The SQL CREATE DATABASE Statement

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

Syntax

CREATE DATABASE *databasename*;

CREATE DATABASE Example

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

Example

CREATE DATABASE testDB;

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

The SQL DROP DATABASE Statement

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

Syntax

DROP DATABASE *databasename*;

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

DROP DATABASE Example

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

Example

DROP DATABASE testDB;

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

The SQL CREATE TABLE Statement

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

Syntax

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

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

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

**Tip:** For an overview of the available data types, go to our complete [Data Types Reference](https://www.w3schools.com/sql/sql_datatypes.asp).

SQL CREATE TABLE Example

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

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_create_table)

The PersonID column is of type int and will hold an integer.

The LastName, FirstName, Address, and City columns are of type varchar and will hold characters, and the maximum length for these fields is 255 characters.

The empty "Persons" table will now look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PersonID** | **LastName** | **FirstName** | **Address** | **City** |
|  |  |  |  |  |

**Tip:** The empty "Persons" table can now be filled with data with the SQL [INSERT INTO](https://www.w3schools.com/sql/sql_insert.asp) statement.

Create Table Using Another Table

A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement.

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 SQL DROP TABLE Statement

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

Syntax

DROP TABLE *table\_name*;

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

SQL DROP TABLE Example

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

Example

DROP TABLE Shippers;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_drop_table)

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

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

ALTER TABLE *table\_name*  
ADD *column\_name datatype*;

ALTER TABLE - DROP COLUMN

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*;

ALTER TABLE - ALTER/MODIFY COLUMN

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*;

SQL ALTER TABLE Example

Look at the "Persons" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Now we want to add a column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

ALTER TABLE Persons  
ADD DateOfBirth date;

Notice that the new column, "DateOfBirth", is of type date and is going to hold a date. The data type specifies what type of data the column can hold. For a complete reference of all the data types available in MS Access, MySQL, and SQL Server, go to our complete [Data Types reference](https://www.w3schools.com/sql/sql_datatypes.asp).

The "Persons" table will now look like this:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **LastName** | **FirstName** | **Address** | **City** | **DateOfBirth** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |  |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |  |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |  |

Change Data Type Example

Now we want to change the data type of the column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

ALTER TABLE Persons  
ALTER COLUMN DateOfBirth year;

Notice that the "DateOfBirth" column is now of type year and is going to hold a year in a two- or four-digit format.

DROP COLUMN Example

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

We use the following SQL statement:

ALTER TABLE Persons  
DROP COLUMN DateOfBirth;

The "Persons" table will now look like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

## 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) - Uniquely identifies a row/record in another table
* [**CHECK**](https://www.w3schools.com/sql/sql_check.asp) - Ensures that all values in a column satisfies a specific condition
* [**DEFAULT**](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column when no value is specified
* [**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.
* The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values:
* Example
* CREATE TABLE Persons (  
      ID int NOT NULL,  
      LastName varchar(255) NOT NULL,  
      FirstName varchar(255) NOT NULL,  
      Age int  
  );
* [Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_create_constraint_not_null)
* **Tip:** If the table has already been created, you can add a NOT NULL constraint to a column with the [ALTER TABLE](https://www.w3schools.com/sql/sql_alter.asp) statement.
* 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 database table.
* Primary keys must contain UNIQUE values, and cannot contain NULL values.
* A table can have only one primary key, which may consist of single or multiple 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 the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created).
* 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
* A FOREIGN KEY is a key used to link two tables together.
* A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
* The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.
* Look at the following two tables:
* "Persons" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **PersonID** | **LastName** | **FirstName** | **Age** |
| 1 | Hansen | Ola | 30 |
| 2 | Svendson | Tove | 23 |
| 3 | Pettersen | Kari | 20 |

* "Orders" table:

|  |  |  |
| --- | --- | --- |
| **OrderID** | **OrderNumber** | **PersonID** |
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 2 |
| 4 | 24562 | 1 |

* Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.
* The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.
* The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.
* The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
* The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.
* SQL FOREIGN KEY 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;
* 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 on CREATE TABLE
* The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that you can not have any person below 18 years:
* **MySQL:**
* CREATE TABLE Persons (  
      ID int NOT NULL,  
      LastName varchar(255) NOT NULL,  
      FirstName varchar(255),  
      Age int,  
      CHECK (Age>=18)  
  );
* **SQL Server / Oracle / MS Access:**
* CREATE TABLE Persons (  
      ID int NOT NULL,  
      LastName varchar(255) NOT NULL,  
      FirstName varchar(255),  
      Age int CHECK (Age>=18)  
  );
* To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:
* **MySQL / SQL Server / Oracle / MS Access:**
* CREATE TABLE Persons (  
      ID int NOT NULL,  
      LastName varchar(255) NOT NULL,  
      FirstName varchar(255),  
      Age int,  
      City varchar(255),  
      CONSTRAINT CHK\_Person CHECK (Age>=18 AND City='Sandnes')  
  );
* SQL CHECK on ALTER TABLE
* To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:
* **MySQL / SQL Server / Oracle / MS Access:**
* ALTER TABLE Persons  
  ADD CHECK (Age>=18);
* To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:
* **MySQL / SQL Server / Oracle / MS Access:**
* ALTER TABLE Persons  
  ADD CONSTRAINT CHK\_PersonAge CHECK (Age>=18 AND City='Sandnes');
* DROP a CHECK Constraint
* To drop a CHECK constraint, use the following SQL:
* **SQL Server / Oracle / MS Access:**
* ALTER TABLE Persons  
  DROP CONSTRAINT CHK\_PersonAge;
* **MySQL:**
* ALTER TABLE Persons  
  DROP CHECK CHK\_PersonAge;
* SQL DEFAULT Constraint
* The DEFAULT constraint is used to provide a default value for a column.
* The default value will be added to all new records IF no other value is specified.
* SQL DEFAULT on CREATE TABLE
* The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:
* **My SQL / SQL Server / Oracle / MS Access:**
* CREATE TABLE Persons (  
      ID int NOT NULL,  
      LastName varchar(255) NOT NULL,  
      FirstName varchar(255),  
      Age int,  
      City varchar(255) DEFAULT 'Sandnes'  
  );
* The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE():
* CREATE TABLE Orders (  
      ID int NOT NULL,  
      OrderNumber int NOT NULL,  
      OrderDate date DEFAULT GETDATE()  
  );
* SQL DEFAULT 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';
* 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;
* SQL CREATE INDEX Statement
* The CREATE INDEX statement is used to create indexes in tables.
* Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.
* **Note:** Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.
* CREATE INDEX Syntax
* Creates an index on a table. Duplicate values are allowed:
* CREATE INDEX *index\_name*  
  ON *table\_name* (*column1*, *column2*, ...);
* CREATE UNIQUE INDEX Syntax
* Creates a unique index on a table. Duplicate values are not allowed:
* CREATE UNIQUE INDEX *index\_name*  
  ON *table\_name* (*column1*, *column2*, ...);
* **Note:** The syntax for creating indexes varies among different databases. Therefore: Check the syntax for creating indexes in your database.
* CREATE INDEX Example
* The SQL statement below creates an index named "idx\_lastname" on the "LastName" column in the "Persons" table:
* CREATE INDEX idx\_lastname  
  ON Persons (LastName);
* If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:
* CREATE INDEX idx\_pname  
  ON Persons (LastName, FirstName);
* DROP INDEX Statement
* The DROP INDEX statement is used to delete an index in a table.
* **MS Access:**
* DROP INDEX *index\_name* ON *table\_name*;
* **SQL Server:**
* DROP INDEX *table\_name*.*index\_name*;
* **DB2/Oracle:**
* DROP INDEX *index\_name*;
* **MySQL:**
* ALTER TABLE *table\_name*DROP INDEX *index\_name*;
* 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 "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),  
      Age int,  
      PRIMARY KEY (ID)  
  );
* MySQL uses the AUTO\_INCREMENT keyword to perform an auto-increment feature.
* By default, the starting value for AUTO\_INCREMENT is 1, and it will increment by 1 for each new record.
* To let the AUTO\_INCREMENT sequence start with another value, use the following SQL statement:
* ALTER TABLE Persons AUTO\_INCREMENT=100;
* To insert a new record into the "Persons" table, we will NOT have to specify a value for the "ID" column (a unique value will be added automatically):
* INSERT INTO Persons (FirstName,LastName)  
  VALUES ('Lars','Monsen');
* The SQL statement above would insert a new record into the "Persons" table. The "ID" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".
* Syntax for SQL Server
* The following SQL statement defines the "ID" column to be an auto-increment primary key field in the "Persons" table:
* CREATE TABLE Persons (  
      ID int IDENTITY(1,1) PRIMARY KEY,  
      LastName varchar(255) NOT NULL,  
      FirstName varchar(255),  
      Age int  
  );
* The MS SQL Server uses the IDENTITY keyword to perform an auto-increment feature.
* In the example above, the starting value for IDENTITY is 1, and it will increment by 1 for each new record.
* **Tip:** To specify that the "ID" column should start at value 10 and increment by 5, change it to IDENTITY(10,5).
* To insert a new record into the "Persons" table, we will NOT have to specify a value for the "ID" column (a unique value will be added automatically):
* INSERT INTO Persons (FirstName,LastName)  
  VALUES ('Lars','Monsen');
* The SQL statement above would insert a new record into the "Persons" table. The "ID" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".
* Syntax for Access
* The following SQL statement defines the "ID" column to be an auto-increment primary key field in the "Persons" table:
* CREATE TABLE Persons (  
      ID Integer PRIMARY KEY AUTOINCREMENT,  
      LastName varchar(255) NOT NULL,  
      FirstName varchar(255),  
      Age int  
  );
* The MS Access uses the AUTOINCREMENT keyword to perform an auto-increment feature.
* By default, the starting value for AUTOINCREMENT is 1, and it will increment by 1 for each new record.
* **Tip:** To specify that the "ID" column should start at value 10 and increment by 5, change the autoincrement to AUTOINCREMENT(10,5).
* To insert a new record into the "Persons" table, we will NOT have to specify a value for the "ID" column (a unique value will be added automatically):
* INSERT INTO Persons (FirstName,LastName)  
  VALUES ('Lars','Monsen');
* The SQL statement above would insert a new record into the "Persons" table. The "P\_Id" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".
* Syntax for Oracle
* In Oracle the code is a little bit more tricky.
* You will have to create an auto-increment field with the sequence object (this object generates a number sequence).
* Use the following CREATE SEQUENCE syntax:
* CREATE SEQUENCE seq\_person  
  MINVALUE 1  
  START WITH 1  
  INCREMENT BY 1  
  CACHE 10;
* The code above creates a sequence object called seq\_person, that starts with 1 and will increment by 1. It will also cache up to 10 values for performance. The cache option specifies how many sequence values will be stored in memory for faster access.
* To insert a new record into the "Persons" table, we will have to use the nextval function (this function retrieves the next value from seq\_person sequence):
* INSERT INTO Persons (ID,FirstName,LastName)  
  VALUES (seq\_person.nextval,'Lars','Monsen');
* The SQL statement above would insert a new record into the "Persons" table. The "ID" column would be assigned the next number from the seq\_person sequence. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

## SQL Dates

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

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

## SQL Date Data Types

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

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

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

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

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

## SQL Working with Dates

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

Assume we have the following "Orders" table:

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

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

We use the following SELECT statement:

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

The result-set will look like this:

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

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

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

If we use the same SELECT statement as above:

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

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

**Tip:** To keep your queries simple and easy to maintain, do not allow time components in your dates!

SQL CREATE VIEW Statement

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

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

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

CREATE VIEW Syntax

CREATE VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

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

SQL CREATE VIEW Examples

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

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

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

Then, we can query the view as follows:

SELECT \* FROM [Current Product List];

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

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

We can query the view above as follows:

SELECT \* FROM [Products Above Average Price];

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

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

We can query the view above as follows:

SELECT \* FROM [Category Sales For 1997];

We can also add a condition to the query. Let's see the total sale only for the category "Beverages":

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

SQL Updating a View

You can update a view by using the following syntax:

SQL CREATE OR REPLACE VIEW Syntax

CREATE OR REPLACE VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

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

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

SQL Dropping a View

You can delete a view with the DROP VIEW command.

SQL DROP VIEW Syntax

DROP VIEW view\_name;

SQL Injection

SQL injection is a code injection technique that might destroy your database.

SQL injection is one of the most common web hacking techniques.

SQL injection is the placement of malicious code in SQL statements, via web page input.

SQL in Web Pages

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will **unknowingly** run on your database.

Look at the following example which creates a SELECT statement by adding a variable (txtUserId) to a select string. The variable is fetched from user input (getRequestString):

Example

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

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

SQL Injection Based on 1=1 is Always True

Look at the example above again. The original purpose of the code was to create an SQL statement to select a user, with a given user id.

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

UserId: 

Then, the SQL statement will look like this:

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

The SQL above is valid and will return ALL rows from the "Users" table, since **OR 1=1** is always TRUE.

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

The SQL statement above is much the same as this:

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

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

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

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

Username:  


Password:  


Example

uName = getRequestString("username");  
uPass = getRequestString("userpassword");  
  
sql = 'SELECT \* FROM Users WHERE Name ="' + uName + '" AND Pass ="' + uPass + '"'

Result

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

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

User Name:  


Password:  


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

Result

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

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

SQL Injection Based on Batched SQL Statements

Most databases support batched SQL statement.

A batch of SQL statements is a group of two or more SQL statements, separated by semicolons.

The SQL statement below will return all rows from the "Users" table, then delete the "Suppliers" table.

Example

SELECT \* FROM Users; DROP TABLE Suppliers

Look at the following example:

Example

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

And the following input:

User id: 

The valid SQL statement would look like this:

Result

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

Use SQL Parameters for Protection

To protect a web site from SQL injection, you can use SQL parameters.

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

ASP.NET Razor Example

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

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

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

Another Example

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

Examples

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

SELECT STATEMENT IN ASP.NET:

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

INSERT INTO STATEMENT IN ASP.NET:

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

INSERT INTO STATEMENT IN PHP:

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

SQL Hosting

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

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

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

MS SQL Server

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

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

Oracle

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

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

MySQL

MySQL is also a popular database software for web sites.

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

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

Access

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

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

## Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

In this tutorial we will use the well-known Northwind sample database (included in MS Access and MS SQL Server).

Below is a selection from the "Customers" table:

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

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

## SQL Statements

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

The following SQL statement selects all the records in the "Customers" table:

### Example

SELECT \* FROM Customers;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all)

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

## Keep in Mind That...

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

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

## Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement.

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

In this tutorial, we will use semicolon at the end of each SQL statement.

## Some of The Most Important SQL Commands

* **SELECT** - extracts data from a database
* **UPDATE** - updates data in a database
* **DELETE** - deletes data from a database
* **INSERT INTO** - inserts new data into a database
* **CREATE DATABASE** - creates a new database
* **ALTER DATABASE** - modifies a database
* **CREATE TABLE** - creates a new table
* **ALTER TABLE** - modifies a table
* **DROP TABLE** - deletes a table
* **CREATE INDEX** - creates an index (search key)
* **DROP INDEX** - deletes an index
* The SQL SELECT Statement
* The SELECT statement is used to select data from a database.
* The data returned is stored in a result table, called the result-set.
* SELECT Syntax
* SELECT *column1*,*column2, ...*  
  FROM *table\_name*;
* Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:
* SELECT \* FROM *table\_name*;
* Demo Database
* Below is a selection from the "Customers" table in the Northwind sample database:

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

* SELECT Column Example
* The following SQL statement selects the "CustomerName" and "City" columns from the "Customers" table:
* Example
* SELECT CustomerName, City FROM Customers;
* [Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_columns)
* SELECT \* Example
* The following SQL statement selects all the columns from the "Customers" table:
* Example
* SELECT \* FROM Customers;
* The SQL SELECT DISTINCT Statement
* The SELECT DISTINCT statement is used to return only distinct (different) values.
* Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.
* The SELECT DISTINCT statement is used to return only distinct (different) values.
* SELECT DISTINCT Syntax
* SELECT DISTINCT *column1*,*column2, ...*  
  FROM *table\_name*;
* Demo Database
* Below is a selection from the "Customers" table in the Northwind sample database:

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

* SELECT Example
* The following SQL statement selects all (and duplicate) values from the "Country" column in the "Customers" table:
* Example
* SELECT Country FROM Customers;
* [Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_no_distinct)
* Now, let us use the DISTINCT keyword with the above SELECT statement and see the result.
* SELECT DISTINCT Examples
* The following SQL statement selects only the DISTINCT values from the "Country" column in the "Customers" table:
* Example
* SELECT DISTINCT Country FROM Customers;
* [Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_distinct)
* The following SQL statement lists the number of different (distinct) customer countries:
* Example
* SELECT COUNT(DISTINCT Country) FROM Customers;
* [Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_distinct2)
* **Note: The example above will not work in Firefox and Microsoft Edge!** Because COUNT(DISTINCT *column\_name*) is not supported in Microsoft Access databases. Firefox and Microsoft Edge are using Microsoft Access in our examples.
* Here is the workaround for MS Access:
* Example
* SELECT Count(\*) AS DistinctCountries  
  FROM (SELECT DISTINCT Country FROM Customers);
* The SQL WHERE Clause
* The WHERE clause is used to filter records.
* The WHERE clause is used to extract only those records that fulfill a specified condition.
* WHERE Syntax
* SELECT *column1*,*column2, ...*  
  FROM *table\_name*  
  WHERE *condition*;
* **Note:** The WHERE clause is not only used in SELECT statement, it is also used in UPDATE, DELETE statement, etc.!
* Demo Database
* Below is a selection from the "Customers" table in the Northwind sample database:

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

* WHERE Clause Example
* The following SQL statement selects all the customers from the country "Mexico", in the "Customers" table:
* Example
* SELECT \* FROM Customers  
  WHERE Country='Mexico';
* [Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_where)
* Text Fields vs. Numeric Fields
* SQL requires single quotes around text values (most database systems will also allow double quotes).
* However, numeric fields should not be enclosed in quotes:
* Example
* SELECT \* FROM Customers  
  WHERE CustomerID=1;
* [Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_where_number)
* 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 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 is TRUE.
* The OR operator displays a record if any of the conditions separated by OR is TRUE.

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

### AND Syntax

SELECT column1, column2, ...  
FROM table\_name  
WHERE condition1 AND condition2 AND condition3 ...;

### OR Syntax

SELECT column1, column2, ...  
FROM table\_name  
WHERE condition1 OR condition2 OR condition3 ...;

### NOT Syntax

SELECT column1, column2, ...  
FROM table\_name  
WHERE NOT condition;

## Demo Database

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

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

## AND Example

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

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_where_and)

## OR Example

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

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_where_or)

## NOT Example

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

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_where_not)

## Combining AND, OR and NOT

You can also combine the AND, OR and NOT operators.

The following SQL statement selects all fields from "Customers" where country is "Germany" AND city must be "Berlin" OR "München" (use parenthesis to form complex expressions):

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_where_and_or)

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

### Example

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

The SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

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

ORDER BY Syntax

SELECT *column1*,*column2, ...*  
FROM *table\_name*  
ORDER BY *column1, column2, ...*ASC|DESC;

Demo Database

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

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

ORDER BY Example

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

Example

SELECT \* FROM Customers  
ORDER BY Country;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_orderby)

ORDER BY DESC Example

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

Example

SELECT \* FROM Customers  
ORDER BY Country DESC;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_orderby_desc)

ORDER BY Several Columns Example

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

Example

SELECT \* FROM Customers  
ORDER BY Country, CustomerName;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_orderby2)

ORDER BY Several Columns Example 2

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

Example

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

The SQL INSERT INTO Statement

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

INSERT INTO Syntax

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

The first way specifies both the column names and the values to be inserted:

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

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

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

Demo Database

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 89 | White Clover Markets | Karl Jablonski | 305 - 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |

INSERT INTO Example

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

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_insert_colname)

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

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

**Did you notice that we did not insert any number into the CustomerID field?**  
The CustomerID column is an [auto-increment](https://www.w3schools.com/sql/sql_autoincrement.asp) field and will be generated automatically when a new record is inserted into the table.

Insert Data Only in Specified Columns

It is also possible to only insert data in specific columns.

The following SQL statement will insert a new record, but only insert data in the "CustomerName", "City", and "Country" columns (CustomerID will be updated automatically):

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_insert_cols)

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 89 | White Clover Markets | Karl Jablonski | 305 - 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |
| 92 | null | null | Stavanger | null | 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:** It is very important to understand that a NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

How to Test for NULL Values?

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

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

IS NULL Syntax

SELECT *column\_names*FROM *table\_name*  
WHERE *column\_name* IS NULL;

IS NOT NULL Syntax

SELECT *column\_names*FROM *table\_name*  
WHERE *column\_name* IS NOT NULL;

Demo Database

Assume we have the following "Persons" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Doe | John | 542 W. 27th Street | New York |
| 2 | Bloggs | Joe |  | London |
| 3 | Roe | Jane |  | New York |
| 4 | Smith | John | 110 Bishopsgate | London |

Suppose that the "Address" column in the "Persons" table is optional. If a record is inserted with no value for "Address", the "Address" column will be saved with a NULL value.

The IS NULL Operator

The following SQL statement uses the IS NULL operator to list all persons that have no address:

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

The result-set will look like this:

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **Address** |
| Bloggs | Joe |  |
| Roe | Jane |  |

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

The IS NOT NULL Operator

The following SQL statement uses the IS NOT NULL operator to list all persons that do have an address:

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

The result-set will look like this:

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **Address** |
| Doe | John | 542 W. 27th Street |
| Smith | John | 110 Bishopsgate |

The SQL UPDATE Statement

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

UPDATE Syntax

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

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

Demo Database

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

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

UPDATE Table

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

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_update_2)

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

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

UPDATE Multiple Records

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

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

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_update_3)

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

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

Update Warning!

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

Example

UPDATE Customers  
SET ContactName='Juan';

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_update_4)

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

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

The SQL DELETE Statement

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

DELETE Syntax

DELETE FROM *table\_name*  
WHERE *condition*;

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

Demo Database

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

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

SQL DELETE Example

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

Example

DELETE FROM Customers  
WHERE CustomerName='Alfreds Futterkiste';

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_delete)

The "Customers" table will now look like this:

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

Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

DELETE FROM *table\_name*;

or:

DELETE \* FROM *table\_name*;

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 on 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 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 Syntax:**

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

Demo Database

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

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

SQL TOP, LIMIT and ROWNUM Examples

The following SQL statement selects the first three records from the "Customers" table:

Example

SELECT TOP 3 \* FROM Customers;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_top&ss=-1)

The following SQL statement shows the equivalent example using the LIMIT clause:

Example

SELECT \* FROM Customers  
LIMIT 3;

[Try it Yourself »](https://www.w3schools.com/sql/trymysql.asp?filename=trysql_select_limit)

The following SQL statement shows the equivalent example using ROWNUM:

Example

SELECT \* FROM Customers  
WHERE ROWNUM <= 3;

SQL TOP PERCENT Example

The following SQL statement selects the first 50% of the records from the "Customers" table:

Example

SELECT TOP 50 PERCENT \* FROM Customers;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_top_percent&ss=-1)

ADD a WHERE CLAUSE

The following SQL statement selects the first three records from the "Customers" table, where the country is "Germany":

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_top_where&ss=-1)

The following SQL statement shows the equivalent example using the LIMIT clause:

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trymysql.asp?filename=trysql_select_limit_where)

The following SQL statement shows the equivalent example using ROWNUM:

Example

SELECT \* FROM Customers  
WHERE Country='Germany' AND ROWNUM <= 3;

The SQL MIN() and MAX() Functions

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

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

MIN() Syntax

SELECT MIN(*column\_name*)  
FROM *table\_name*  
WHERE *condition*;

MAX() Syntax

SELECT MAX(*column\_name*)  
FROM *table\_name*  
WHERE *condition*;

Demo Database

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

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

MIN() Example

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

Example

SELECT MIN(Price) AS SmallestPrice  
FROM Products;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_min)

MAX() Example

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

Example

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 criteria.

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

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

COUNT() Syntax

SELECT COUNT(*column\_name*)  
FROM *table\_name*  
WHERE *condition*;

AVG() Syntax

SELECT AVG(*column\_name*)  
FROM *table\_name*  
WHERE *condition*;

SUM() Syntax

SELECT SUM(*column\_name*)  
FROM *table\_name*  
WHERE *condition*;

Demo Database

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

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

COUNT() Example

The following SQL statement finds the number of products:

Example

SELECT COUNT(ProductID)  
FROM Products;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_count)

AVG() Example

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

Example

SELECT AVG(Price)  
FROM Products;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_avg)

Demo Database

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

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

SUM() Example

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

Example

SELECT SUM(Quantity)  
FROM OrderDetails;

## 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 used in conjunction with the LIKE operator:

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

**Note:** MS Access uses 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.

Here are some examples showing different LIKE operators with '%' and '\_' wildcards:

|  |  |
| --- | --- |
| **LIKE Operator** | **Description** |
| WHERE CustomerName LIKE 'a%' | Finds any values that starts with "a" |
| WHERE CustomerName LIKE '%a' | Finds any values that ends 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 starts with "a" and are at least 3 characters in length |
| WHERE ContactName LIKE 'a%o' | Finds any values that starts with "a" and ends with "o" |

## Demo Database

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

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

## SQL LIKE Examples

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

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_like)

The following SQL statement selects all customers with a CustomerName ending with "a":

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_like_ending)

The following SQL statement selects all customers with a CustomerName that have "or" in any position:

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_like_pattern)

The following SQL statement selects all customers with a CustomerName that have "r" in the second position:

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_like_underscore)

The following SQL statement selects all customers with a CustomerName that starts with "a" and are at least 3 characters in length:

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_like_start_least)

The following SQL statement selects all customers with a ContactName that starts with "a" and ends with "o":

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_like_start_end)

The following SQL statement selects all customers with a CustomerName that does NOT start with "a":

### Example

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

## SQL Wildcard Characters

A wildcard character is used to substitute any other character(s) in a string.

Wildcard characters are used with the [SQL LIKE](https://www.w3schools.com/sql/sql_like.asp) operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

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

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

**Note:** MS Access uses a question mark (?) instead of the underscore (\_).

In MS Access and SQL Server you can also use:

* [charlist] - Defines sets and ranges of characters to match
* [^charlist] or [!charlist] - Defines sets and ranges of characters NOT to match

The wildcards can also be used in combinations!

Here are some examples showing different LIKE operators with '%' and '\_' wildcards:

|  |  |
| --- | --- |
| **LIKE Operator** | **Description** |
| WHERE CustomerName LIKE 'a%' | Finds any values that starts with "a" |
| WHERE CustomerName LIKE '%a' | Finds any values that ends 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 starts with "a" and are at least 3 characters in length |
| WHERE ContactName LIKE 'a%o' | Finds any values that starts with "a" and ends with "o" |

## Demo Database

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

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

## Using the % Wildcard

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

### Example

SELECT \* FROM Customers  
WHERE City LIKE 'ber%';

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_wildcard_percent)

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

### Example

SELECT \* FROM Customers  
WHERE City LIKE '%es%';

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_wildcard_percent_pattern)

## Using the \_ Wildcard

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

### Example

SELECT \* FROM Customers  
WHERE City LIKE '\_erlin';

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_wildcard_underscore)

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

### Example

SELECT \* FROM Customers  
WHERE City LIKE 'L\_n\_on';

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_wildcard_underscore2)

## Using the [charlist] Wildcard

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

### Example

SELECT \* FROM Customers  
WHERE City LIKE '[bsp]%';

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_wildcard_charlist&ss=-1)

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

### Example

SELECT \* FROM Customers  
WHERE City LIKE '[a-c]%';

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_wildcard_charlist2&ss=-1)

## Using the [!charlist] Wildcard

The two following SQL statements select all customers with a City NOT starting with "b", "s", or "p":

### Example

SELECT \* FROM Customers  
WHERE City LIKE '[!bsp]%';

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_wildcard_not_charlist&ss=-1)

Or:

### Example

SELECT \* FROM Customers  
WHERE City NOT LIKE '[bsp]%';

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*);

Demo Database

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

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

IN Operator Examples

The following SQL statement selects all customers that are located in "Germany", "France" and "UK":

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_in)

The following SQL statement selects all customers that are NOT located in "Germany", "France" or "UK":

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_in_not)

The following SQL statement selects all customers that are from the same countries as the suppliers:

Example

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;*

Demo Database

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

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

BETWEEN Example

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

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_between)

NOT BETWEEN Example

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

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_not_between)

BETWEEN with IN Example

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

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_between_in)

BETWEEN Text Values Example

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

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_between_text)

NOT BETWEEN Text Values Example

The following SQL statement selects all products with a ProductName NOT BETWEEN 'Carnarvon Tigers' and 'Mozzarella di Giovanni':

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_not_between_text)

Sample Table

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

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

BETWEEN Dates Example

The following SQL statement selects all orders with an OrderDate BETWEEN '04-July-1996' and '09-July-1996':

Example

SELECT \* FROM Orders  
WHERE OrderDate BETWEEN #07/04/1996# AND #07/09/1996#;

## SQL Aliases

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

Aliases are often used to make column names more readable.

An alias only exists for the duration of the query.

### Alias Column Syntax

SELECT column\_name AS alias\_name  
FROM table\_name;

### Alias Table Syntax

SELECT column\_name(s)  
FROM table\_name AS alias\_name;

## Demo Database

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

Below is a selection from the "Customers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |

And a selection from the "Orders" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10354 | 58 | 8 | 1996-11-14 | 3 |
| 10355 | 4 | 6 | 1996-11-15 | 1 |
| 10356 | 86 | 6 | 1996-11-18 | 2 |

## Alias for Columns Examples

The following SQL statement creates two aliases, one for the CustomerID column and one for the CustomerName column:

### Example

SELECT CustomerID as ID, CustomerName AS Customer  
FROM Customers;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_alias_column0)

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

### Example

SELECT CustomerName AS Customer, ContactName AS [Contact Person]  
FROM Customers;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_alias_column)

The following SQL statement creates an alias named "Address" that combine four columns (Address, PostalCode, City and Country):

### Example

SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ', ' + Country ASAddress  
FROM Customers;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_alias_column2&ss=-1)

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

SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,', ',Country) ASAddress  
FROM Customers;

## Alias for Tables Example

The following SQL statement selects all the orders from the customer with CustomerID=4 (Around the Horn). We use the "Customers" and "Orders" tables, and give them the table aliases of "c" and "o" respectively (Here we use aliases to make the SQL shorter):

### Example

SELECT o.OrderID, o.OrderDate, c.CustomerName  
FROM Customers AS c, Orders AS o  
WHERE c.CustomerName="Around the Horn" AND c.CustomerID=o.CustomerID;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_alias_table)

The following SQL statement is the same as above, but without aliases:

### Example

SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName  
FROM Customers, Orders  
WHERE Customers.CustomerName="Around the Horn" ANDCustomers.CustomerID=Orders.CustomerID;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_alias_no)

Aliases can be useful when:

* There are more than one table involved in a query
* Functions are used in the query
* Column names are big or not very readable
* Two or more columns are combined together

## SQL JOIN

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

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

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

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

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

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

Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_join)

and it will produce something like this:

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

## Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

* **(INNER) JOIN**: Returns records that have matching values in both tables
* **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
* **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
* **FULL (OUTER) JOIN**: Return 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 *table2*ON *table1.column\_name*=*table2.column\_name*;



Demo Database

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

Below is a selection from the "Orders" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

And a selection from the "Customers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

SQL INNER JOIN Example

The following SQL statement selects all orders with customer information:

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_join_inner)

**Note:** The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are records in the "Orders" table that do not have matches in "Customers", these orders will not be shown!

 JOIN Three Tables

The following SQL statement selects all orders with customer and shipper information:

Example

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 matched records from the right table (table2). The result is NULL from the right side, if there is no match.

LEFT JOIN Syntax

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

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



Demo Database

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

Below is a selection from the "Customers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

And a selection from the "Orders" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

SQL LEFT JOIN Example

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

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_join_left)

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

SQL RIGHT JOIN Keyword

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

RIGHT JOIN Syntax

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

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



Demo Database

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

Below is a selection from the "Orders" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

And a selection from the "Employees" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EmployeeID** | **LastName** | **FirstName** | **BirthDate** | **Photo** |
| 1 | Davolio | Nancy | 12/8/1968 | EmpID1.pic |
| 2 | Fuller | Andrew | 2/19/1952 | EmpID2.pic |
| 3 | Leverling | Janet | 8/30/1963 | EmpID3.pic |

SQL RIGHT JOIN Example

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

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_join_right&ss=-1)

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

SQL FULL OUTER JOIN Keyword

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

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

FULL OUTER JOIN Syntax

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



Demo Database

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

Below is a selection from the "Customers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

And a selection from the "Orders" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

SQL FULL OUTER JOIN Example

The following SQL statement selects all customers, and all orders:

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

A selection from the result set may look like this:

|  |  |
| --- | --- |
| **CustomerName** | **OrderID** |
| Alfreds Futterkiste |  |
| Ana Trujillo Emparedados y helados | 10308 |
| Antonio Moreno Taquería | 10365 |
|  | 10382 |
|  | 10351 |

**Note:** The FULL OUTER JOIN keyword returns all the rows from the left table (Customers), and all the rows from the right table (Orders). If there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

SQL Self JOIN

A self JOIN is a regular join, but the table is joined with itself.

Self JOIN Syntax

SELECT *column\_name(s)*  
FROM *table1 T1, table1 T2*  
WHERE *condition*;

Demo Database

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

Below is a selection from the "Customers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

SQL Self JOIN Example

The following SQL statement matches customers that are from the same city:

Example

SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City  
FROM Customers A, Customers B  
WHERE A.CustomerID <> B.CustomerID  
AND A.City = B.City   
ORDER BY A.City;

## The SQL UNION Operator

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

* Each SELECT statement within UNION must have the same number of columns
* The columns must also have similar data types
* The columns in each 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 in the UNION.

## Demo Database

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

Below is a selection from the "Customers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

And a selection from the "Suppliers" table:

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

## SQL UNION Example

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

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_union)

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

## SQL UNION ALL Example

The following SQL statement selects all cities (duplicate values also) from "Customers" and "Suppliers":

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_union_all)

## SQL UNION With WHERE

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

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_union2)

## SQL UNION ALL With WHERE

The following SQL statement selects all German cities (duplicate values also) from "Customers" and "Suppliers":

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_union_all2)

## Another UNION Example

The following SQL statement lists all customers and suppliers:

### Example

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

The SQL GROUP BY Statement

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);*

Demo Database

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

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

SQL GROUP BY Examples

The following SQL statement lists the number of customers in each country:

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_groupby)

The following SQL statement lists the number of customers in each country, sorted high to low:

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_groupby_orderby)

Demo Database

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10248 | 90 | 5 | 1996-07-04 | 3 |
| 10249 | 81 | 6 | 1996-07-05 | 1 |
| 10250 | 34 | 4 | 1996-07-08 | 2 |

And a selection from the "Shippers" table:

|  |  |
| --- | --- |
| **ShipperID** | **ShipperName** |
| 1 | Speedy Express |
| 2 | United Package |
| 3 | Federal Shipping |

GROUP BY With JOIN Example

The following SQL statement lists the number of orders sent by each shipper:

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 could not be used with aggregate functions.

HAVING Syntax

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

Demo Database

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

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

SQL HAVING Examples

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_having)

The following SQL statement lists the number of customers in each country, sorted high to low (Only include countries with more than 5 customers):

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_having_orderby)

Demo Database

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10248 | 90 | 5 | 1996-07-04 | 3 |
| 10249 | 81 | 6 | 1996-07-05 | 1 |
| 10250 | 34 | 4 | 1996-07-08 | 2 |

And a selection from the "Employees" table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EmployeeID** | **LastName** | **FirstName** | **BirthDate** | **Photo** | **Notes** |
| 1 | Davolio | Nancy | 1968-12-08 | EmpID1.pic | Education includes a BA.... |
| 2 | Fuller | Andrew | 1952-02-19 | EmpID2.pic | Andrew received his BTS.... |
| 3 | Leverling | Janet | 1963-08-30 | EmpID3.pic | Janet has a BS degree.... |

More HAVING Examples

The following SQL statement lists the employees that have registered more than 10 orders:

Example

SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders  
FROM (Orders  
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)  
GROUP BY LastName  
HAVING COUNT(Orders.OrderID) > 10;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_having2)

The following SQL statement lists if the employees "Davolio" or "Fuller" have registered more than 25 orders:

Example

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

The SQL EXISTS Operator

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

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

EXISTS Syntax

SELECT *column\_name(s)*  
FROM *table\_name*  
WHERE EXISTS  
(SELECT *column\_name*FROM *table\_name* WHERE *condition*);

Demo Database

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

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

And a selection from the "Suppliers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SupplierID** | **SupplierName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Exotic Liquid | Charlotte Cooper | 49 Gilbert St. | London | EC1 4SD | UK |
| 2 | New Orleans Cajun Delights | Shelley Burke | P.O. Box 78934 | New Orleans | 70117 | USA |
| 3 | Grandma Kelly's Homestead | Regina Murphy | 707 Oxford Rd. | Ann Arbor | 48104 | USA |
| 4 | Tokyo Traders | Yoshi Nagase | 9-8 Sekimai Musashino-shi | Tokyo | 100 | Japan |

SQL EXISTS Examples

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

Example

SELECT SupplierName  
FROM Suppliers  
WHERE EXISTS (SELECT ProductName FROM Products WHERE SupplierId = Suppliers.supplierId AND Price < 20);

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_exists)

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

Example

SELECT SupplierName  
FROM Suppliers  
WHERE EXISTS (SELECT ProductName FROM Products WHERE SupplierId = Suppliers.supplierId AND Price = 22);

The SQL ANY and ALL Operators

The ANY and ALL operators are used with a WHERE or HAVING clause.

The ANY operator returns true if any of the subquery values meet the condition.

The ALL operator returns true if all of the subquery values meet the condition.

ANY Syntax

SELECT *column\_name(s)*  
FROM *table\_name*  
WHERE *column\_name operator* ANY  
(SELECT *column\_name*FROM *table\_name* WHERE *condition*);

ALL Syntax

SELECT *column\_name(s)*  
FROM *table\_name*  
WHERE *column\_name operator* ALL  
(SELECT *column\_name*FROM *table\_name*WHERE *condition*);

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

Demo Database

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

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

And a selection from the "OrderDetails" table:

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

SQL ANY Examples

The ANY operator returns TRUE if any of the subquery values meet the condition.

The following SQL statement returns TRUE and lists the productnames if it finds ANY records in the OrderDetails table that quantity = 10:

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_any&ss=-1)

The following SQL statement returns TRUE and lists the productnames if it finds ANY records in the OrderDetails table that quantity > 99:

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_any2&ss=-1)

SQL ALL Example

The ALL operator returns TRUE if all of the subquery values meet the condition.

The following SQL statement returns TRUE and lists the productnames if ALL the records in the OrderDetails table has quantity = 10:

Example

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

The SQL SELECT INTO Statement

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

SELECT INTO Syntax

Copy all columns into a new table:

SELECT \*  
INTO *newtable* [IN *externaldb*]  
FROM *oldtable*WHERE *condition*;

Copy only some columns into a new table:

SELECT *column1*, *column2*, *column3*, ...  
INTO *newtable* [IN *externaldb*]  
FROM *oldtable*WHERE *condition;*

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

SQL SELECT INTO Examples

The following SQL statement creates a backup copy of Customers:

SELECT \* INTO CustomersBackup2017  
FROM Customers;

The following SQL statement uses the IN clause to copy the table into a new table in another database:

SELECT \* INTO CustomersBackup2017 IN 'Backup.mdb'  
FROM Customers;

The following SQL statement copies only a few columns into a new table:

SELECT CustomerName, ContactName INTO CustomersBackup2017  
FROM Customers;

The following SQL statement copies only the German customers into a new table:

SELECT \* INTO CustomersGermany  
FROM Customers  
WHERE Country = 'Germany';

The following SQL statement copies data from more than one table into a new table:

SELECT Customers.CustomerName, Orders.OrderID  
INTO CustomersOrderBackup2017  
FROM Customers  
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

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

SELECT \* INTO *newtable*  
FROM *oldtable*  
WHERE 1 = 0;

## The SQL INSERT INTO SELECT Statement

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

* INSERT INTO SELECT requires that data types in source and target tables match
* The existing records in the target table are unaffected

### INSERT INTO SELECT Syntax

Copy all columns from one table to another table:

INSERT INTO table2  
SELECT \* FROM table1WHERE condition;

Copy only some columns from one table into another table:

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

## Demo Database

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

Below is a selection from the "Customers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

And a selection from the "Suppliers" table:

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

## SQL INSERT INTO SELECT Examples

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

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_insert_into_select)

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

### Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_insert_into_select2)

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

### Example

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

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

Look at the following "Products" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **ProductName** | **UnitPrice** | **UnitsInStock** | **UnitsOnOrder** |
| 1 | Jarlsberg | 10.45 | 16 | 15 |
| 2 | Mascarpone | 32.56 | 23 |  |
| 3 | Gorgonzola | 15.67 | 9 | 20 |

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

Look at the following SELECT statement:

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

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

Solutions

**MySQL**

The MySQL [IFNULL()](https://www.w3schools.com/sql/func_mysql_ifnull.asp) function lets you return an alternative value if an expression is NULL:

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

or we can use the [COALESCE()](https://www.w3schools.com/sql/func_mysql_coalesce.asp) function, like this:

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

**SQL Server**

The SQL Server [ISNULL()](https://www.w3schools.com/sql/func_sqlserver_isnull.asp) function lets you return an alternative value when an expression is NULL:

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

**MS Access**

The MS Access [IsNull()](https://www.w3schools.com/sql/func_msaccess_isnull.asp) function returns TRUE (-1) if the expression is a null value, otherwise FALSE (0):

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

**Oracle**

The Oracle NVL() function achieves the same result:

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

SQL Comments

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

**Note: The examples in this chapter will not work in Firefox and Microsoft Edge!**

Comments are not supported in Microsoft Access databases. Firefox and Microsoft Edge are using Microsoft Access database in our examples.

Single Line Comments

Single line comments start with --.

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

The following example uses a single-line comment as an explanation:

Example

--Select all:  
SELECT \* FROM Customers;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_comment_single_1)

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

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_comment_single_2)

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

Example

--SELECT \* FROM Customers;  
SELECT \* FROM Products;

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_comment_single_3)

Multi-line Comments

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

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

The following example uses a multi-line comment as an explanation:

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_comment_multi_1)

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

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_comment_multi_2)

To ignore just a part of a statement, also use the /\* \*/ comment.

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

Example

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

[Try it Yourself »](https://www.w3schools.com/sql/trysql.asp?filename=trysql_comment_part_1)

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

Example

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

MySQL String Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [ASCII](https://www.w3schools.com/sql/func_mysql_ascii.asp) | Returns the number code that represents the specific character |
| [CHAR\_LENGTH](https://www.w3schools.com/sql/func_mysql_char_length.asp) | Returns the length of the specified string (in characters) |
| [CHARACTER\_LENGTH](https://www.w3schools.com/sql/func_mysql_character_length.asp) | Returns the length of the specified string (in characters) |
| [CONCAT](https://www.w3schools.com/sql/func_mysql_concat.asp) | Concatenates two or more expressions together |
| [CONCAT\_WS](https://www.w3schools.com/sql/func_mysql_concat_ws.asp) | Concatenates two or more expressions together and adds a separator between them |
| [FIELD](https://www.w3schools.com/sql/func_mysql_field.asp) | Returns the position of a value in a list of values |
| [FIND\_IN\_SET](https://www.w3schools.com/sql/func_mysql_find_in_set.asp) | Returns the position of a string in a string list |
| [FORMAT](https://www.w3schools.com/sql/func_mysql_format.asp) | Formats a number as a format of "#,###.##", rounding it to a certain number of decimal places |
| [INSERT](https://www.w3schools.com/sql/func_mysql_insert.asp) | Inserts a substring into a string at a specified position for a certain number of characters |
| [INSTR](https://www.w3schools.com/sql/func_mysql_instr.asp) | Returns the position of the first occurrence of a string in another string |
| [LCASE](https://www.w3schools.com/sql/func_mysql_lcase.asp) | Converts a string to lower-case |
| [LEFT](https://www.w3schools.com/sql/func_mysql_left.asp) | Extracts a substring from a string (starting from left) |
| [LENGTH](https://www.w3schools.com/sql/func_mysql_length.asp) | Returns the length of the specified string (in bytes) |
| [LOCATE](https://www.w3schools.com/sql/func_mysql_locate.asp) | Returns the position of the first occurrence of a substring in a string |
| [LOWER](https://www.w3schools.com/sql/func_mysql_lower.asp) | Converts a string to lower-case |
| [LPAD](https://www.w3schools.com/sql/func_mysql_lpad.asp) | Returns a string that is left-padded with a specified string to a certain length |
| [LTRIM](https://www.w3schools.com/sql/func_mysql_ltrim.asp) | Removes leading spaces from a string |
| [MID](https://www.w3schools.com/sql/func_mysql_mid.asp) | Extracts a substring from a string (starting at any position) |
| [POSITION](https://www.w3schools.com/sql/func_mysql_position.asp) | Returns the position of the first occurrence of a substring in a string |
| [REPEAT](https://www.w3schools.com/sql/func_mysql_repeat.asp) | Repeats a string a specified number of times |
| [REPLACE](https://www.w3schools.com/sql/func_mysql_replace.asp) | Replaces all occurrences of a specified string |
| [REVERSE](https://www.w3schools.com/sql/func_mysql_reverse.asp) | Reverses a string and returns the result |
| [RIGHT](https://www.w3schools.com/sql/func_mysql_right.asp) | Extracts a substring from a string (starting from right) |
| [RPAD](https://www.w3schools.com/sql/func_mysql_rpad.asp) | Returns a string that is right-padded with a specified string to a certain length |
| [RTRIM](https://www.w3schools.com/sql/func_mysql_rtrim.asp) | Removes trailing spaces from a string |
| [SPACE](https://www.w3schools.com/sql/func_mysql_space.asp) | Returns a string with a specified number of spaces |
| [STRCMP](https://www.w3schools.com/sql/func_mysql_strcmp.asp) | Tests whether two strings are the same |
| [SUBSTR](https://www.w3schools.com/sql/func_mysql_substr.asp) | Extracts a substring from a string (starting at any position) |
| [SUBSTRING](https://www.w3schools.com/sql/func_mysql_substring.asp) | Extracts a substring from a string (starting at any position) |
| [SUBSTRING\_INDEX](https://www.w3schools.com/sql/func_mysql_substring_index.asp) | Returns the substring of *string* before *number* of occurrences of *delimiter* |
| [TRIM](https://www.w3schools.com/sql/func_mysql_trim.asp) | Removes leading and trailing spaces from a string |
| [UCASE](https://www.w3schools.com/sql/func_mysql_ucase.asp) | Converts a string to upper-case |
| [UPPER](https://www.w3schools.com/sql/func_mysql_upper.asp) | Converts a string to upper-case |

MySQL Numeric Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [ABS](https://www.w3schools.com/sql/func_mysql_abs.asp) | Returns the absolute value of a number |
| [ACOS](https://www.w3schools.com/sql/func_mysql_acos.asp) | Returns the arc cosine of a number |
| [ASIN](https://www.w3schools.com/sql/func_mysql_asin.asp) | Returns the arc sine of a number |
| [ATAN](https://www.w3schools.com/sql/func_mysql_atan.asp) | Returns the arc tangent of a number or the arc tangent of n and m |
| [ATAN2](https://www.w3schools.com/sql/func_mysql_atan2.asp) | Returns the arc tangent of n and m |
| [AVG](https://www.w3schools.com/sql/func_mysql_avg.asp) | Returns the average value of an expression |
| [CEIL](https://www.w3schools.com/sql/func_mysql_ceil.asp) | Returns the smallest integer value that is greater than or equal to a number |
| [CEILING](https://www.w3schools.com/sql/func_mysql_ceiling.asp) | Returns the smallest integer value that is greater than or equal to a number |
| [COS](https://www.w3schools.com/sql/func_mysql_cos.asp) | Returns the cosine of a number |
| [COT](https://www.w3schools.com/sql/func_mysql_cot.asp) | Returns the cotangent of a number |
| [COUNT](https://www.w3schools.com/sql/func_mysql_count.asp) | Returns the number of records in a select query |
| [DEGREES](https://www.w3schools.com/sql/func_mysql_degrees.asp) | Converts a radian value into degrees |
| [DIV](https://www.w3schools.com/sql/func_mysql_div.asp) | Used for integer division |
| [EXP](https://www.w3schools.com/sql/func_mysql_exp.asp) | Returns e raised to the power of number |
| [FLOOR](https://www.w3schools.com/sql/func_mysql_floor.asp) | Returns the largest integer value that is less than or equal to a number |
| [GREATEST](https://www.w3schools.com/sql/func_mysql_greatest.asp) | Returns the greatest value in a list of expressions |
| [LEAST](https://www.w3schools.com/sql/func_mysql_least.asp) | Returns the smallest value in a list of expressions |
| [LN](https://www.w3schools.com/sql/func_mysql_ln.asp) | Returns the natural logarithm of a number |
| [LOG](https://www.w3schools.com/sql/func_mysql_log.asp) | Returns the natural logarithm of a number or the logarithm of a number to a specified base |
| [LOG10](https://www.w3schools.com/sql/func_mysql_log10.asp) | Returns the base-10 logarithm of a number |
| [LOG2](https://www.w3schools.com/sql/func_mysql_log2.asp) | Returns the base-2 logarithm of a number |
| [MAX](https://www.w3schools.com/sql/func_mysql_max.asp) | Returns the maximum value of an expression |
| [MIN](https://www.w3schools.com/sql/func_mysql_min.asp) | Returns the minimum value of an expression |
| [MOD](https://www.w3schools.com/sql/func_mysql_mod.asp) | Returns the remainder of n divided by m |
| [PI](https://www.w3schools.com/sql/func_mysql_pi.asp) | Returns the value of PI displayed with 6 decimal places |
| [POW](https://www.w3schools.com/sql/func_mysql_pow.asp) | Returns m raised to the nth power |
| [POWER](https://www.w3schools.com/sql/func_mysql_power.asp) | Returns m raised to the nth power |
| [RADIANS](https://www.w3schools.com/sql/func_mysql_radians.asp) | Converts a value in degrees to radians |
| [RAND](https://www.w3schools.com/sql/func_mysql_rand.asp) | Returns a random number or a random number within a range |
| [ROUND](https://www.w3schools.com/sql/func_mysql_round.asp) | Returns a number rounded to a certain number of decimal places |
| [SIGN](https://www.w3schools.com/sql/func_mysql_sign.asp) | Returns a value indicating the sign of a number |
| [SIN](https://www.w3schools.com/sql/func_mysql_sin.asp) | Returns the sine of a number |
| [SQRT](https://www.w3schools.com/sql/func_mysql_sqrt.asp) | Returns the square root of a number |
| [SUM](https://www.w3schools.com/sql/func_mysql_sum.asp) | Returns the summed value of an expression |
| [TAN](https://www.w3schools.com/sql/func_mysql_tan.asp) | Returns the tangent of a number |
| [TRUNCATE](https://www.w3schools.com/sql/func_mysql_truncate.asp) | Returns a number truncated to a certain number of decimal places |

MySQL Date Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [ADDDATE](https://www.w3schools.com/sql/func_mysql_adddate.asp) | Returns a date after a certain time/date interval has been added |
| [ADDTIME](https://www.w3schools.com/sql/func_mysql_addtime.asp) | Returns a time/datetime after a certain time interval has been added |
| [CURDATE](https://www.w3schools.com/sql/func_mysql_curdate.asp) | Returns the current date |
| [CURRENT\_DATE](https://www.w3schools.com/sql/func_mysql_current_date.asp) | Returns the current date |
| [CURRENT\_TIME](https://www.w3schools.com/sql/func_mysql_current_time.asp) | Returns the current time |
| [CURRENT\_TIMESTAMP](https://www.w3schools.com/sql/func_mysql_current_timestamp.asp) | Returns the current date and time |
| [CURTIME](https://www.w3schools.com/sql/func_mysql_curtime.asp) | Returns the current time |
| [DATE](https://www.w3schools.com/sql/func_mysql_date.asp) | Extracts the date value from a date or datetime expression |
| [DATEDIFF](https://www.w3schools.com/sql/func_mysql_datediff.asp) | Returns the difference in days between two date values |
| [DATE\_ADD](https://www.w3schools.com/sql/func_mysql_date_add.asp) | Returns a date after a certain time/date interval has been added |
| [DATE\_FORMAT](https://www.w3schools.com/sql/func_mysql_date_format.asp) | Formats a date as specified by a format mask |
| [DATE\_SUB](https://www.w3schools.com/sql/func_mysql_date_sub.asp) | Returns a date after a certain time/date interval has been subtracted |
| [DAY](https://www.w3schools.com/sql/func_mysql_day.asp) | Returns the day portion of a date value |
| [DAYNAME](https://www.w3schools.com/sql/func_mysql_dayname.asp) | Returns the weekday name for a date |
| [DAYOFMONTH](https://www.w3schools.com/sql/func_mysql_dayofmonth.asp) | Returns the day portion of a date value |
| [DAYOFWEEK](https://www.w3schools.com/sql/func_mysql_dayofweek.asp) | Returns the weekday index for a date value |
| [DAYOFYEAR](https://www.w3schools.com/sql/func_mysql_dayofyear.asp) | Returns the day of the year for a date value |
| [EXTRACT](https://www.w3schools.com/sql/func_mysql_extract.asp) | Extracts parts from a date |
| [FROM\_DAYS](https://www.w3schools.com/sql/func_mysql_from_days.asp) | Returns a date value from a numeric representation of the day |
| [HOUR](https://www.w3schools.com/sql/func_mysql_hour.asp) | Returns the hour portion of a date value |
| [LAST\_DAY](https://www.w3schools.com/sql/func_mysql_last_day.asp) | Returns the last day of the month for a given date |
| [LOCALTIME](https://www.w3schools.com/sql/func_mysql_localtime.asp) | Returns the current date and time |
| [LOCALTIMESTAMP](https://www.w3schools.com/sql/func_mysql_localtimestamp.asp) | Returns the current date and time |
| [MAKEDATE](https://www.w3schools.com/sql/func_mysql_makedate.asp) | Returns the date for a certain year and day-of-year value |
| [MAKETIME](https://www.w3schools.com/sql/func_mysql_maketime.asp) | Returns the time for a certain hour, minute, second combination |
| [MICROSECOND](https://www.w3schools.com/sql/func_mysql_microsecond.asp) | Returns the microsecond portion of a date value |
| [MINUTE](https://www.w3schools.com/sql/func_mysql_minute.asp) | Returns the minute portion of a date value |
| [MONTH](https://www.w3schools.com/sql/func_mysql_month.asp) | Returns the month portion of a date value |
| [MONTHNAME](https://www.w3schools.com/sql/func_mysql_monthname.asp) | Returns the full month name for a date |
| [NOW](https://www.w3schools.com/sql/func_mysql_now.asp) | Returns the current date and time |
| [PERIOD\_ADD](https://www.w3schools.com/sql/func_mysql_period_add.asp) | Takes a period and adds a specified number of months to it |
| [PERIOD\_DIFF](https://www.w3schools.com/sql/func_mysql_period_diff.asp) | Returns the difference in months between two periods |
| [QUARTER](https://www.w3schools.com/sql/func_mysql_quarter.asp) | Returns the quarter portion of a date value |
| [SECOND](https://www.w3schools.com/sql/func_mysql_second.asp) | Returns the second portion of a date value |
| [SEC\_TO\_TIME](https://www.w3schools.com/sql/func_mysql_sec_to_time.asp) | Converts numeric seconds into a time value |
| [STR\_TO\_DATE](https://www.w3schools.com/sql/func_mysql_str_to_date.asp) | Takes a string and returns a date specified by a format mask |
| [SUBDATE](https://www.w3schools.com/sql/func_mysql_subdate.asp) | Returns a date after which a certain time/date interval has been subtracted |
| [SUBTIME](https://www.w3schools.com/sql/func_mysql_subtime.asp) | Returns a time/datetime value after a certain time interval has been subtracted |
| [SYSDATE](https://www.w3schools.com/sql/func_mysql_sysdate.asp) | Returns the current date and time |
| [TIME](https://www.w3schools.com/sql/func_mysql_time.asp) | Extracts the time value from a time/datetime expression |
| [TIME\_FORMAT](https://www.w3schools.com/sql/func_mysql_time_format.asp) | Formats a time as specified by a format mask |
| [TIME\_TO\_SEC](https://www.w3schools.com/sql/func_mysql_time_to_sec.asp) | Converts a time value into numeric seconds |
| [TIMEDIFF](https://www.w3schools.com/sql/func_mysql_timediff.asp) | Returns the difference between two time/datetime values |
| [TIMESTAMP](https://www.w3schools.com/sql/func_mysql_timestamp.asp) | Converts an expression to a datetime value and if specified adds an optional time interval to the value |
| [TO\_DAYS](https://www.w3schools.com/sql/func_mysql_to_days.asp) | Converts a date into numeric days |
| [WEEK](https://www.w3schools.com/sql/func_mysql_week.asp) | Returns the week portion of a date value |
| [WEEKDAY](https://www.w3schools.com/sql/func_mysql_weekday.asp) | Returns the weekday index for a date value |
| [WEEKOFYEAR](https://www.w3schools.com/sql/func_mysql_weekofyear.asp) | Returns the week of the year for a date value |
| [YEAR](https://www.w3schools.com/sql/func_mysql_year.asp) | Returns the year portion of a date value |
| [YEARWEEK](https://www.w3schools.com/sql/func_mysql_yearweek.asp) | Returns the year and week for a date value |

MySQL Advanced Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [BIN](https://www.w3schools.com/sql/func_mysql_bin.asp) | Converts a decimal number to a binary number |
| [BINARY](https://www.w3schools.com/sql/func_mysql_binary.asp) | Converts a value to a binary string |
| [CASE](https://www.w3schools.com/sql/func_mysql_case.asp) | Lets you evaluate conditions and return a value when the first condition is met |
| [CAST](https://www.w3schools.com/sql/func_mysql_cast.asp) | Converts a value from one datatype to another datatype |
| [COALESCE](https://www.w3schools.com/sql/func_mysql_coalesce.asp) | Returns the first non-null expression in a list |
| [CONNECTION\_ID](https://www.w3schools.com/sql/func_mysql_connection_id.asp) | Returns the unique connection ID for the current connection |
| [CONV](https://www.w3schools.com/sql/func_mysql_conv.asp) | Converts a number from one number base to another |
| [CONVERT](https://www.w3schools.com/sql/func_mysql_convert.asp) | Converts a value from one datatype to another, or one character set to another |
| [CURRENT\_USER](https://www.w3schools.com/sql/func_mysql_current_user.asp) | Returns the user name and host name for the MySQL account used by the server to authenticate the current client |
| [DATABASE](https://www.w3schools.com/sql/func_mysql_database.asp) | Returns the name of the default database |
| [IF](https://www.w3schools.com/sql/func_mysql_if.asp) | Returns one value if a condition is TRUE, or another value if a condition is FALSE |
| [IFNULL](https://www.w3schools.com/sql/func_mysql_ifnull.asp) | Lets you to return an alternate value if an expression is NULL |
| [ISNULL](https://www.w3schools.com/sql/func_mysql_isnull.asp) | Tests whether an expression is NULL |
| [LAST\_INSERT\_ID](https://www.w3schools.com/sql/func_mysql_last_insert_id.asp) | Returns the first AUTO\_INCREMENT value that was set by the most recent INSERT or UPDATE statement |
| [NULLIF](https://www.w3schools.com/sql/func_mysql_nullif.asp) | Compares two expressions |
| [SESSION\_USER](https://www.w3schools.com/sql/func_mysql_session_user.asp) | Returns the user name and host name for the current MySQL user |
| [SYSTEM\_USER](https://www.w3schools.com/sql/func_mysql_system_user.asp) | Returns the user name and host name for the current MySQL user |
| [USER](https://www.w3schools.com/sql/func_mysql_user.asp) | Returns the user name and host name for the current MySQL user |
| [VERSION](https://www.w3schools.com/sql/func_mysql_version.asp) | Returns the version of the MySQL database |

SQL Server String Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [ASCII](https://www.w3schools.com/sql/func_sqlserver_ascii.asp) | Returns the number code that represents the specific character |
| [CHAR](https://www.w3schools.com/sql/func_sqlserver_char.asp) | Returns the ASCII character based on the number code |
| [CHARINDEX](https://www.w3schools.com/sql/func_sqlserver_charindex.asp) | Returns the location of a substring in a string |
| [CONCAT](https://www.w3schools.com/sql/func_sqlserver_concat.asp) | Concatenates two or more strings together |
| [Concat with +](https://www.w3schools.com/sql/func_sqlserver_concat_with_plus.asp) | Concatenates two or more strings together |
| [DATALENGTH](https://www.w3schools.com/sql/func_sqlserver_datalength.asp) | Returns the length of an expression (in bytes) |
| [LEFT](https://www.w3schools.com/sql/func_sqlserver_left.asp) | Extracts a substring from a string (starting from left) |
| [LEN](https://www.w3schools.com/sql/func_sqlserver_len.asp) | Returns the length of the specified string |
| [LOWER](https://www.w3schools.com/sql/func_sqlserver_lower.asp) | Converts a string to lower-case |
| [LTRIM](https://www.w3schools.com/sql/func_sqlserver_ltrim.asp) | Removes leading spaces from a string |
| [NCHAR](https://www.w3schools.com/sql/func_sqlserver_nchar.asp) | Returns the Unicode character based on the number code |
| [PATINDEX](https://www.w3schools.com/sql/func_sqlserver_patindex.asp) | Returns the location of a pattern in a string |
| [REPLACE](https://www.w3schools.com/sql/func_sqlserver_replace.asp) | Replaces a sequence of characters in a string with another set of characters |
| [RIGHT](https://www.w3schools.com/sql/func_sqlserver_right.asp) | Extracts a substring from a string (starting from right) |
| [RTRIM](https://www.w3schools.com/sql/func_sqlserver_rtrim.asp) | Removes trailing spaces from a string |
| [SPACE](https://www.w3schools.com/sql/func_sqlserver_space.asp) | Returns a string with a specified number of spaces |
| [STR](https://www.w3schools.com/sql/func_sqlserver_str.asp) | Returns a string representation of a number |
| [STUFF](https://www.w3schools.com/sql/func_sqlserver_stuff.asp) | Deletes a sequence of characters from a string and then inserts another sequence of characters into the string, starting at a specified position |
| [SUBSTRING](https://www.w3schools.com/sql/func_sqlserver_substring.asp) | Extracts a substring from a string |
| [UPPER](https://www.w3schools.com/sql/func_sqlserver_upper.asp) | Converts a string to upper-case |

SQL Server Numeric Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [ABS](https://www.w3schools.com/sql/func_sqlserver_abs.asp) | Returns the absolute value of a number |
| [AVG](https://www.w3schools.com/sql/func_sqlserver_avg.asp) | Returns the average value of an expression |
| [CEILING](https://www.w3schools.com/sql/func_sqlserver_ceiling.asp) | Returns the smallest integer value that is greater than or equal to a number |
| [COUNT](https://www.w3schools.com/sql/func_sqlserver_count.asp) | Returns the count of an expression |
| [FLOOR](https://www.w3schools.com/sql/func_sqlserver_floor.asp) | Returns the largest integer value that is equal to or less than a number |
| [MAX](https://www.w3schools.com/sql/func_sqlserver_max.asp) | Returns the maximum value of an expression |
| [MIN](https://www.w3schools.com/sql/func_sqlserver_min.asp) | Returns the minimum value of an expression |
| [RAND](https://www.w3schools.com/sql/func_sqlserver_rand.asp) | Returns a random number or a random number within a range |
| [ROUND](https://www.w3schools.com/sql/func_sqlserver_round.asp) | Returns a number rounded to a certain number of decimal places |
| [SIGN](https://www.w3schools.com/sql/func_sqlserver_sign.asp) | Returns a value indicating the sign of a number |
| [SUM](https://www.w3schools.com/sql/func_sqlserver_sum.asp) | Returns the summed value of an expression |

SQL Server Date Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [CURRENT\_TIMESTAMP](https://www.w3schools.com/sql/func_sqlserver_current_timestamp.asp) | Returns the current date and time |
| [DATEADD](https://www.w3schools.com/sql/func_sqlserver_dateadd.asp) | Returns a date after a certain time/date interval has been added |
| [DATEDIFF](https://www.w3schools.com/sql/func_sqlserver_datediff.asp) | Returns the difference between two date values, based on the interval specified |
| [DATENAME](https://www.w3schools.com/sql/func_sqlserver_datename.asp) | Returns a specified part of a given date, as a string value |
| [DATEPART](https://www.w3schools.com/sql/func_sqlserver_datepart.asp) | Returns a specified part of a given date, as an integer value |
| [DAY](https://www.w3schools.com/sql/func_sqlserver_day.asp) | Returns the day of the month (from 1 to 31) for a given date |
| [GETDATE](https://www.w3schools.com/sql/func_sqlserver_getdate.asp) | Returns the current date and time |
| [GETUTCDATE](https://www.w3schools.com/sql/func_sqlserver_getutcdate.asp) | Returns the current UTC date and time |
| [MONTH](https://www.w3schools.com/sql/func_sqlserver_month.asp) | Returns the month (from 1 to 12) for a given date |
| [YEAR](https://www.w3schools.com/sql/func_sqlserver_year.asp) | Returns the year (as a four-digit number) for a given date |

SQL Server Conversion Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [CAST](https://www.w3schools.com/sql/func_sqlserver_cast.asp) | Converts an expression from one data type to another |
| [CONVERT](https://www.w3schools.com/sql/func_sqlserver_convert.asp) | Converts an expression from one data type to another |

SQL Server Advanced Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [COALESCE](https://www.w3schools.com/sql/func_sqlserver_coalesce.asp) | Returns the first non-null expression in a list |
| [CURRENT\_USER](https://www.w3schools.com/sql/func_sqlserver_current_user.asp) | Returns the name of the current user in the SQL Server database |
| [ISDATE](https://www.w3schools.com/sql/func_sqlserver_isdate.asp) | Returns 1 if the expression is a valid date, otherwise 0 |
| [ISNULL](https://www.w3schools.com/sql/func_sqlserver_isnull.asp) | Lets you return an alternative value when an expression is NULL |
| [ISNUMERIC](https://www.w3schools.com/sql/func_sqlserver_isnumeric.asp) | Returns 1 if the expression is a valid number, otherwise 0 |
| [NULLIF](https://www.w3schools.com/sql/func_sqlserver_nullif.asp) | Compares two expressions |
| [SESSION\_USER](https://www.w3schools.com/sql/func_sqlserver_session_user.asp) | Returns the user name of the current session in the SQL Server database |
| [SESSIONPROPERTY](https://www.w3schools.com/sql/func_sqlserver_sessionproperty.asp) | Returns the setting for a specified option of a session |
| [SYSTEM\_USER](https://www.w3schools.com/sql/func_sqlserver_system_user.asp) | Returns the login name information for the current user in the SQL Server database |
| [USER\_NAME](https://www.w3schools.com/sql/func_sqlserver_user_name.asp) | Returns the user name in the SQL Server database |

MS Access String Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [Asc](https://www.w3schools.com/sql/func_msaccess_asc.asp) | Returns the number code that represents the specific character |
| [Chr](https://www.w3schools.com/sql/func_msaccess_chr.asp) | Returns the character based on the number code |
| [Concat with &](https://www.w3schools.com/sql/func_msaccess_concat.asp) | Concatenates multiple strings together into a single string with the & operator |
| [CurDir](https://www.w3schools.com/sql/func_msaccess_curdir.asp) | Returns the current path |
| [Format](https://www.w3schools.com/sql/func_msaccess_format.asp) | Formats a string expression |
| [InStr](https://www.w3schools.com/sql/func_msaccess_instr.asp) | Returns the position of the first occurrence of a string in another string |
| [InstrRev](https://www.w3schools.com/sql/func_msaccess_instrrev.asp) | Returns the position of the first occurrence of a string in another string, starting from the end of the string |
| [LCase](https://www.w3schools.com/sql/func_msaccess_lcase.asp) | Converts a string to lower-case |
| [Left](https://www.w3schools.com/sql/func_msaccess_left.asp) | Extracts a substring from a string (starting from left) |
| [Len](https://www.w3schools.com/sql/func_msaccess_len.asp) | Returns the length of the specified string |
| [LTrim](https://www.w3schools.com/sql/func_msaccess_ltrim.asp) | Removes leading spaces from a string |
| [Mid](https://www.w3schools.com/sql/func_msaccess_mid.asp) | Extracts a substring from a string (starting at any position) |
| [Replace](https://www.w3schools.com/sql/func_msaccess_replace.asp) | Replaces a sequence of characters in a string with another set of characters (a number of times) |
| [Right](https://www.w3schools.com/sql/func_msaccess_right.asp) | Extracts a substring from a string (starting from right) |
| [RTrim](https://www.w3schools.com/sql/func_msaccess_rtrim.asp) | Removes trailing spaces from a string |
| [Space](https://www.w3schools.com/sql/func_msaccess_space.asp) | Returns a string with a specified number of spaces |
| [Split](https://www.w3schools.com/sql/func_msaccess_split.asp) | Splits a string into substrings (based on a delimiter) and returns the result as an array of substrings |
| [Str](https://www.w3schools.com/sql/func_msaccess_str.asp) | Returns a string representation of a number |
| [StrComp](https://www.w3schools.com/sql/func_msaccess_strcomp.asp) | Returns an integer value representing the result of a string comparison |
| [StrConv](https://www.w3schools.com/sql/func_msaccess_strconv.asp) | Returns a converted string |
| [StrReverse](https://www.w3schools.com/sql/func_msaccess_strreverse.asp) | Returns a string whose characters are in reverse order |
| [Trim](https://www.w3schools.com/sql/func_msaccess_trim.asp) | Removes leading and trailing spaces from a string |
| [UCase](https://www.w3schools.com/sql/func_msaccess_ucase.asp) | Converts a string to upper-case |

MS Access Numeric Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [Abs](https://www.w3schools.com/sql/func_msaccess_abs.asp) | Returns the absolute value of a number |
| [Atn](https://www.w3schools.com/sql/func_msaccess_atn.asp) | Returns the arctangent of a number |
| [Avg](https://www.w3schools.com/sql/func_msaccess_avg.asp) | Returns the average value of an expression |
| [Cos](https://www.w3schools.com/sql/func_msaccess_cos.asp) | Returns the cosine of an angle |
| [Count](https://www.w3schools.com/sql/func_msaccess_count.asp) | Returns the number of records in a select query |
| [Exp](https://www.w3schools.com/sql/func_msaccess_exp.asp) | Returns e raised to the nth power |
| [Fix](https://www.w3schools.com/sql/func_msaccess_fix.asp) | Returns the integer portion of a number |
| [Format](https://www.w3schools.com/sql/func_msaccess_format_number.asp) | Takes a numeric expression and returns it as a formatted string |
| [Int](https://www.w3schools.com/sql/func_msaccess_int.asp) | Returns the integer portion of a number |
| [Max](https://www.w3schools.com/sql/func_msaccess_max.asp) | Returns the maximum value of an expression |
| [Min](https://www.w3schools.com/sql/func_msaccess_min.asp) | Returns the minimum value of an expression |
| [Randomize](https://www.w3schools.com/sql/func_msaccess_randomize.asp) | Allows you to change the seed value used by the random number generator for the Rnd() function |
| [Rnd](https://www.w3schools.com/sql/func_msaccess_rnd.asp) | Generates a random number |
| [Round](https://www.w3schools.com/sql/func_msaccess_round.asp) | Returns a number rounded to a certain number of decimal places |
| [Sgn](https://www.w3schools.com/sql/func_msaccess_sgn.asp) | Returns the sign of a number |
| [Sqr](https://www.w3schools.com/sql/func_msaccess_sqr.asp) | Returns the square root of a number |
| [Sum](https://www.w3schools.com/sql/func_msaccess_sum.asp) | Returns the summed value of an expression |
| [Val](https://www.w3schools.com/sql/func_msaccess_val.asp) | Accepts a string as input and returns the numbers found in that string |

MS Access Date Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [Date](https://www.w3schools.com/sql/func_msaccess_date.asp) | Returns the current system date |
| [DateAdd](https://www.w3schools.com/sql/func_msaccess_dateadd.asp) | Returns a date after a time/date interval has been added |
| [DateDiff](https://www.w3schools.com/sql/func_msaccess_datediff.asp) | Returns the difference between two date values |
| [DatePart](https://www.w3schools.com/sql/func_msaccess_datepart.asp) | Returns a specified part of a date |
| [DateSerial](https://www.w3schools.com/sql/func_msaccess_dateserial.asp) | Returns a date given a year, month, and day value |
| [DateValue](https://www.w3schools.com/sql/func_msaccess_datevalue.asp) | Converts a string to a date |
| [Day](https://www.w3schools.com/sql/func_msaccess_day.asp) | Returns the day of the month (from 1 to 31) for a date |
| [Format](https://www.w3schools.com/sql/func_msaccess_format_date.asp) | Takes a date expression and returns it as a formatted string |
| [Hour](https://www.w3schools.com/sql/func_msaccess_hour.asp) | Returns the hour (from 0 to 23) for a time value |
| [Minute](https://www.w3schools.com/sql/func_msaccess_minute.asp) | Returns the minute of the hour (from 0 to 59) for a time value |
| [Month](https://www.w3schools.com/sql/func_msaccess_month.asp) | Returns the month (from 1 to 12) for a date |
| [MonthName](https://www.w3schools.com/sql/func_msaccess_monthname.asp) | Returns the monthname given a number from 1 to 12 |
| [Now](https://www.w3schools.com/sql/func_msaccess_now.asp) | Returns the current system date and time |
| [Second](https://www.w3schools.com/sql/func_msaccess_second.asp) | Returns the second of the minute (from 0 to 59) for a time value |
| [Time](https://www.w3schools.com/sql/func_msaccess_time.asp) | Returns the current system time |
| [TimeSerial](https://www.w3schools.com/sql/func_msaccess_timeserial.asp) | Returns a time given an hour, minute, and second value |
| [TimeValue](https://www.w3schools.com/sql/func_msaccess_timevalue.asp) | Converts a string to a time |
| [Weekday](https://www.w3schools.com/sql/func_msaccess_weekday.asp) | Returns a number (from 1 to 7) representing the day of the week for a date |
| [WeekdayName](https://www.w3schools.com/sql/func_msaccess_weekdayname.asp) | Returns the weekday name given a number from 1 to 7 |
| [Year](https://www.w3schools.com/sql/func_msaccess_year.asp) | Returns the year (four digits) for a date |

MS Access Information Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| [CurrentUser](https://www.w3schools.com/sql/func_msaccess_currentuser.asp) | Returns the name of the current Access user |
| [Environ](https://www.w3schools.com/sql/func_msaccess_environ.asp) | Returns the value of an operating system environment variable |
| [IsDate](https://www.w3schools.com/sql/func_msaccess_isdate.asp) | Returns TRUE if the expression is a valid date, otherwise FALSE |
| [IsNull](https://www.w3schools.com/sql/func_msaccess_isnull.asp) | Returns TRUE if the expression is a null value, otherwise FALSE |
| [IsNumeric](https://www.w3schools.com/sql/func_msaccess_isnumeric.asp) | Returns TRUE if the expression is a valid number, otherwise FALSE |

Oracle String Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| ASCII | Returns the number code that represents the specified character |
| ASCIISTR | Converts a string in any character set to an ASCII string using the database character set |
| CHR | Returns the character based on the number code |
| COMPOSE | Returns a Unicode string |
| CONCAT | Allows you to concatenate two strings together |
| Concat with || | Allows you to concatenate two or more strings together |
| CONVERT | Converts a string from one character set to another |
| DECOMPOSE | Accepts a string and returns a Unicode string |
| DUMP | Returns a varchar2 value that includes the datatype code, the length in bytes, and the internal representation of the expression |
| INITCAP | Sets the first character in each word to uppercase and the rest to lowercase |
| INSTR | Returns the location of a substring in a string |
| INSTR2 | Returns the location of a substring in a string, using UCS2 code points |
| INSTR4 | Returns the location of a substring in a string, using UCS4 code points |
| INSTRB | Returns the location of a substring in a string, using bytes instead of characters |
| INSTRC | Returns the location of a substring in a string, using Unicode complete characters |
| LENGTH | Returns the length of the specified string |
| LENGTH2 | Returns the length of the specified string, using UCS2 code points |
| LENGTH4 | Returns the length of the specified string, using UCS4 code points |
| LENGTHB | Returns the length of the specified string, using bytes instead of characters |
| LENGTHC | Returns the length of the specified string, using Unicode complete of characters |
| LOWER | Converts all letters in the specified string to lowercase |
| LPAD | Pads the left-side of a string with a specific set of characters |
| LTRIM | Removes all specified characters from the left-hand side of a string |
| NCHR | Returns the character based on the number code in the national character set |
| REGEXP\_INSTR | Returns the location of a regular expression pattern in a string |
| REGEXP\_REPLACE | Allows you to replace a sequence of characters in a string with another set of characters using regular expression pattern matching |
| REGEXP\_SUBSTR | Allows you to extract a substring from a string using regular expression pattern matching |
| REPLACE | Replaces a sequence of characters in a string with another set of characters |
| RPAD | Pads the right-side of a string with a specific set of characters |
| RTRIM | Removes all specified characters from the right-hand side of a string |
| SOUNDEX | Returns a phonetic representation (the way it sounds) of a string |
| SUBSTR | Allows you to extract a substring from a string |
| TRANSLATE | Replaces a sequence of characters in a string with another set of characters |
| TRIM | Removes all specified characters either from the beginning or the end of a string |
| UPPER | Converts all letters in the specified string to uppercase |
| VSIZE | Returns the number of bytes in the internal representation of an expression |

Oracle Numeric Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| ABS | Returns the absolute value of a number |
| ACOS | Returns the arc cosine of a number |
| ASIN | Returns the arc sine of a number |
| ATAN | Returns the arc tangent of a number |
| ATAN2 | Returns the arc tangent of n and m |
| AVG | Returns the average value of an expression |
| BITAND | Returns an integer representing an AND operation on the bits of expr1 and expr2 |
| CEIL | Returns the smallest integer value that is greater than or equal to a number |
| COS | Returns the cosine of a number |
| COSH | Returns the hyperbolic cosine of a number |
| COUNT | Returns the count of an expression |
| EXP | Returns e raised to the power of number |
| FLOOR | Returns the largest integer value that is equal to or less than a number |
| GREATEST | Returns the greatest value in a list of expressions |
| LEAST | Returns the smallest value in a list of expressions |
| LN | Returns the natural logarithm of a number |
| LOG | Returns the natural logarithm of a number to a specified base |
| MAX | Returns the maximum value of an expression |
| MEDIAN | Returns the median of an expression |
| MIN | Returns the minimum value of an expression |
| MOD | Returns the remainder of n divided by m |
| POWER | Returns m raised to the nth power |
| REGEXP\_COUNT | Counts the number of times that a pattern occurs in a string |
| REMAINDER | Returns the remainder of m divided by n |
| ROUND | Returns a number rounded to a certain number of decimal places |
| ROWNUM | Returns a number that represents the order that a row is |
| SIGN | Returns a value indicating the sign of a number |
| SIN | Returns the sine of a number |
| SQRT | Returns the square root of a number |
| SUM | Returns the summed value of an expression |
| TAN | Returns the tangent of a number |
| TANH | Returns the hyperbolic tangent of n |
| TRUNC | Returns a number truncated to a certain number of decimal places |

Oracle Date Functions

|  |  |
| --- | --- |
| **Function** | **Description** |
| ADD\_MONTHS | Returns a date with a specified number of months added |
| CURRENT\_DATE | Returns the current date in the time zone of the current SQL session as set by the ALTER SESSION command |
| CURRENT\_TIMESTAMP | Returns the current date and time in the time zone of the current SQL session as set by the ALTER SESSION command |
| DBTIMEZONE | returns the database time zone as a time zone offset or a time zone region name |
| EXTRACT | Extracts a value from a date or interval value |
| LAST\_DAY | Returns the last day of the month based on a date value |
| LOCALTIMESTAMP | Returns the current date and time in the time zone of the current SQL session as set by the ALTER SESSION command |
| MONTHS\_BETWEEN | Returns the number of months between date1 and date2 |
| NEW\_TIME | Converts a date from time zone1 to a date in time zone2 |
| NEXT\_DAY | Returns the first weekday that is greater than a date |
| ROUND | Returns a date rounded to a specific unit of measure |
| SESSIONTIMEZONE | Returns the current session's time zone as a time zone offset or a time zone region name |
| SYSDATE | Returns the current system date and time on your local database |
| SYSTIMESTAMP | Returns the current system date and time (including fractional seconds and time zone) on your local database |
| TRUNC | Returns a date truncated to a specific unit of measure |
| TZ\_OFFSET | Returns the time zone offset of a value |

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 | [Try it](https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_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 |  |

SQL Data Types

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

An SQL developer must decide what type of data that will be stored inside each column when creating a table. The data type is 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.

**Note:** Data types might have different names in different database. And even if the name is the same, the size and other details may be different! **Always check the documentation!**

MySQL Data Types

In MySQL there are three main data types: text, number, and date.

Text data 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 data 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 data 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 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 |  |

Number 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 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 |

Microsoft 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 Quick Reference From W3Schools**

|  |  |
| --- | --- |
| **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 |