Structured Query Language

Short Notes

Ashutosh: SQL

2017

## What is SQL?

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

## 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.

## 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).

## SQL Statements

Most of the actions you need to perform on a database are done with SQL statements.

## 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

## Single Line Comments

Single line comments start with --.

Any line between -- and the end of the line will be ignored (will not be executed).

## Multi-line Comments

Multi-line comments start with /\* and end with \*/.

Any text between /\* and \*/ will be ignored.

## The SQL SELECT Statement

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

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

### **SQL SELECT Syntax**

SELECT column\_name,column\_name  
FROM table\_name;

## SELECT Column Example

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

Ex: - SELECT CustomerName,City FROM Customers;

## SELECT \* Example

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

Ex: - SELECT \* FROM Customers;

## The SQL SELECT DISTINCT Statement

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

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

### **SQL SELECT DISTINCT Syntax**

SELECT DISTINCT column\_name,column\_name  
FROM table\_name;

## The SQL WHERE Clause

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

### **SQL WHERE Syntax**

SELECT column\_name,column\_name  
FROM table\_name  
WHERE column\_name operator value;

Operators in The WHERE Clause

The following operators can be used in the WHERE clause:

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

## The SQL AND & OR Operators

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

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

## The SQL ORDER BY Keyword

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

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

### **SQL ORDER BY Syntax**

SELECT column\_name,column\_name  
FROM table\_name  
ORDER BY column\_nameASC|DESC,column\_name ASC|DESC;

## The SQL INSERT INTO Statement

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

### **SQL INSERT INTO Syntax**

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

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

INSERT INTO table\_name  
VALUES (value1,value2,value3,...);

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

INSERT INTO table\_name (column1,column2,column3,...)  
VALUES (value1,value2,value3,...);

## The SQL UPDATE Statement

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

### **Syntax**

UPDATE table\_name  
SET column1=value1,column2=value2,...  
WHERE some\_column=some\_value;

## The SQL DELETE Statement

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

### **SQL DELETE Syntax**

DELETE FROM table\_name  
WHERE some\_column=some\_value;

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:

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

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

## The SQL SELECT TOP Clause

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

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

**Note:** Not all database systems support the SELECT TOP clause.

### **SQL Server / MS Access Syntax**

SELECT TOP number|percent column\_name(s)  
FROM table\_name;

### **MySQL Syntax**

SELECT column\_name(s)  
FROM table\_name  
LIMIT number;

### **Oracle Syntax**

SELECT column\_name(s)  
FROM table\_name  
WHERE ROWNUM <= number;

## The SQL LIKE Operator

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

### **SQL LIKE Syntax**

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name LIKE pattern;

SQL Wildcard Characters

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

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

With SQL, the wildcards are:

|  |  |
| --- | --- |
| **Wildcard** | **Description** |
| % | A substitute for zero or more characters |
| \_ | A substitute for a single character |
| [*charlist*] | Sets and ranges of characters to match |
| [^*charlist*] or [!*charlist*] | Matches only a character NOT specified within the brackets |

## The IN Operator

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

### **SQL IN Syntax**

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name IN (value1,value2,...);

## The SQL BETWEEN Operator

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

### **SQL BETWEEN Syntax**

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_nameBETWEEN value1 AND value2;

## The SQL BETWEEN Operator

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

### **SQL BETWEEN Syntax**

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_nameBETWEEN value1 AND value2;

## SQL JOIN

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

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

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

## Different SQL JOINs

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

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

## SQL INNER JOIN Keyword

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

### **SQL INNER JOIN Syntax**

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2  
ON table1.column\_name=table2.column\_name;

or:

SELECT column\_name(s)  
FROM table1  
JOIN table2  
ON table1.column\_name=table2.column\_name;

**PS!** INNER JOIN is the same as JOIN.



## SQL LEFT JOIN Keyword

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

### **SQL LEFT JOIN Syntax**

SELECT column\_name(s)  
FROM table1  
LEFT JOIN table2  
ON table1.column\_name=table2.column\_name;

or:

SELECT column\_name(s)  
FROM table1  
LEFT OUTER JOIN table2  
ON table1.column\_name=table2.column\_name;

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



## SQL RIGHT JOIN Keyword

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

### **SQL RIGHT JOIN Syntax**

SELECT column\_name(s)  
FROM table1  
RIGHT JOIN table2  
ON table1.column\_name=table2.column\_name;

or:

SELECT column\_name(s)  
FROM table1  
RIGHT OUTER JOIN table2  
ON table1.column\_name=table2.column\_name;

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



## SQL FULL OUTER JOIN Keyword

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

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

### **SQL FULL OUTER JOIN Syntax**

SELECT column\_name(s)  
FROM table1  
FULL OUTER JOIN table2  
ON table1.column\_name=table2.column\_name;



## The SQL UNION Operator

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

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

### **SQL UNION Syntax**

SELECT column\_name(s) FROM table1  
UNION  
SELECT column\_name(s) FROM table2;

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

### **SQL UNION ALL Syntax**

SELECT column\_name(s) FROM table1  
UNION ALL  
SELECT column\_name(s) FROM table2;

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

## The SQL SELECT INTO Statement

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

### **SQL SELECT INTO Syntax**

We can copy all columns into the new table:

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

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

SELECT column\_name(s)  
INTO newtable [IN externaldb]  
FROM table1;

## The SQL INSERT INTO SELECT Statement

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

### **SQL INSERT INTO SELECT Syntax**

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

INSERT INTO table2  
SELECT \* FROM table1;

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

INSERT INTO table2  
(column\_name(s))  
SELECT column\_name(s)  
FROM table1;

## The SQL CREATE DATABASE Statement

The CREATE DATABASE statement is used to create a database.

### **SQL CREATE DATABASE Syntax**

CREATE DATABASE dbname;

## SQL CREATE DATABASE Example

The following SQL statement creates a database called "my\_db":

CREATE DATABASE my\_db;

## The SQL CREATE TABLE Statement

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

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

### **SQL CREATE TABLE Syntax**

CREATE TABLE table\_name  
(  
column\_name1 data\_type(size),  
column\_name2 data\_type(size),  
column\_name3 data\_type(size),  
....  
);

The column\_name parameters specify the names of the columns of the table.

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

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

**Tip:** For an overview of the data types available in MS Access, MySQL, and SQL Server, go to our complete [Data Types Reference](http://www.w3schools.com/sql/sql_datatypes.asp).

## SQL Constraints

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

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

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

### **SQL CREATE TABLE + CONSTRAINT Syntax**

CREATE TABLE table\_name  
(  
column\_name1 data\_type(size) constraint\_name,  
column\_name2 data\_type(size) constraint\_name,  
column\_name3 data\_type(size) constraint\_name,  
....  
);

 In SQL, we have the following constraints:

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

## SQL NOT NULL Constraint

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

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

The following SQL enforces the "P\_Id" column and the "LastName" column to not accept NULL values:

## SQL UNIQUE Constraint

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

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

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

Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

## SQL PRIMARY KEY Constraint

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

Primary keys must contain UNIQUE values.

A primary key column cannot contain NULL values.

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

## SQL FOREIGN KEY Constraint

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

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

The "Persons" table:

**MySQL:**

CREATE TABLE Orders  
(  
O\_Id int NOT NULL,  
OrderNo int NOT NULL,  
P\_Id int,  
PRIMARY KEY (O\_Id),  
FOREIGN KEY (P\_Id) REFERENCES Persons(P\_Id)  
)

**SQL Server / Oracle / MS Access:**

CREATE TABLE Orders  
(  
O\_Id int NOT NULL PRIMARY KEY,  
OrderNo int NOT NULL,  
P\_Id int FOREIGN KEY REFERENCES Persons(P\_Id)  
)

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

**MySQL / SQL Server / Oracle / MS Access:**

CREATE TABLE Orders  
(  
O\_Id int NOT NULL,  
OrderNo int NOT NULL,  
P\_Id int,  
PRIMARY KEY (O\_Id),  
CONSTRAINT fk\_PerOrders FOREIGN KEY (P\_Id)  
REFERENCES Persons(P\_Id)  
)

**MySQL / SQL Server / Oracle / MS Access:**

ALTER TABLE Orders  
ADD FOREIGN KEY (P\_Id)  
REFERENCES Persons(P\_Id)

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

**MySQL / SQL Server / Oracle / MS Access:**

ALTER TABLE Orders  
ADD CONSTRAINT fk\_PerOrders  
FOREIGN KEY (P\_Id)  
REFERENCES Persons(P\_Id)

To DROP a FOREIGN KEY Constraint

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

**MySQL:**

ALTER TABLE Orders  
DROP FOREIGN KEY fk\_PerOrders

**SQL Server / Oracle / MS Access:**

ALTER TABLE Orders  
DROP CONSTRAINT fk\_PerOrders

## SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SQL CHECK Constraint on ALTER TABLE

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

**MySQL / SQL Server / Oracle / MS Access:**

ALTER TABLE Persons  
ADD CHECK (P\_Id>0)

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

**MySQL / SQL Server / Oracle / MS Access:**

ALTER TABLE Persons  
ADD CONSTRAINT chk\_Person CHECK (P\_Id>0 AND City='Sandnes')

To DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

**SQL Server / Oracle / MS Access:**

ALTER TABLE Persons  
DROP CONSTRAINT chk\_Person

**MySQL:**

ALTER TABLE Persons  
DROP CHECK chk\_Person

SQL DEFAULT Constraint

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

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

SQL DEFAULT Constraint on CREATE TABLE

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

**My SQL / SQL Server / Oracle / MS Access:**

CREATE TABLE Persons  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255) DEFAULT 'Sandnes'  
)

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

CREATE TABLE Orders  
(  
O\_Id int NOT NULL,  
OrderNo int NOT NULL,  
P\_Id int,  
OrderDate date DEFAULT GETDATE()  
)

SQL DEFAULT Constraint on ALTER TABLE

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

**MySQL:**

ALTER TABLE Persons  
ALTER City SET DEFAULT 'SANDNES'

**SQL Server / MS Access:**

ALTER TABLE Persons  
ALTER COLUMN City SET DEFAULT 'SANDNES'

**Oracle:**

ALTER TABLE Persons  
MODIFY City DEFAULT 'SANDNES'

To DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:

**MySQL:**

ALTER TABLE Persons  
ALTER City DROP DEFAULT

**SQL Server / Oracle / MS Access:**

ALTER TABLE Persons  
ALTER COLUMN City DROP DEFAULT

## Indexes

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

The users cannot see the indexes, they are just used to speed up searches/queries.

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

### **SQL CREATE INDEX Syntax**

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

CREATE INDEX index\_name  
ON table\_name (column\_name)

### **SQL CREATE UNIQUE INDEX Syntax**

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

CREATE UNIQUE INDEX index\_name  
ON table\_name (column\_name)

## The DROP INDEX Statement

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

### **DROP INDEX Syntax for MS Access:**

DROP INDEX index\_name ON table\_name

### **DROP INDEX Syntax for MS SQL Server:**

DROP INDEX table\_name.index\_name

### **DROP INDEX Syntax for DB2/Oracle:**

DROP INDEX index\_name

### **DROP INDEX Syntax for MySQL:**

ALTER TABLE table\_name DROP INDEX index\_name

## The DROP TABLE Statement

The DROP TABLE statement is used to delete a table.

DROP TABLE table\_name

## The DROP DATABASE Statement

The DROP DATABASE statement is used to delete a database.

DROP DATABASE database\_name

## The TRUNCATE TABLE Statement

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

Then, use the TRUNCATE TABLE statement:

TRUNCATE TABLE table\_name

## The ALTER TABLE Statement

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

### **SQL ALTER TABLE Syntax**

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

ALTER TABLE table\_name  
ADD column\_name datatype

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

ALTER TABLE table\_name  
DROP COLUMN column\_name

To change the data type of a column in a table, use the following syntax:

**SQL Server / MS Access:**

ALTER TABLE table\_name  
ALTER COLUMN column\_name datatype

**My SQL / Oracle (prior version 10G):**

ALTER TABLE table\_name  
MODIFY COLUMN column\_name datatype

**Oracle 10G and later:**

ALTER TABLE table\_name  
MODIFY column\_name datatype

DROP COLUMN Example

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

We use the following SQL statement:

ALTER TABLE Persons  
DROP COLUMN DateOfBirth

AUTO INCREMENT a Field

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

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

Syntax for MySQL

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

CREATE TABLE Persons  
(  
ID int NOT NULL AUTO\_INCREMENT,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255),  
PRIMARY KEY (ID)  
)

## SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

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

### **SQL CREATE VIEW Syntax**

CREATE VIEW view\_name AS  
SELECT column\_name(s)  
FROM table\_name  
WHERE condition

## SQL Dates

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

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

## MySQL Date Functions

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

|  |  |
| --- | --- |
| **Function** | **Description** |
| [NOW()](http://www.w3schools.com/sql/func_now.asp) | Returns the current date and time |
| [CURDATE()](http://www.w3schools.com/sql/func_curdate.asp) | Returns the current date |
| [CURTIME()](http://www.w3schools.com/sql/func_curtime.asp) | Returns the current time |
| [DATE()](http://www.w3schools.com/sql/func_date.asp) | Extracts the date part of a date or date/time expression |
| [EXTRACT()](http://www.w3schools.com/sql/func_extract.asp) | Returns a single part of a date/time |
| [DATE\_ADD()](http://www.w3schools.com/sql/func_date_add.asp) | Adds a specified time interval to a date |
| [DATE\_SUB()](http://www.w3schools.com/sql/func_date_sub.asp) | Subtracts a specified time interval from a date |
| [DATEDIFF()](http://www.w3schools.com/sql/func_datediff_mysql.asp) | Returns the number of days between two dates |
| [DATE\_FORMAT()](http://www.w3schools.com/sql/func_date_format.asp) | 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()](http://www.w3schools.com/sql/func_getdate.asp) | Returns the current date and time |
| [DATEPART()](http://www.w3schools.com/sql/func_datepart.asp) | Returns a single part of a date/time |
| [DATEADD()](http://www.w3schools.com/sql/func_dateadd.asp) | Adds or subtracts a specified time interval from a date |
| [DATEDIFF()](http://www.w3schools.com/sql/func_datediff.asp) | Returns the time between two dates |
| [CONVERT()](http://www.w3schools.com/sql/func_convert.asp) | Displays date/time data in different formats |

## SQL Date Data Types

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

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

**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 NULL Values

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

NULL values are treated differently from other values.

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

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

SQL General Data Types

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

SQL developers have to decide what types of data will be stored inside each and every table column when creating a SQL table. The data type is a label and a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

The following table lists the general data types in SQL:

|  |  |
| --- | --- |
| **Data type** | **Description** |
| CHARACTER(n) | Character string. Fixed-length n |
| VARCHAR(n) or CHARACTER VARYING(n) | Character string. Variable length. Maximum length n |
| BINARY(n) | Binary string. Fixed-length n |
| BOOLEAN | Stores TRUE or FALSE values |
| VARBINARY(n) or BINARY VARYING(n) | Binary string. Variable length. Maximum length n |
| INTEGER(p) | Integer numerical (no decimal). Precision p |
| SMALLINT | Integer numerical (no decimal). Precision 5 |
| INTEGER | Integer numerical (no decimal). Precision 10 |
| BIGINT | Integer numerical (no decimal). Precision 19 |
| DECIMAL(p,s) | Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal |
| NUMERIC(p,s) | Exact numerical, precision p, scale s. (Same as DECIMAL) |
| FLOAT(p) | Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation. The size argument for this type consists of a single number specifying the minimum precision |
| REAL | Approximate numerical, mantissa precision 7 |
| FLOAT | Approximate numerical, mantissa precision 16 |
| DOUBLE PRECISION | Approximate numerical, mantissa precision 16 |
| DATE | Stores year, month, and day values |
| TIME | Stores hour, minute, and second values |
| TIMESTAMP | Stores year, month, day, hour, minute, and second values |
| INTERVAL | Composed of a number of integer fields, representing a period of time, depending on the type of interval |
| ARRAY | A set-length and ordered collection of elements |
| MULTISET | A variable-length and unordered collection of elements |
| XML | Stores XML data |

SQL Data Type Quick Reference

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

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data type** | **Access** | **SQLServer** | **Oracle** | **MySQL** | **PostgreSQL** |
| *boolean* | Yes/No | Bit | Byte | N/A | Boolean |
| *integer* | Number (integer) | Int | Number | Int Integer | Int Integer |
| *float* | Number (single) | Float Real | Number | Float | Numeric |
| *currency* | Currency | Money | N/A | N/A | Money |
| *string (fixed)* | N/A | Char | Char | Char | Char |
| *string (variable)* | Text (<256) Memo (65k+) | Varchar | Varchar Varchar2 | Varchar | Varchar |
| *binary object* | OLE Object Memo | Binary (fixed up to 8K) Varbinary (<8K) Image (<2GB) | Long Raw | Blob Text | Binary Varbinary |

MySQL Data Types

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

**Text types:**

|  |  |
| --- | --- |
| **Data type** | **Description** |
| CHAR(size) | Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters |
| VARCHAR(size) | Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. **Note:** If you put a greater value than 255 it will be converted to a TEXT type |
| TINYTEXT | Holds a string with a maximum length of 255 characters |
| TEXT | Holds a string with a maximum length of 65,535 characters |
| BLOB | For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data |
| MEDIUMTEXT | Holds a string with a maximum length of 16,777,215 characters |
| MEDIUMBLOB | For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data |
| LONGTEXT | Holds a string with a maximum length of 4,294,967,295 characters |
| LONGBLOB | For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data |
| ENUM(x,y,z,etc.) | Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted.  **Note:** The values are sorted in the order you enter them.  You enter the possible values in this format: ENUM('X','Y','Z') |
| SET | Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice |

**Number types:**

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

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

**Date types:**

|  |  |
| --- | --- |
| **Data type** | **Description** |
| DATE() | A date. Format: YYYY-MM-DD  **Note:** The supported range is from '1000-01-01' to '9999-12-31' |
| DATETIME() | \*A date and time combination. Format: YYYY-MM-DD HH:MI:SS  **Note:** The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59' |
| TIMESTAMP() | \*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MI:SS  **Note:** The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC |
| TIME() | A time. Format: HH:MI:SS  **Note:** The supported range is from '-838:59:59' to '838:59:59' |
| YEAR() | A year in two-digit or four-digit format.  **Note:** Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069 |

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

SQL Server Data Types

**String types:**

|  |  |  |
| --- | --- | --- |
| **Data type** | **Description** | **Storage** |
| char(n) | Fixed width character string. Maximum 8,000 characters | Defined width |
| varchar(n) | Variable width character string. Maximum 8,000 characters | 2 bytes + number of chars |
| varchar(max) | Variable width character string. Maximum 1,073,741,824 characters | 2 bytes + number of chars |
| text | Variable width character string. Maximum 2GB of text data | 4 bytes + number of chars |
| nchar | Fixed width Unicode string. Maximum 4,000 characters | Defined width x 2 |
| nvarchar | Variable width Unicode string. Maximum 4,000 characters |  |
| nvarchar(max) | Variable width Unicode string. Maximum 536,870,912 characters |  |
| ntext | Variable width Unicode string. Maximum 2GB of text data |  |
| bit | Allows 0, 1, or NULL |  |
| binary(n) | Fixed width binary string. Maximum 8,000 bytes |  |
| varbinary | Variable width binary string. Maximum 8,000 bytes |  |
| varbinary(max) | Variable width binary string. Maximum 2GB |  |
| image | Variable width binary string. Maximum 2GB |  |

**Number types:**

|  |  |  |
| --- | --- | --- |
| **Data type** | **Description** | **Storage** |
| tinyint | Allows whole numbers from 0 to 255 | 1 byte |
| smallint | Allows whole numbers between -32,768 and 32,767 | 2 bytes |
| int | Allows whole numbers between -2,147,483,648 and 2,147,483,647 | 4 bytes |
| bigint | Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807 | 8 bytes |
| decimal(p,s) | Fixed precision and scale numbers.  Allows numbers from -10^38 +1 to 10^38 –1.  The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18.  The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0 | 5-17 bytes |
| numeric(p,s) | Fixed precision and scale numbers.  Allows numbers from -10^38 +1 to 10^38 –1.  The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18.  The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0 | 5-17 bytes |
| smallmoney | Monetary data from -214,748.3648 to 214,748.3647 | 4 bytes |
| money | Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807 | 8 bytes |
| float(n) | Floating precision number data from -1.79E + 308 to 1.79E + 308.  The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53. | 4 or 8 bytes |
| real | Floating precision number data from -3.40E + 38 to 3.40E + 38 | 4 bytes |

**Date types:**

|  |  |  |
| --- | --- | --- |
| **Data type** | **Description** | **Storage** |
| datetime | From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds | 8 bytes |
| datetime2 | From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds | 6-8 bytes |
| smalldatetime | From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute | 4 bytes |
| date | Store a date only. From January 1, 0001 to December 31, 9999 | 3 bytes |
| time | Store a time only to an accuracy of 100 nanoseconds | 3-5 bytes |
| datetimeoffset | The same as datetime2 with the addition of a time zone offset | 8-10 bytes |
| timestamp | Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable |  |

**Other data types:**

|  |  |
| --- | --- |
| **Data type** | **Description** |
| sql\_variant | Stores up to 8,000 bytes of data of various data types, except text, ntext, and timestamp |
| uniqueidentifier | Stores a globally unique identifier (GUID) |
| xml | Stores XML formatted data. Maximum 2GB |
| cursor | Stores a reference to a cursor used for database operations |
| table | Stores a result-set for later processing |

## SQL Aggregate Functions

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

Useful aggregate functions:

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

## SQL Scalar functions

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

Useful scalar functions:

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

## The GROUP BY Statement

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

### **SQL GROUP BY Syntax**

SELECT column\_name, aggregate\_function(column\_name)  
FROM table\_name  
WHERE column\_name operator value  
GROUP BY column\_name;

## The HAVING Clause

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

### **SQL HAVING Syntax**

SELECT column\_name, aggregate\_function(column\_name)  
FROM table\_name  
WHERE column\_name operator value  
GROUP BY column\_name  
HAVING aggregate\_function(column\_name) operator value;

|  |  |
| --- | --- |
| **SQL Statement** | **Syntax** |
| AND / OR | SELECT column\_name(s) FROM table\_name WHERE condition AND|OR condition |
| ALTER TABLE | ALTER TABLE table\_name  ADD column\_name datatype  or  ALTER TABLE table\_name  DROP COLUMN column\_name |
| AS (alias) | SELECT column\_name AS column\_alias FROM table\_name  or  SELECT column\_name FROM table\_name  AS table\_alias |
| BETWEEN | SELECT column\_name(s) FROM table\_name WHERE column\_name BETWEEN value1 AND value2 |
| CREATE DATABASE | CREATE DATABASE database\_name |
| CREATE TABLE | CREATE TABLE table\_name ( column\_name1 data\_type, column\_name2 data\_type, column\_name3 data\_type, ... ) |
| CREATE INDEX | CREATE INDEX index\_name ON table\_name (column\_name)  or  CREATE UNIQUE INDEX index\_name ON table\_name (column\_name) |
| CREATE VIEW | CREATE VIEW view\_name AS SELECT column\_name(s) FROM table\_name WHERE condition |
| DELETE | DELETE FROM table\_name WHERE some\_column=some\_value  or  DELETE FROM table\_name  (**Note:**Deletes the entire table!!)  DELETE \* FROM table\_name  (**Note:**Deletes the entire table!!) |
| DROP DATABASE | DROP DATABASE database\_name |
| DROP INDEX | DROP INDEX table\_name.index\_name (SQL Server) DROP INDEX index\_name ON table\_name (MS Access) DROP INDEX index\_name (DB2/Oracle) ALTER TABLE table\_name DROP INDEX index\_name (MySQL) |
| DROP TABLE | DROP TABLE table\_name |
| EXISTS | IF EXISTS (SELECT \* FROM table\_name WHERE id = ?) BEGIN --do what needs to be done if exists END ELSE BEGIN --do what needs to be done if not END |
| GROUP BY | SELECT column\_name, aggregate\_function(column\_name) FROM table\_name WHERE column\_name operator value GROUP BY column\_name |
| HAVING | SELECT column\_name, aggregate\_function(column\_name) FROM table\_name WHERE column\_name operator value GROUP BY column\_name HAVING aggregate\_function(column\_name) operator value |
| IN | SELECT column\_name(s) FROM table\_name WHERE column\_name IN (value1,value2,..) |
| INSERT INTO | INSERT INTO table\_name VALUES (value1, value2, value3,....)  *or*  INSERT INTO table\_name (column1, column2, column3,...) VALUES (value1, value2, value3,....) |
| INNER JOIN | SELECT column\_name(s) FROM table\_name1 INNER JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| LEFT JOIN | SELECT column\_name(s) FROM table\_name1 LEFT JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| RIGHT JOIN | SELECT column\_name(s) FROM table\_name1 RIGHT JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| FULL JOIN | SELECT column\_name(s) FROM table\_name1 FULL JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| LIKE | SELECT column\_name(s) FROM table\_name WHERE column\_name LIKE pattern |
| ORDER BY | SELECT column\_name(s) FROM table\_name ORDER BY column\_name [ASC|DESC] |
| SELECT | SELECT column\_name(s) FROM table\_name |
| SELECT \* | SELECT \* FROM table\_name |
| SELECT DISTINCT | SELECT DISTINCT column\_name(s) FROM table\_name |
| SELECT INTO | SELECT \* INTO new\_table\_name [IN externaldatabase] FROM old\_table\_name  *or*  SELECT column\_name(s) INTO new\_table\_name [IN externaldatabase] FROM old\_table\_name |
| SELECT TOP | SELECT TOP number|percent column\_name(s) FROM table\_name |
| TRUNCATE TABLE | TRUNCATE TABLE table\_name |
| UNION | SELECT column\_name(s) FROM table\_name1 UNION SELECT column\_name(s) FROM table\_name2 |
| UNION ALL | SELECT column\_name(s) FROM table\_name1 UNION ALL SELECT column\_name(s) FROM table\_name2 |
| UPDATE | UPDATE table\_name SET column1=value, column2=value,... WHERE some\_column=some\_value |
| WHERE | SELECT column\_name(s) FROM table\_name WHERE column\_name operator value |