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity > 99);
```

The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity larger than 1000 (this will return FALSE because the Quantity column has no values larger than 1000):

Example:

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity > 1000);
```

SQL ALL Examples

The following SQL statement lists ALL the product names:

Example:

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

The following SQL statement lists the ProductName if ALL the records in the OrderDetails table has Quantity equal to 10. This will of course return FALSE because the Quantity column has many different values (not only the value of 10):

Example:

```
SELECT ProductName
FROM Products
WHERE ProductID = ALL
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

SQL INSERT INTO SELECT Statement

The SQL INSERT INTO SELECT Statement

- The INSERT INTO SELECT statement copies data from one table and inserts it into another table.
- The INSERT INTO SELECT statement requires that the data types in source and target tables matches.
- **Note:** The existing records in the target table are unaffected.

INSERT INTO SELECT Syntax:

Copy all columns from one table to another table:

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

Copy only some columns from one table into another table:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

Demo Database

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

Below is a selection from the "Customers" table:

Customer ID	Customer Name	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

And a selection from the "Suppliers" table:

Supplier ID	Supplier Name	Contact Name	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

The following SQL statement copies "Suppliers" into "Customers" (the columns that are not filled with data, will contain NULL):

Example:

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

The following SQL statement copies "Suppliers" into "Customers" (fill all columns):

Example:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City,
PostalCode, Country)
SELECT SupplierName, ContactName, Address, City,
PostalCode, Country FROM Suppliers;
```

The following SQL statement copies only the German suppliers into "Customers":

Example:

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

SQL NULL Functions

SQL IFNULL(), ISNULL(), COALESCE(), and NVL() 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.
Look at 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 will be NULL.

Solutions

MySQL

The MySQL IFNULL() function lets you return an alternative value if an expression is NULL:

```
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;
```

SQL Server

The SQL Server ISNULL() function lets you return an alternative value when an expression is NULL:

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

MS Access

The MS Access IsNull() function returns TRUE (-1) if the expression is a null value, otherwise FALSE (0):

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

Oracle
The Oracle NVL() function achieves the same result:

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

SQL Comments

SQL Comments

Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.

Note: The examples in this chapter will not work in Firefox and Microsoft Edge!
Comments are not supported in Microsoft Access databases. Firefox and Microsoft Edge are using Microsoft Access database in our examples.

Single Line Comments

Single line comments start with --.
Any text between -- and the end of the line will be ignored (will not be executed).
The following example uses a single-line comment as an explanation:
Example:

```
--Select all:
SELECT * FROM Customers;
```

The following example uses a single-line comment to ignore the end of a line:

Example:

```
SELECT * FROM Customers -- WHERE City='Berlin';
```

The following example uses a single-line comment to ignore a statement:

Example:

```
--SELECT * FROM Customers;
SELECT * FROM Products;
```

Multi-line Comments

Multi-line comments start with /* and end with */.
Any text between /* and */ will be ignored.
The following example uses a multi-line comment as an explanation:

Example:

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```

The following example uses a multi-line comment to ignore many statements:

Example:

```
/*SELECT * FROM Customers;
SELECT * FROM Products;
SELECT * FROM Orders;
SELECT * FROM Categories;*/
SELECT * FROM Suppliers;
```

To ignore just a part of a statement, also use the /* */ comment.
The following example uses a comment to ignore part of a line:

Example:

```
SELECT CustomerName, /*City,*/ Country FROM Customers;
```

The following example uses a comment to ignore part of a statement:

Example:

```
SELECT * FROM Customers WHERE (CustomerName LIKE 'L%'
OR CustomerName LIKE 'R%' /*OR CustomerName LIKE 'S%'
OR CustomerName LIKE 'T%'*/ OR CustomerName LIKE 'W%')
AND Country='USA'
ORDER BY CustomerName;
```


SQL Operators

SQL Arithmetic Operators

Operator	Description
+	Add

-	Subtract
*	Multiply
/	Divide
%	Modulo

SQL Bitwise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

SQL Comparison Operators

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

SQL Compound Operators

Operator	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

SQL Logical Operators

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records

IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

SQL CREATE DATABASE Statement

The SQL CREATE DATABASE Statement

The CREATE DATABASE statement is used to create a new SQL database.

Syntax:

CREATE DATABASE *databasename*;

CREATE DATABASE Example

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

Example:

CREATE DATABASE testDB;

Tip: Make sure you have admin privilege before creating any database. Once a database is created, you can check it in the list of databases with the following SQL command: SHOW DATABASES;

SQL DROP DATABASE Statement

The SQL DROP DATABASE Statement

The DROP DATABASE statement is used to drop an existing SQL database.

Syntax:

DROP DATABASE *databasename*;

Note: Be careful before dropping a database. Deleting a database will result in loss of complete information stored in the database!

DROP DATABASE Example

The following SQL statement drops the existing database "testDB":

Example:

DROP DATABASE testDB;

SQL BACKUP DATABASE for SQL Server

The SQL BACKUP DATABASE Statement

The BACKUP DATABASE statement is used in SQL Server to create a full back up of an existing SQL database.

Syntax:

```
BACKUP DATABASE databasename
TO DISK = 'filepath';
```

The SQL BACKUP WITH DIFFERENTIAL Statement

A differential back up only backs up the parts of the database that have changed since the last full database backup.

Syntax:

```
BACKUP DATABASE databasename
TO DISK = 'filepath'
WITH DIFFERENTIAL;
```

BACKUP DATABASE Example

The following SQL statement creates a full back up of the existing database "testDB" to the D disk:

Example:

```
BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak';
```


Tip: Always back up the database to a different drive than the actual database. Then, if you get a disk crash, you will not lose your backup file along with the database.

BACKUP WITH DIFFERENTIAL Example

The following SQL statement creates a differential back up of the database "testDB":

Example:

```
BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak'
WITH DIFFERENTIAL;
```

Tip: A differential back up reduces the back-up time (since only the changes are backed up).

SQL CREATE TABLE Statement

The SQL CREATE TABLE Statement

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

Syntax:

```
CREATE TABLE table_name
   (column1 datatype,
   column2 datatype,
   column3 datatype,
  ....
);
```

The column parameters specify the names of the columns of the table.
The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

SQL CREATE TABLE Example

The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

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:

PersonID	LastName	FirstName	Address	City

Tip: The empty "Persons" table can now be filled with data with the SQL INSERT INTO statement.

Create Table Using Another Table

- A copy of an existing table can also be created using CREATE TABLE.
- The new table gets the same column definitions. All columns or specific columns can be selected.
- If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

Syntax:

```
CREATE TABLE new_table_name AS
   SELECT column1, column2,...
   FROM existing_table_name
   WHERE.....;
```

The following SQL creates a new table called "TestTables" (which is a copy of the "Customers" table):

Example:

```
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;
```

Tip: Make sure you have admin privilege before dropping any database. Once a database is dropped, you can check it in the list of databases with the following SQL command: SHOW DATABASES;

SQL DROP TABLE Statement

The SQL DROP TABLE Statement

The DROP TABLE statement is used to drop an existing table in a database.

Syntax:

```
DROP TABLE table_name;
```

Note: Be careful before dropping a table. Deleting a table will result in loss of complete information stored in the table!

SQL DROP TABLE Example

The following SQL statement drops the existing table "Shippers":

Example:

```
DROP TABLE Shippers;
```

SQL TRUNCATE TABLE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

Syntax:

TRUNCATE TABLE *table_name*;

SQL NOT NULL Constraint

SQL NOT NULL Constraint

- By default, a column can hold NULL values.
- The NOT NULL constraint enforces a column to NOT accept NULL values.
- This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

SQL NOT NULL on CREATE TABLE

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

Example:

```
CREATE  TABLE  Persons
   (ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255) NOT NULL,
   Age int
);
```


SQL NOT NULL on ALTER TABLE

To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:

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

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 column it will allow only certain values for this column.
- If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SQL CHECK on CREATE TABLE

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

MySQL:

```
CREATE TABLE Persons
   (ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   CHECK (Age>=18)
);
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
   (ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int CHECK (Age>=18)
);
```


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

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
   (ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   City varchar(255),
   CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
);
```

SQL CHECK on ALTER TABLE

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

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

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

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');
```

DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT CHK_PersonAge;
```

MySQL:

```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```


SQL CREATE INDEX Statement

SQL CREATE INDEX Statement

- The CREATE INDEX statement is used to create indexes in tables.
- Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

CREATE INDEX Syntax:

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

CREATE INDEX *index_name*
ON *table_name* (*column1*, *column2*, ...);

CREATE UNIQUE INDEX Syntax:
Creates a unique index on a table. Duplicate values are not allowed:

CREATE UNIQUE INDEX *index_name*
ON *table_name* (*column1*, *column2*, ...);

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

CREATE INDEX Example

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

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```


DROP INDEX Statement

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

MS Access:
DROP INDEX *index_name* ON *table_name*;

SQL Server:
DROP INDEX *table_name.index_name*;

DB2/Oracle:
DROP INDEX *index_name*;

MySQL:
ALTER TABLE *table_name*
DROP INDEX *index_name*;

SQL WORKING WITH DATES

SQL Dates

The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.

As long as your data contains only the date portion, your queries will work as expected. However, if a time portion is involved, it gets more complicated.

SQL Date Data Types

MySQL comes with the following data types for storing a date or a date/time value in the database:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY

SQL Server comes with the following data types for storing a date or a date/time value in the database:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- SMALLDATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: a unique number

Note: The date types are chosen for a column when you create a new table in your database!

SQL Working with Dates

Look at the following table:

Orders Table

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
2	Camembert Pierrot	2008-11-09
3	Mozzarella di Giovanni	2008-11-11
4	Mascarpone Fabioli	2008-10-29

Now we want to select the records with an OrderDate of "2008-11-11" from the table above.

We use the following SELECT statement:

```
SELECT * FROM Orders WHERE OrderDate='2008-11-11'
```

The result-set will look like this:

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
3	Mozzarella di Giovanni	2008-11-11

Note: Two dates can easily be compared if there is no time component involved!

Now, assume that the "Orders" table looks like this (notice the added time-component in the "OrderDate" column):

OrderId	ProductName	OrderDate
1	Geitost	2008-11-11 13:23:44
2	Camembert Pierrot	2008-11-09 15:45:21
3	Mozzarella di Giovanni	2008-11-11 11:12:01
4	Mascarpone Fabioli	2008-10-29 14:56:59

If we use the same SELECT statement as above:

```
SELECT * FROM Orders WHERE OrderDate='2008-11-11'
```

we will get no result! This is because the query is looking only for dates with no time portion.

Tip: To keep your queries simple and easy to maintain, do not use time-components in your dates, unless you have to!

SQL Injection

SQL Injection

- SQL injection is a code injection technique that might destroy your database.
- SQL injection is one of the most common web hacking techniques.
- SQL injection is the placement of malicious code in SQL statements, via web page input.

SQL in Web Pages

- SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will **unknowingly** run on your database.
- Look at the following example which creates a SELECT statement by adding a variable (txtUserId) to a select string. The variable is fetched from user input (getRequestString):

Example:

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

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

SQL Injection Based on 1=1 is Always True

- Look at the example above again. 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: 105 OR

- Then, the SQL statement will look like this:

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

- The SQL above is valid and will return ALL rows from the "Users" table, since **OR 1=1** is always TRUE.
- Does the example above look dangerous? What if the "Users" table contains names and passwords?
- The SQL statement above is much the same as this:

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

- A hacker might get access to all the user names and passwords in a database, by simply inserting 105 OR 1=1 into the input field.

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

Here is an example of a user login on a web site:

Username:
John Do

Password:
myPass

Example:
uName = getRequestString("username");
uPass = getRequestString("userpassword");

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

Result:
SELECT * FROM Users WHERE Name ="John Doe" AND Pass ="myPass"

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

User Name:
" or ""="
Password:
" or ""="

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

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

The SQL above is valid and will return all rows from the "Users" table, since **OR ""=""** is always TRUE.

SQL Injection Based on Batched SQL Statements

- Most databases support batched SQL statement.
- A batch of SQL statements is a group of two or more SQL statements, separated by semicolons.
- The SQL statement below will return all rows from the "Users" table, then delete the "Suppliers" table.

Example:
SELECT * FROM Users; DROP TABLE Suppliers

Look at the following example:

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

And the following input:

User id: 105; DR

The valid SQL statement would look like this:

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

Use SQL Parameters for Protection

- To protect a web site from SQL injection, you can 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);

- 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);

Examples:
The following examples shows how to build parameterized queries in some common web languages.
SELECT STATEMENT IN ASP.NET:

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

INSERT INTO STATEMENT IN ASP.NET:

```
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();
```

INSERT INTO STATEMENT IN PHP:

```
$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();
```

VIDYAPITH ACADEMY
WWW.VIDYAPITH.IN
ESTD 2020