SQL is a standard language for accessing and manipulating databases.

SQL is a standard language for storing, manipulating and retrieving data in databases.

**What is SQL?**

* SQL stands for Structured Query Language
* SQL lets you access and manipulate databases
* SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987.

**What Can SQL do?**

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

Using SQL in Your Web Site

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

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

## RDBMS

RDBMS stands for Relational Database Management System.

RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

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

A record, also called a row, is each individual entry that exists in a table. For example, there are 91 records in the above Customers table. A record is a horizontal entity in a table.

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

# **SQL Syntax**

## Database Tables

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

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

## 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:

SELECT \* FROM Customers;

Keep in Mind That...

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

## 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.

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;

## SELECT Column Example

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

SELECT CustomerName, City FROM Customers;

## SELECT \* 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.

### **SELECT DISTINCT Syntax**

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

## SELECT DISTINCT Examples

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

SELECT DISTINCT Country FROM Customers;

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

SELECT COUNT(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;

## WHERE Clause Example

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

SELECT \* FROM Customers  
WHERE CustomerID=1;

## The SQL AND, OR and NOT Operators

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

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

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

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

### **AND Syntax**

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

## AND Example

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

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

### **OR Syntax**

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

## OR Example

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

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

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

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

### **NOT Syntax**

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

## NOT Example

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

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

## Combining AND, OR and NOT

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

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

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

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

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;

## ORDER BY Example

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

SELECT \* FROM Customers  
ORDER BY Country;

## ORDER BY DESC Example

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

SELECT \* FROM Customers  
ORDER BY Country DESC;

## ORDER BY Several Columns Example

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

SELECT \* FROM Customers  
ORDER BY Country, CustomerName;

## ORDER BY Several Columns Example 2

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

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, ...);

## INSERT INTO Example

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

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

## Insert Data Only in Specified Columns

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):

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

The unspecified field are null.

# **SQL NULL Values**

## What is a NULL Value?

A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

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

## How to Test for NULL Values?

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

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

### **IS NULL Syntax**

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

### **IS NOT NULL Syntax**

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

## The IS NULL Operator

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

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

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

Always use IS NULL to look for NULL values.

## The IS NOT NULL Operator

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

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

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

## 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;

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!

## UPDATE Table

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

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

## 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":

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

## Update Warning!

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

UPDATE Customers  
SET ContactName='Juan';

## The SQL DELETE Statement

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

### **DELETE Syntax**

DELETE FROM table\_name WHERE condition;

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

## SQL DELETE Example

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

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

## Delete All Records

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

DELETE FROM table\_name;

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

DELETE FROM Customers;

## 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;

## MIN() Example

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

SELECT MIN(Price) AS SmallestPrice  
FROM Products;

## MAX() Example

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

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;

## COUNT() Example

The following SQL statement finds the number of products:

SELECT COUNT(ProductID)  
FROM Products;

**Note:** NULL values are not counted.

## AVG() Example

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

SELECT AVG(Price)  
FROM Products;

**Note:** NULL values are ignored.

## SUM() Example

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

SELECT SUM(Quantity)  
FROM OrderDetails;

**Note:** NULL values are ignored.

The SQL LIKE Operator

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

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

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

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

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

### **LIKE Syntax**

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

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 start with "a"

WHERE CustomerName LIKE '%a' Finds any values that end with "a"

WHERE CustomerName LIKE '%or%' Finds any values that have "or" in any position

WHERE CustomerName LIKE '\_r%' Finds any values that have "r" in the second position

WHERE CustomerName LIKE 'a\_%\_%' Finds any values that start with "a" and are at least 3

Characters in length

WHERE ContactName LIKE 'a%o' Finds any values that start with "a" and ends with "o"

## SQL LIKE Examples

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## SQL Wildcard Characters

A wildcard character is used to substitute one or more characters 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.

**Wildcard Characters in MS Access**

Symbol Description Example

\* Represents zero or more characters bl\* finds bl, black, blue, and blob

? Represents a single character h?t finds hot, hat, and hit

[] Represents any single character within the brackets h[oa]t finds hot and hat, but not hit

! Represents any character not in the brackets h[!oa]t finds hit, but not hot and hat

- Represents a range of characters c[a-b]t finds cat and cbt

# Represents any single numeric character 2#5 finds 205, 215, 225, 235, 245, 255, 265, 275, 285, and 295

**Wildcard Characters in SQL Server**

Symbol Description Example

% Represents zero or more characters bl% finds bl, black, blue, and blob

\_ Represents a single character h\_t finds hot, hat, and hit

[] Represents any single character within the brackets h[oa]t finds hot and hat, but not hit

^ Represents any character not in the brackets h[^oa]t finds hit, but not hot and hat

- Represents a range of characters c[a-b]t finds cat and cbt

All 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"

## Using the % Wildcard

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

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

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

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

## Using the \_ Wildcard

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

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

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":

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

## Using the [charlist] Wildcard

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

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

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

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

## Using the [!charlist] Wildcard

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

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

OR

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

## 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:

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

and it will produce something like this:

|  |  |  |
| --- | --- | --- |
| OrderID | CustomerName | OrderDate |
| 10308 | Ana Trujillo Emparedados y helados | 9/18/1996 |
| 10365 | Antonio Moreno Taquería | 11/27/1996 |
| 10383 | Around the Horn | 12/16/1996 |
| 10355 | Around the Horn | 11/15/1996 |
| 10278 | Berglunds snabbkop | 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 table2ON table1.column\_name = table2.column\_name;



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 Trujilo | Avda.de la Constitucion 2222 | Mexico DF | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | Mexico DF | 05023 | Mexico |

## SQL INNER JOIN Example

The following SQL statement selects all orders with customer information:

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

**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:

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 table2ON table1.column\_name = table2.column\_name;

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



## SQL LEFT JOIN Example

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

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

**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 table2ON table1.column\_name = table2.column\_name;

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



Selection from the "Orders" table:

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 placed:

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

**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 table2ON table1.column\_name = table2.column\_name;



Using selection from Orders and Customers table:

## 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;

Selection from the "Customers" table,

## SQL Self JOIN Example

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

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;