**SQL**

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

SELECT:

SELECT column1, column2, ... FROM table\_name;

SELECT \* FROM table\_name;

SELECT \* FROM Customers;

SELECT CustomerName, City FROM Customers;

The SQL SELECT DISTINCT Statement

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

SELECT DISTINCT column1, column2, ...  
FROM table\_name;

SELECT DISTINCT Country FROM Customers;

SELECT COUNT(DISTINCT Country) FROM Customers;

SELECT Count(\*) AS DistinctCountries

FROM (SELECT DISTINCT Country FROM Customers);

### **WHERE Syntax**

SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

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

Operators in The WHERE Clause

The following operators can be used in the WHERE clause:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Equal | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_equal_to) |
| > | Greater than | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_greater_than) |
| < | Less than | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_less_than) |
| >= | Greater than or equal | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_greater_than2) |
| <= | Less than or equal | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_less_than2) |
| <> | Not equal. **Note:** In some versions of SQL this operator may be written as != | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_not_equal_to) |
| BETWEEN | Between a certain range | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_between) |
| LIKE | Search for a pattern | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_like) |
| IN | To specify multiple possible values for a column |  |

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

SELECT \* FROM Customers  
WHERE CustomerID=1;

SELECT \* FROM Products

WHERE Price = 18;

SELECT \* FROM Products

WHERE Price > 30;

SELECT \* FROM Products

WHERE Price < 30;

SELECT \* FROM Products

WHERE Price < =30;

SELECT \* FROM Products

WHERE Price >=30;

SELECT \* FROM Products

WHERE Price <> 18;

SELECT \* FROM Products

WHERE Price BETWEEN 50 AND 60;

SELECT \* FROM Customers

WHERE City LIKE 's%';

SELECT \* FROM Customers

WHERE City IN ('Paris','London');

## 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;

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

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

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

SELECT \* FROM Customers  
WHERE NOT Country='Germany';

## Combining AND, OR and NOT

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

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

## 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;

SELECT \* FROM Customers  
ORDER BY Country;

SELECT \* FROM Customers  
ORDER BY Country DESC;

## ORDER BY Several Columns Example

## SELECT \* FROM Customers ORDER BY Country, CustomerName;

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

## The SQL INSERT INTO Statement

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

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

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

## Insert Data Only in Specified Columns

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

## 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\_namesFROM table\_name  
WHERE column\_name IS NULL;

### **IS NOT NULL Syntax**

SELECT column\_namesFROM table\_name  
WHERE column\_name IS NOT NULL;

## The IS NULL Operator

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

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).

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

### **UPDATE Syntax**

UPDATE table\_name  
SET column1 = value1, column2 = value2, ...  
WHERE condition;

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

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

If you omit the WHERE clause, ALL records will be updated!

UPDATE Customers  
SET ContactName='Juan';(all records change)

### **DELETE Syntax**

DELETE FROM table\_name WHERE condition;

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

\*delete all records

DELETE FROM table\_name;

DELETE FROM Customers;

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*;

SELECT TOP 3 \* FROM Customers;

SELECT \* FROM Customers  
LIMIT 3;

SELECT \* FROM Customers  
FETCH FIRST 3 ROWS ONLY;

SELECT TOP 50 PERCENT \* FROM Customers;

SELECT \* FROM Customers  
FETCH FIRST 50 PERCENT ROWS ONLY;

## ADD a WHERE CLAUSE

SELECT TOP 3 \* FROM Customers  
WHERE Country='Germany';

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

SELECT \* FROM Customers  
WHERE Country='Germany'  
FETCH FIRST 3 ROWS ONLY;

## 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;

SELECT MIN(Price) AS SmallestPrice  
FROM Products;

SELECT MAX(Price) AS LargestPrice  
FROM Products;

## 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;

SELECT COUNT(ProductID)  
FROM Products;

NULL values are not counted.

SELECT AVG(Price)  
FROM Products;

NULL values are ignored.

SELECT SUM(Quantity)  
FROM OrderDetails;

 NULL values are ignored.

## The SQL LIKE Operator

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

There are two wildcards often used in conjunction with the LIKE operator:

* The percent sign (%) represents zero, one, or multiple characters
* The underscore sign (\_) represents one, single character

**Note:** MS Access uses an asterisk (\*) instead of the percent sign (%), and a question mark (?) instead of the underscore (\_).

The percent sign and the underscore can also be used in combinations!

### **LIKE Syntax**

SELECT column1, column2, ...  
FROM table\_name  
WHERE columnN LIKE pattern;

**Tip:** You can also combine any number of conditions using AND or OR operators.

|  |  |
| --- | --- |
| **LIKE Operator** | **Description** |
| WHERE CustomerName LIKE 'a%' | Finds any values that start with "a" |
| WHERE CustomerName LIKE '%a' | Finds any values that end with "a" |
| WHERE CustomerName LIKE '%or%' | Finds any values that have "or" in any position |
| WHERE CustomerName LIKE '\_r%' | Finds any values that have "r" in the second position |
| WHERE CustomerName LIKE 'a\_%' | Finds any values that start with "a" and are at least 2 characters in length |
| WHERE CustomerName LIKE 'a\_\_%' | Finds any values that start with "a" and are at least 3 characters in length |
| WHERE ContactName LIKE 'a%o' | Finds any values that start with "a" and ends with "o" |

SELECT \* FROM Customers  
WHERE CustomerName LIKE 'a%';

SELECT \* FROM Customers  
WHERE CustomerName LIKE '%a';

SELECT \* FROM Customers  
WHERE CustomerName LIKE '%or%';

SELECT \* FROM Customers  
WHERE CustomerName LIKE '\_r%';

SELECT \* FROM Customers  
WHERE CustomerName LIKE 'a\_\_%';

SELECT \* FROM Customers  
WHERE ContactName LIKE 'a%o';

SELECT \* FROM Customers  
WHERE CustomerName NOT LIKE 'a%';

## The SQL IN Operator

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

The IN operator is a shorthand for multiple OR conditions.

### **IN Syntax**

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

or:

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name IN (*SELECT* STATEMENT);

SELECT \* FROM Customers  
WHERE Country IN ('Germany', 'France', 'UK');

SELECT \* FROM Customers  
WHERE Country NOT IN ('Germany', 'France', 'UK');

SELECT \* FROM Customers  
WHERE Country IN (SELECT Country FROM Suppliers);

## 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;

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

## NOT BETWEEN Example

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

## BETWEEN with IN Example

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

## BETWEEN Text Values Example

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

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

## NOT BETWEEN Text Values Example

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

## BETWEEN Dates Example

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

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

## SQL JOIN

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

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

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 INNER JOIN Keyword

The INNER JOIN keyword selects records that have matching values in both tables.

### **INNER JOIN Syntax**

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2ON table1.column\_name = table2.column\_name;



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

## JOIN Three Tables

SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName  
FROM ((Orders  
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)  
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);

## SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

### **LEFT JOIN Syntax**

SELECT column\_name(s)  
FROM table1  
LEFT JOIN table2ON table1.column\_name = table2.column\_name;

**Note:** In some databases LEFT JOIN is called LEFT OUTER JOIN.



SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID  
ORDER BY Customers.CustomerName;

## SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

### **RIGHT JOIN Syntax**

SELECT column\_name(s)  
FROM table1  
RIGHT JOIN table2ON table1.column\_name = table2.column\_name;

**Note:** In some databases RIGHT JOIN is called RIGHT OUTER JOIN.



SELECT Orders.OrderID, Employees.LastName, Employees.FirstName  
FROM Orders  
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
ORDER BY Orders.OrderID;

## SQL FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

**Tip:** FULL OUTER JOIN and FULL JOIN are the same.

### **FULL OUTER JOIN Syntax**

SELECT column\_name(s)  
FROM table1  
FULL OUTER JOIN table2ON table1.column\_name = table2.column\_nameWHERE condition;



**Note:** FULL OUTER JOIN can potentially return very large result-sets!

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID  
ORDER BY Customers.CustomerName;

## 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.

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

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

## SQL UNION With WHERE

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

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

## The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

### **GROUP BY Syntax**

SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)ORDER BY column\_name(s);

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country;

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
ORDER BY COUNT(CustomerID) DESC;

## GROUP BY With JOIN Example

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

## The SQL HAVING Clause

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

### **HAVING Syntax**

SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)HAVING conditionORDER BY column\_name(s);

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5;

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5  
ORDER BY COUNT(CustomerID) DESC;

SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
WHERE LastName = 'Davolio' OR LastName = 'Fuller'  
GROUP BY LastName  
HAVING COUNT(Orders.OrderID) > 25;

SQL Arithmetic Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Add | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_add) |
| - | Subtract | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_subtract) |
| \* | Multiply | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_multiply) |
| / | Divide | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_divide) |
| % | Modulo | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_modulo) |

SQL Bitwise Operators

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

SQL Comparison Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Equal to | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_equal_to) |
| > | Greater than | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_greater_than) |
| < | Less than | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_less_than) |
| >= | Greater than or equal to | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_greater_than2) |
| <= | Less than or equal to | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_less_than2) |
| <> | 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** | **Example** |
| ALL | TRUE if all of the subquery values meet the condition | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_all&ss=-1) |
| AND | TRUE if all the conditions separated by AND is TRUE | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_and) |
| ANY | TRUE if any of the subquery values meet the condition | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_any&ss=-1) |
| BETWEEN | TRUE if the operand is within the range of comparisons | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_between) |
| EXISTS | TRUE if the subquery returns one or more records | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_exists) |
| IN | TRUE if the operand is equal to one of a list of expressions | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_in) |
| LIKE | TRUE if the operand matches a pattern | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_like) |
| NOT | Displays a record if the condition(s) is NOT TRUE | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_not) |
| OR | TRUE if any of the conditions separated by OR is TRUE | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_or) |
| SOME | TRUE if any of the subquery values meet the condition |  |

## The SQL CREATE DATABASE Statement

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

### **Syntax**

CREATE DATABASE databasename;

CREATE DATABASE testDB;

## 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 testDB;

## 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,  
   ....  
);

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

## 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;

## The SQL DROP TABLE Statement

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

### **Syntax**

DROP TABLE table\_name;

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 ALTER TABLE Statement

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

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

## ALTER TABLE - ADD Column

ALTER TABLE table\_name  
ADD column\_name datatype;

ALTER TABLE Customers  
ADD Email varchar(255);

## ALTER TABLE - DROP COLUMN

ALTER TABLE table\_name  
DROP COLUMN column\_name;

ALTER TABLE Customers  
DROP COLUMN Email;

## ALTER TABLE - RENAME COLUMN

ALTER TABLE table\_name  
RENAME COLUMN old\_name to new\_name;

ALTER TABLE - ALTER/MODIFY DATATYPE

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*;

ALTER TABLE Persons  
ADD DateOfBirth date;

## Change Data Type Example

ALTER TABLE Persons  
ALTER COLUMN DateOfBirth year;

## DROP COLUMN Example

ALTER TABLE Persons  
DROP COLUMN DateOfBirth;

## SQL Create Constraints

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

### **Syntax**

CREATE TABLE table\_name (  
    column1 datatype *constraint*,  
    column2 datatype *constraint*,  
    column3 datatype *constraint*,  
    ....  
);

## SQL Constraints

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

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

* [NOT NULL](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value
* [UNIQUE](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different
* [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [FOREIGN KEY](https://www.w3schools.com/sql/sql_foreignkey.asp) - Prevents actions that would destroy links between tables
* [CHECK](https://www.w3schools.com/sql/sql_check.asp) - Ensures that the values in a column satisfies a specific condition
* [DEFAULT](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column if no value is specified
* [CREATE INDEX](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly

## 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:

**SQL Server / MS Access:**

ALTER TABLE Persons  
ALTER COLUMN Age int NOT NULL;

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

ALTER TABLE Persons  
MODIFY COLUMN Age int NOT NULL;

**Oracle 10G and later:**

ALTER TABLE Persons  
MODIFY Age int NOT NULL;

SQL UNIQUE Constraint

The UNIQUE constraint ensures that all values in a column are different.

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

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

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

SQL UNIQUE Constraint on CREATE TABLE

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

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

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

**MySQL:**

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

To name a UNIQUE constraint, and to define a UNIQUE 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,  
    CONSTRAINT UC\_Person UNIQUE (ID,LastName)  
);

SQL UNIQUE Constraint on ALTER TABLE

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

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

ALTER TABLE Persons  
ADD UNIQUE (ID);

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

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

ALTER TABLE Persons  
ADD CONSTRAINT UC\_Person UNIQUE (ID,LastName);

DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

**MySQL:**

ALTER TABLE Persons  
DROP INDEX UC\_Person;

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

ALTER TABLE Persons  
DROP CONSTRAINT UC\_Person;

SQL PRIMARY KEY Constraint

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

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

SQL PRIMARY KEY on CREATE TABLE

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

**MySQL:**

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

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

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

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

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

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName)  
);

**Note:** In the example above there is only ONE PRIMARY KEY (PK\_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).

SQL PRIMARY KEY on ALTER TABLE

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

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

ALTER TABLE Persons  
ADD PRIMARY KEY (ID);

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

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

ALTER TABLE Persons  
ADD CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName);

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

DROP a PRIMARY KEY Constraint

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

**MySQL:**

ALTER TABLE Persons  
DROP PRIMARY KEY;

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

ALTER TABLE Persons  
DROP CONSTRAINT PK\_Person;

## SQL FOREIGN KEY Constraint

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

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

SQL FOREIGN KEY on CREATE TABLE

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

**MySQL:**

CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)  
);

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

CREATE TABLE Orders (  
    OrderID int NOT NULL PRIMARY KEY,  
    OrderNumber int NOT NULL,  
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)  
);

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 (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    CONSTRAINT FK\_PersonOrder FOREIGN KEY (PersonID)  
    REFERENCES Persons(PersonID)  
);

SQL FOREIGN KEY on ALTER TABLE

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

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

ALTER TABLE Orders  
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

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\_PersonOrder  
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

DROP a FOREIGN KEY Constraint

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

**MySQL:**

ALTER TABLE Orders  
DROP FOREIGN KEY FK\_PersonOrder;

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

ALTER TABLE Orders  
DROP CONSTRAINT FK\_PersonOrder;

AUTO INCREMENT Field

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

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

Syntax for MySQL

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

CREATE TABLE Persons (  
    Personid int NOT NULL AUTO\_INCREMENT,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    PRIMARY KEY (Personid)  
);

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!

## MySQL Data Types (Version 8.0)

In MySQL there are three main data types: string, numeric, and date and time.

### **String Data Types**

|  |  |
| --- | --- |
| **Data type** | **Description** |
| CHAR(size) | A FIXED length string (can contain letters, numbers, and special characters). The size parameter specifies the column length in characters - can be from 0 to 255. Default is 1 |
| VARCHAR(size) | A VARIABLE length string (can contain letters, numbers, and special characters). The size parameter specifies the maximum string length in characters - can be from 0 to 65535 |
| BINARY(size) | Equal to CHAR(), but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1 |
| VARBINARY(size) | Equal to VARCHAR(), but stores binary byte strings. The size parameter specifies the maximum column length in bytes. |
| TINYBLOB | For BLOBs (Binary Large Objects). Max length: 255 bytes |
| TINYTEXT | Holds a string with a maximum length of 255 characters |
| TEXT(size) | Holds a string with a maximum length of 65,535 bytes |
| BLOB(size) | 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(val1, val2, val3, ...) | A string object that can have only one value, chosen from 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. The values are sorted in the order you enter them |
| SET(val1, val2, val3, ...) | A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list |

### **Numeric Data Types**

|  |  |
| --- | --- |
| **Data type** | **Description** |
| BIT(size) | A bit-value type. The number of bits per value is specified in size. The size parameter can hold a value from 1 to 64. The default value for size is 1. |
| TINYINT(size) | A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The size parameter specifies the maximum display width (which is 255) |
| BOOL | Zero is considered as false, nonzero values are considered as true. |
| BOOLEAN | Equal to BOOL |
| SMALLINT(size) | A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The size parameter specifies the maximum display width (which is 255) |
| MEDIUMINT(size) | A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The size parameter specifies the maximum display width (which is 255) |
| INT(size) | A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255) |
| INTEGER(size) | Equal to INT(size) |
| BIGINT(size) | A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The size parameter specifies the maximum display width (which is 255) |
| FLOAT(size, d) | A floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions |
| FLOAT(p) | A floating point number. MySQL uses the p value to determine whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT(). If p is from 25 to 53, the data type becomes DOUBLE() |
| DOUBLE(size, d) | A normal-size floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter |
| DOUBLE PRECISION(size, d) |  |
| DECIMAL(size, d) | An exact fixed-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. The maximum number for size is 65. The maximum number for d is 30. The default value for size is 10. The default value for d is 0. |
| DEC(size, d) | Equal to DECIMAL(size,d) |

**Note:** All the numeric data types may have an extra option: UNSIGNED or ZEROFILL. If you add the UNSIGNED option, MySQL disallows negative values for the column. If you add the ZEROFILL option, MySQL automatically also adds the UNSIGNED attribute to the column.

### **Date and Time Data Types**

|  |  |
| --- | --- |
| **Data type** | **Description** |
| DATE | A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31' |
| DATETIME(fsp) | A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time |
| TIMESTAMP(fsp) | 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:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT\_TIMESTAMP and ON UPDATE CURRENT\_TIMESTAMP in the column definition |
| TIME(fsp) | A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59' |
| YEAR | A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format. |

## SQL Server Data Types

### **String Data Types**

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

### **Numeric Data Types**

|  |  |  |
| --- | --- | --- |
| **Data type** | **Description** | **Storage** |
| bit | Integer that can be 0, 1, or NULL |  |
| 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 and Time Data 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 |

## MS Access Data Types

|  |  |  |
| --- | --- | --- |
| **Data type** | **Description** | **Storage** |
| Text | Use for text or combinations of text and numbers. 255 characters maximum |  |
| Memo | Memo is used for larger amounts of text. Stores up to 65,536 characters. **Note:** You cannot sort a memo field. However, they are searchable |  |
| Byte | Allows whole numbers from 0 to 255 | 1 byte |
| Integer | Allows whole numbers between -32,768 and 32,767 | 2 bytes |
| Long | Allows whole numbers between -2,147,483,648 and 2,147,483,647 | 4 bytes |
| Single | Single precision floating-point. Will handle most decimals | 4 bytes |
| Double | Double precision floating-point. Will handle most decimals | 8 bytes |
| Currency | Use for currency. Holds up to 15 digits of whole dollars, plus 4 decimal places. **Tip:** You can choose which country's currency to use | 8 bytes |
| AutoNumber | AutoNumber fields automatically give each record its own number, usually starting at 1 | 4 bytes |
| Date/Time | Use for dates and times | 8 bytes |
| Yes/No | A logical field can be displayed as Yes/No, True/False, or On/Off. In code, use the constants True and False (equivalent to -1 and 0). **Note:** Null values are not allowed in Yes/No fields | 1 bit |
| Ole Object | Can store pictures, audio, video, or other BLOBs (Binary Large Objects) | up to 1GB |
| Hyperlink | Contain links to other files, including web pages |  |
| Lookup Wizard | Let you type a list of options, which can then be chosen from a drop-down list | 4 bytes |

SQL Keywords

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| [ADD](https://www.w3schools.com/sql/sql_ref_add.asp) | Adds a column in an existing table |
| [ADD CONSTRAINT](https://www.w3schools.com/sql/sql_ref_add_constraint.asp) | Adds a constraint after a table is already created |
| [ALL](https://www.w3schools.com/sql/sql_ref_all.asp) | Returns true if all of the subquery values meet the condition |
| [ALTER](https://www.w3schools.com/sql/sql_ref_alter.asp) | Adds, deletes, or modifies columns in a table, or changes the data type of a column in a table |
| [ALTER COLUMN](https://www.w3schools.com/sql/sql_ref_alter_column.asp) | Changes the data type of a column in a table |
| [ALTER TABLE](https://www.w3schools.com/sql/sql_ref_alter_table.asp) | Adds, deletes, or modifies columns in a table |
| [AND](https://www.w3schools.com/sql/sql_ref_and.asp) | Only includes rows where both conditions is true |
| [ANY](https://www.w3schools.com/sql/sql_ref_any.asp) | Returns true if any of the subquery values meet the condition |
| [AS](https://www.w3schools.com/sql/sql_ref_as.asp) | Renames a column or table with an alias |
| [ASC](https://www.w3schools.com/sql/sql_ref_asc.asp) | Sorts the result set in ascending order |
| [BACKUP DATABASE](https://www.w3schools.com/sql/sql_ref_backup_database.asp) | Creates a back up of an existing database |
| [BETWEEN](https://www.w3schools.com/sql/sql_ref_between.asp) | Selects values within a given range |
| [CASE](https://www.w3schools.com/sql/sql_ref_case.asp) | Creates different outputs based on conditions |
| [CHECK](https://www.w3schools.com/sql/sql_ref_check.asp) | A constraint that limits the value that can be placed in a column |
| [COLUMN](https://www.w3schools.com/sql/sql_ref_column.asp) | Changes the data type of a column or deletes a column in a table |
| [CONSTRAINT](https://www.w3schools.com/sql/sql_ref_constraint.asp) | Adds or deletes a constraint |
| [CREATE](https://www.w3schools.com/sql/sql_ref_create.asp) | Creates a database, index, view, table, or procedure |
| [CREATE DATABASE](https://www.w3schools.com/sql/sql_ref_create_database.asp) | Creates a new SQL database |
| [CREATE INDEX](https://www.w3schools.com/sql/sql_ref_create_index.asp) | Creates an index on a table (allows duplicate values) |
| [CREATE OR REPLACE VIEW](https://www.w3schools.com/sql/sql_ref_create_or_replace_view.asp) | Updates a view |
| [CREATE TABLE](https://www.w3schools.com/sql/sql_ref_create_table.asp) | Creates a new table in the database |
| [CREATE PROCEDURE](https://www.w3schools.com/sql/sql_ref_create_procedure.asp) | Creates a stored procedure |
| [CREATE UNIQUE INDEX](https://www.w3schools.com/sql/sql_ref_create_unique_index.asp) | Creates a unique index on a table (no duplicate values) |
| [CREATE VIEW](https://www.w3schools.com/sql/sql_ref_create_view.asp) | Creates a view based on the result set of a SELECT statement |
| [DATABASE](https://www.w3schools.com/sql/sql_ref_database.asp) | Creates or deletes an SQL database |
| [DEFAULT](https://www.w3schools.com/sql/sql_ref_default.asp) | A constraint that provides a default value for a column |
| [DELETE](https://www.w3schools.com/sql/sql_ref_delete.asp) | Deletes rows from a table |
| [DESC](https://www.w3schools.com/sql/sql_ref_desc.asp) | Sorts the result set in descending order |
| [DISTINCT](https://www.w3schools.com/sql/sql_ref_distinct.asp) | Selects only distinct (different) values |
| [DROP](https://www.w3schools.com/sql/sql_ref_drop.asp) | Deletes a column, constraint, database, index, table, or view |
| [DROP COLUMN](https://www.w3schools.com/sql/sql_ref_drop_column.asp) | Deletes a column in a table |
| [DROP CONSTRAINT](https://www.w3schools.com/sql/sql_ref_drop_constraint.asp) | Deletes a UNIQUE, PRIMARY KEY, FOREIGN KEY, or CHECK constraint |
| [DROP DATABASE](https://www.w3schools.com/sql/sql_ref_drop_database.asp) | Deletes an existing SQL database |
| [DROP DEFAULT](https://www.w3schools.com/sql/sql_ref_drop_default.asp) | Deletes a DEFAULT constraint |
| [DROP INDEX](https://www.w3schools.com/sql/sql_ref_drop_index.asp) | Deletes an index in a table |
| [DROP TABLE](https://www.w3schools.com/sql/sql_ref_drop_table.asp) | Deletes an existing table in the database |
| [DROP VIEW](https://www.w3schools.com/sql/sql_ref_drop_view.asp) | Deletes a view |
| [EXEC](https://www.w3schools.com/sql/sql_ref_exec.asp) | Executes a stored procedure |
| [EXISTS](https://www.w3schools.com/sql/sql_ref_exists.asp) | Tests for the existence of any record in a subquery |
| [FOREIGN KEY](https://www.w3schools.com/sql/sql_ref_foreign_key.asp) | A constraint that is a key used to link two tables together |
| [FROM](https://www.w3schools.com/sql/sql_ref_from.asp) | Specifies which table to select or delete data from |
| [FULL OUTER JOIN](https://www.w3schools.com/sql/sql_ref_full_outer_join.asp) | Returns all rows when there is a match in either left table or right table |
| [GROUP BY](https://www.w3schools.com/sql/sql_ref_group_by.asp) | Groups the result set (used with aggregate functions: COUNT, MAX, MIN, SUM, AVG) |
| [HAVING](https://www.w3schools.com/sql/sql_ref_having.asp) | Used instead of WHERE with aggregate functions |
| [IN](https://www.w3schools.com/sql/sql_ref_in.asp) | Allows you to specify multiple values in a WHERE clause |
| [INDEX](https://www.w3schools.com/sql/sql_ref_index.asp) | Creates or deletes an index in a table |
| [INNER JOIN](https://www.w3schools.com/sql/sql_ref_inner_join.asp) | Returns rows that have matching values in both tables |
| [INSERT INTO](https://www.w3schools.com/sql/sql_ref_insert_into.asp) | Inserts new rows in a table |
| [INSERT INTO SELECT](https://www.w3schools.com/sql/sql_ref_insert_into_select.asp) | Copies data from one table into another table |
| [IS NULL](https://www.w3schools.com/sql/sql_ref_is_null.asp) | Tests for empty values |
| [IS NOT NULL](https://www.w3schools.com/sql/sql_ref_is_not_null.asp) | Tests for non-empty values |
| [JOIN](https://www.w3schools.com/sql/sql_ref_join.asp) | Joins tables |
| [LEFT JOIN](https://www.w3schools.com/sql/sql_ref_left_join.asp) | Returns all rows from the left table, and the matching rows from the right table |
| [LIKE](https://www.w3schools.com/sql/sql_ref_like.asp) | Searches for a specified pattern in a column |
| [LIMIT](https://www.w3schools.com/sql/sql_ref_limit.asp) | Specifies the number of records to return in the result set |
| [NOT](https://www.w3schools.com/sql/sql_ref_not.asp) | Only includes rows where a condition is not true |
| [NOT NULL](https://www.w3schools.com/sql/sql_ref_not_null.asp) | A constraint that enforces a column to not accept NULL values |
| [OR](https://www.w3schools.com/sql/sql_ref_or.asp) | Includes rows where either condition is true |
| [ORDER BY](https://www.w3schools.com/sql/sql_ref_order_by.asp) | Sorts the result set in ascending or descending order |
| [OUTER JOIN](https://www.w3schools.com/sql/sql_ref_outer_join.asp) | Returns all rows when there is a match in either left table or right table |
| [PRIMARY KEY](https://www.w3schools.com/sql/sql_ref_primary_key.asp) | A constraint that uniquely identifies each record in a database table |
| [PROCEDURE](https://www.w3schools.com/sql/sql_ref_procedure.asp) | A stored procedure |
| [RIGHT JOIN](https://www.w3schools.com/sql/sql_ref_right_join.asp) | Returns all rows from the right table, and the matching rows from the left table |
| [ROWNUM](https://www.w3schools.com/sql/sql_ref_rownum.asp) | Specifies the number of records to return in the result set |
| [SELECT](https://www.w3schools.com/sql/sql_ref_select.asp) | Selects data from a database |
| [SELECT DISTINCT](https://www.w3schools.com/sql/sql_ref_select_distinct.asp) | Selects only distinct (different) values |
| [SELECT INTO](https://www.w3schools.com/sql/sql_ref_select_into.asp) | Copies data from one table into a new table |
| [SELECT TOP](https://www.w3schools.com/sql/sql_ref_select_top.asp) | Specifies the number of records to return in the result set |
| [SET](https://www.w3schools.com/sql/sql_ref_set.asp) | Specifies which columns and values that should be updated in a table |
| [TABLE](https://www.w3schools.com/sql/sql_ref_table.asp) | Creates a table, or adds, deletes, or modifies columns in a table, or deletes a table or data inside a table |
| [TOP](https://www.w3schools.com/sql/sql_ref_top.asp) | Specifies the number of records to return in the result set |
| [TRUNCATE TABLE](https://www.w3schools.com/sql/sql_ref_truncate_table.asp) | Deletes the data inside a table, but not the table itself |
| [UNION](https://www.w3schools.com/sql/sql_ref_union.asp) | Combines the result set of two or more SELECT statements (only distinct values) |
| [UNION ALL](https://www.w3schools.com/sql/sql_ref_union_all.asp) | Combines the result set of two or more SELECT statements (allows duplicate values) |
| [UNIQUE](https://www.w3schools.com/sql/sql_ref_unique.asp) | A constraint that ensures that all values in a column are unique |
| [UPDATE](https://www.w3schools.com/sql/sql_ref_update.asp) | Updates existing rows in a table |
| [VALUES](https://www.w3schools.com/sql/sql_ref_values.asp) | Specifies the values of an INSERT INTO statement |
| [VIEW](https://www.w3schools.com/sql/sql_ref_view.asp) | Creates, updates, or deletes a view |
| [WHERE](https://www.w3schools.com/sql/sql_ref_where.asp) | Filters a result set to include only records that fulfill a specified condition |