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),
    Age int
);
```

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');
```

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".

SQL Views

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.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
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 Northwind 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;
```

Then, we can query the view as follows:

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

Another view in the Northwind 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);
```

We can query the view above as follows:

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

Another view in the Northwind 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. Let's 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 column1, column2, ...
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 OR REPLACE VIEW [Current Product List] AS
SELECT ProductID, ProductName, Category
FROM Products
WHERE Discontinued = No;
```

SQL Dropping a View

You can delete a view with the DROP VIEW command.

SQL DROP VIEW Syntax

```
DROP VIEW view_name;
```

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:

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:
105 or 1=1

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:

```
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:
John Doe

Password:
myPass

Server Code

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

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

Result

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

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:

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 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
```

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:
105; DROP TABLE Suppliers

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);
```

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.

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();
```

SQL Hosting

If you want your web site to be able to store and retrieve data from a database, your web server should have access to a database-system that uses the SQL language.

If your web server is hosted by an Internet Service Provider (ISP), you will have to look for SQL hosting plans.

The most common SQL hosting databases are MS SQL Server, Oracle, MySQL, and MS Access.

MS SQL Server

Microsoft's SQL Server is a popular database software for database-driven web sites with high traffic.

SQL Server is a very powerful, robust and full featured SQL database system.

Oracle

Oracle is also a popular database software for database-driven web sites with high traffic.

Oracle is a very powerful, robust and full featured SQL database system.

MySQL

MySQL is also a popular database software for web sites.

MySQL is a very powerful, robust and full featured SQL database system.

MySQL is an inexpensive alternative to the expensive Microsoft and Oracle solutions.

Access

When a web site requires only a simple database, Microsoft Access can be a solution.

Access is not well suited for very high-traffic, and not as powerful as MySQL, SQL Server, or Oracle.

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.

Function	Description
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
ROUND()	Rounds a numeric field to the number of decimals specified
SUM()	Returns the sum

SQL String Functions

Function	Description
CHARINDEX	Searches an expression in a string expression and returns its starting position if found
CONCAT()	
LEFT()	
LEN() / LENGTH()	Returns the length of the value in a text field
LOWER() / LCASE()	Converts character data to lower case
LTRIM()	
SUBSTRING() / MID()	Extract characters from a text field
PATINDEX()	
REPLACE()	
RIGHT()	
RTRIM()	
UPPER() / UCASE()	Converts character data to upper case

SQL 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 Northwind sample database.

Below is a selection from the "Products" table:

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	21.35
5	Chef Anton's Gumbo Mix	2	2	36 boxes	25

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);
```

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:

```
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 Northwind sample database.

Below is a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
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;
```

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;
```

SQL 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;
```

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
WHERE ROWNUM <=1
ORDER BY column_name ASC;
```

Example

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

Demo Database

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

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	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 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;
```

SQL 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;
```

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
WHERE ROWNUM <=1
ORDER BY column_name DESC;
```

Example

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

Demo Database

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

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	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 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;
```

SQL 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 Northwind sample database.

Below is a selection from the "Products" table:

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	21.35
5	Chef Anton's Gumbo Mix	2	2	36 boxes	25

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;
```

SQL 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 Northwind sample database.

Below is a selection from the "Products" table:

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	21.35
5	Chef Anton's Gumbo Mix	2	2	36 boxes	25

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;
```

SQL ROUND() Function

The ROUND() Function

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

Note: Many database systems do rounding differently than you might expect. When rounding a number with a fractional part to an integer, our school teachers told us to round .1 through .4 DOWN to the next lower integer, and .5 through .9 UP to the next higher integer. But if all the digits 1 through 9 are equally likely, this introduces a slight bias towards infinity, since we always round .5 up. Many database systems have adopted the IEEE 754 standard for arithmetic operations, according to which the default rounding behavior is "round half to even." In this scheme, .5 is rounded to the nearest even integer. So, both 11.5 and 12.5 would be rounded to 12.

SQL ROUND() Syntax

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

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 Northwind sample database.

Below is a selection from the "Products" table:

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	21.35
5	Chef Anton's Gumbo Mix	2	2	36 boxes	25

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;
```

SQL 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 Northwind sample database.

Below is 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

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;
```

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;
```

Syntax for Oracle

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

Demo Database

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

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	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 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;
```

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 Northwind sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	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 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;
```

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)
length	Optional. The number of characters to return. If omitted, the MID() function returns the rest of the text

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

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

Note: The equivalent function for Oracle is SUBSTR():

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

Demo Database

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

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	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 MID() Example

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

Example

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

SQL UCASE() Function

The UCASE() Function

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

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 Northwind sample database.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	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 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;
```

MySQL Date Functions

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

Function	Description
NOW()	Returns the current date and time
CURDATE()	Returns the current date
CURTIME()	Returns the current time
DATE()	Extracts the date part of a date or date/time expression
EXTRACT()	Returns a single part of a date/time
DATE_ADD()	Adds a specified time interval to a date
DATE_SUB()	Subtracts a specified time interval from a date
DATEDIFF()	Returns the number of days between two dates
DATE_FORMAT()	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
GETDATE()	Returns the current date and time
DATEPART()	Returns a single part of a date/time
DATEADD()	Adds or subtracts a specified time interval from a date
DATEDIFF()	Returns the time between two dates
CONVERT()	Displays date/time data in different formats

SQL Date and Time Data Types and Functions

Function	Description
FORMAT()	Formats how a field is to be displayed
NOW()	Returns the current system date and time

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!

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
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!

w3schools.com

THE WORLD'S LARGEST WEB DEVELOPER SITE

MySQL NOW() Function

Definition and Usage

NOW() returns the current date and time.

Syntax

```
NOW()
```

Example

The following SELECT statement:

```
SELECT NOW(),CURDATE(),CURTIME()
```

will result in something like this:

NOW()	CURDATE()	CURTIME()
2014-11-22 12:45:34	2014-11-22	12:45:34

Example

The following SQL creates an "Orders" table with a datetime column (OrderDate):

```
CREATE TABLE Orders
(
OrderId int NOT NULL,
ProductName varchar(50) NOT NULL,
OrderDate datetime NOT NULL DEFAULT NOW(),
PRIMARY KEY (OrderId)
)
```

Notice that the OrderDate column specifies NOW() as the default value. As a result, when you insert a row into the table, the current date and time are automatically inserted into the column.

Now we want to insert a record into the "Orders" table:

```
INSERT INTO Orders (ProductName) VALUES ('Jarlsberg Cheese')
```

The "Orders" table will now look something like this:

OrderId	ProductName	OrderDate
1	Jarlsberg Cheese	2014-11-22 13:23:44.657

w3schools.com

THE WORLD'S LARGEST WEB DEVELOPER SITE

MySQL CURDATE() Function

Definition and Usage

CURDATE() returns the current date.

Syntax

```
CURDATE()
```

Example

The following SELECT statement:

```
SELECT NOW(),CURDATE(),CURTIME()
```

will result in something like this:

NOW()	CURDATE()	CURTIME()
2014-11-22 12:45:34	2014-11-22	12:45:34

Example

The following SQL creates an "Orders" table with a datetime column (OrderDate):

```
CREATE TABLE Orders
(
OrderId int NOT NULL,
ProductName varchar(50) NOT NULL,
OrderDate datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (OrderId)
)
```

Notice that the OrderDate column specifies CURRENT_TIMESTAMP as the default value. As a result, when you insert a row into the table, the current date and time are automatically inserted into the column.

Now we want to insert a record into the "Orders" table:

```
INSERT INTO Orders (ProductName) VALUES ('Jarlsberg Cheese')
```

The "Orders" table will now look something like this:

OrderId	ProductName	OrderDate
1	Jarlsberg Cheese	2014-11-22 12:45:34

MySQL CURTIME() Function

Definition and Usage

CURTIME() returns the current time.

Syntax

```
CURTIME()
```

Example

The following SELECT statement:

```
SELECT NOW(),CURDATE(),CURTIME()
```

will result in something like this:

NOW()	CURDATE()	CURTIME()
2014-11-22 12:45:34	2014-11-22	12:45:34

MySQL DATE() Function

Definition and Usage

The DATE() function extracts the date part of a date or date/time expression.

Syntax

```
DATE(date)
```

Where date is a valid date expression.

Example

Assume we have the following "Orders" table:

OrderId	ProductName	OrderDate
1	Jarlsberg Cheese	2014-11-22 13:23:44.657

The following SELECT statement:

```
SELECT ProductName, DATE(OrderDate) AS OrderDate
FROM Orders
WHERE OrderId=1
```

will result in this:

ProductName	OrderDate
Jarlsberg Cheese	2014-11-22

MySQL EXTRACT() Function

Definition and Usage

The EXTRACT() function is used to return a single part of a date/time, such as year, month, day, hour, minute, etc.

Syntax

```
EXTRACT(unit FROM date)
```

Where date is a valid date expression and unit can be one of the following:

Unit Value
MICROSECOND
SECOND
MINUTE
HOUR
DAY
WEEK
MONTH
QUARTER
YEAR
SECOND_MICROSECOND
MINUTE_MICROSECOND
MINUTE_SECOND

HOUR_MICROSECOND
HOUR_SECOND
HOUR_MINUTE
DAY_MICROSECOND
DAY_SECOND
DAY_MINUTE
DAY_HOUR
YEAR_MONTH

Example

Assume we have the following "Orders" table:

OrderId	ProductName	OrderDate
1	Jarlsberg Cheese	2014-11-22 13:23:44.657

The following SELECT statement:

```
SELECT EXTRACT(YEAR FROM OrderDate) AS OrderYear,
EXTRACT(MONTH FROM OrderDate) AS OrderMonth,
EXTRACT(DAY FROM OrderDate) AS OrderDay
FROM Orders
WHERE OrderId=1
```

will result in this:

OrderYear	OrderMonth	OrderDay
2014	11	22

MySQL DATE_ADD() Function

Definition and Usage

The DATE_ADD() function adds a specified time interval to a date.

Syntax

```
DATE_ADD(date,INTERVAL expr type)
```

Where date is a valid date expression and expr is the number of interval you want to add.

type can be one of the following:

Type Value
MICROSECOND
SECOND
MINUTE
HOUR
DAY
WEEK
MONTH
QUARTER
YEAR
SECOND_MICROSECOND
MINUTE_MICROSECOND