SQL

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

Practice commands  
**SELECT**

SELECT CustomerId, CustomerName, City FROM Customers;   
This selects the three columns CustomerId, CustomerName and City from the table Customers.

SELECT \* FROM Customers  
Selects all columns and rows from the customers table.

Product Table:

SELECT ProductName AS "ProdName", Price AS "Price" FROM Products  
WHERE Price = 18;  
  
**DISTINCT**statement is used to return only distinct (different) values.

SELECT DISTINCT Country FROM Customers;  
SQL statement selects only the DISTINCT values from the "Country" column in the "Customers" table and lists the distinct countries:

SELECT COUNT(DISTINCT Country) FROM Customers;  
SQL statement lists the number of different (distinct) customer countries, gives the total number of distinct countries:

**Note: The example above will not work in Firefox!** Because COUNT(DISTINCT column\_name) is not supported in Microsoft Access databases. Firefox is using Microsoft Access in our examples.

Here is the workaround for MS Access:

### **Example**

SELECT Count(\*) AS DistinctCountries  
FROM (SELECT DISTINCT Country FROM Customers);

**WHERE**  
clause is used to filter records. It is used to extract only those records that fulfill a specified condition.

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

SELECT \* FROM Customers  
WHERE Country='Mexico';  
  
SQL statement selects all the customers from the country "Mexico", in the "Customers" table:

SELECT \* FROM Customers  
WHERE CustomerID=1;

SQL requires single quotes around text values (most database systems will also allow double quotes). However, numeric fields should not be enclosed in quotes:

## Operators in The WHERE Clause

**Operator** **Description**

= Equal

> Greater than

< Less than

>= Greater than or equal

<= Less than or equal

<> Not equal.

Note: In some versions of SQL this operator may be written as !=

BETWEEN Between a certain range

LIKE Search for a pattern

IN To specify multiple possible values for a column

SELECT \* FROM Customers

WHERE Country='Mexico' or Country ='UK'

OR CustomerID BETWEEN 1 AND 10

ORDER BY country DESC;

SQL statement selects all the customers from the country "Mexico" or UK or with customerID between 1 and 10 and then orders in descending order, in the "Customers" table:

## 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.

**OR**

SELECT \* FROM Customers

WHERE Country='Mexico' OR Country ='UK'

OR CustomerID BETWEEN 1 AND 10

ORDER BY country DESC;

SQL statement selects all the customers from the country "Mexico" or UK or with customerID between 1 and 10 and then orders in descending order, in the "Customers" table:

**AND**

SELECT \* FROM Customers

WHERE Country='Mexico' AND Country ='UK';

### **NOT Syntax**

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

SELECT \* FROM Customers

WHERE NOT Country ='USA';

## Combining AND, OR and NOT

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

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 NOT Country='Germany' AND NOT Country='USA';

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

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

SELECT \* FROM Customers  
ORDER BY Country, CustomerName;

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 ASC, CustomerName DESC;

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

## 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:

1. Specify both the column names and the values to be inserted:

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

2. 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. Here, the INSERT INTO syntax would be as follows:

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

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');

**NULL**

What is a NULL Value?

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

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

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

How to Test for NULL Values?

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

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

**IS NULL**

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

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

**IS NOT NULL**

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 *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!

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

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

## UPDATE Multiple Records

It is the WHERE clause that determines how many records 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';

## 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) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

DELETE FROM Customers   
WHERE CustomerName='Alfreds Futterkiste';

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

## 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;

SQL SELECT TOP Clause

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

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

**Note:** Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses FETCH FIRST *n* ROWS ONLY and ROWNUM.

**SQL Server / MS Access Syntax:**

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

**MySQL Syntax:**

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

**Oracle 12 Syntax:**

SELECT *column\_name(s)*  
FROM *table\_name*ORDER BY *column\_name(s)*  
FETCH FIRST *number* ROWS ONLY;

## SQL TOP, LIMIT and FETCH FIRST Examples

The following SQL statement selects the first three records from the "Customers" table (for SQL Server/MS Access):

SELECT TOP 3 \* FROM Customers;

The following SQL statement shows the equivalent example for MySQL:

SELECT \* FROM Customers  
LIMIT 3;

the following SQL statement shows the equivalent example for Oracle:

SELECT \* FROM Customers  
FETCH FIRST 3 ROWS ONLY;

## SQL TOP PERCENT Example

The following SQL statement selects the first 50% of the records from the "Customers" table (for SQL Server/MS Access):

SELECT TOP 50 PERCENT \* FROM Customers;

The following SQL statement shows the equivalent example for Oracle:

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

## ADD a WHERE CLAUSE

The following SQL statement selects the first three records from the "Customers" table, where the country is "Germany" (for SQL Server/MS Access):

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

The following SQL statement shows the equivalent example for MySQL:

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

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

SELECT MIN(Price) AS SmallestPrice  
FROM Products;

### **MAX() Syntax**

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

## 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 criterion.

### **COUNT() Syntax**

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

## COUNT() Example

The following SQL statement finds the number of products:

SELECT COUNT(ProductID)  
FROM Products;

**Using Count with distinct and alias**

SELECT COUNT ( DISTINCT SupplierID ) AS "SID"

FROM Products;

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

### **AVG() Syntax**

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

## AVG() Example

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

SELECT AVG(Price)  
FROM Products;

**Using Avg with Where**

SELECT AVG(Price)  
FROM Products  
WHERE SupplierId = 1 or SupplierID =7 or SupplierID =9;

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

### **SUM() Syntax**

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

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

SELECT SUM(Quantity)  
FROM OrderDetails;

SELECT SUM(Quantity)  
FROM OrderDetails  
WHERE QUANTITY =40;

## The SQL LIKE Operator

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

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

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

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

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

### **LIKE Syntax**

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

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

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 2 characters in length

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

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

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 [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 any single character within the   
 specified range 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 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 any single character within the specified  
 range 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"

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 "ondon":

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

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]%';

## 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);

## IN Operator Examples

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

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

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

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

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

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;

## BETWEEN Example

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

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

## NOT BETWEEN Example

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

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

## 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:

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

## BETWEEN Text Values Example

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

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

The following SQL statement selects all products with a ProductName between Carnarvon Tigers and Chef Anton's Cajun Seasoning:

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

## NOT BETWEEN Text Values Example

The following SQL statement selects all products with a ProductName not between Carnarvon Tigers and Mozzarella di Giovanni:

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

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 '01-July-1996' and '31-July-1996':

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

OR:

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

## 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 that query.

An alias is created with the AS keyword.

### **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;

## Alias for Columns Examples

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

SELECT CustomerID AS ID, CustomerName AS Customer  
FROM Customers;

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:

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

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

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

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

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

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

SELECT CustomerName, (Address || ', ' || PostalCode || ' ' || City || ', ' || Country) AS Address  
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):

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;

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

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

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:

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;

**Here are the different types of the JOINs in SQL:**

* (INNER) JOIN: Returns records that have matching values in both tables. No null values?
* LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table. Displays Null values as well?
* RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
* FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table

## SQL INNER JOIN Keyword

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

### **INNER JOIN Syntax**

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

## 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 matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

### **LEFT JOIN Syntax**

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

## 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 matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

### **RIGHT JOIN Syntax**

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

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

## 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 returns all records when there is a match in left (table1) or right (table2) table records.

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

### **FULL OUTER JOIN Syntax**

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

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

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;

**Note:** The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, 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;

T1 and T2 are different table aliases for the same 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;

## The SQL UNION Operator

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

* Every SELECT statement within UNION must have the same number of columns
* The columns must also have similar data types
* The columns in every SELECT statement must also be in the same order

### **UNION Syntax**

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

### **UNION ALL Syntax**

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

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

**Note:** The column names in the result-set are usually equal to the column names in the first SELECT statement.

## SQL UNION Example

The following SQL statement returns the cities (only distinct values) from both the "Customers" and the "Suppliers" table:

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

**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 returns the cities (duplicate values also) from both the "Customers" and the "Suppliers" table:

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

## SQL UNION With WHERE

The following SQL statement returns the German cities (only distinct values) from both the "Customers" and the "Suppliers" table:

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

## SQL UNION ALL With WHERE

The following SQL statement returns the German cities (duplicate values also) from both the "Customers" and the "Suppliers" table:

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

## Another UNION Example

The following SQL statement lists all customers and suppliers:

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

Notice the "AS Type" above - it is an alias. [SQL Aliases](https://www.w3schools.com/sql/sql_alias.asp) are used to give a table or a column a temporary name. An alias only exists for the duration of the query. So, here we have created a temporary column named "Type", that list whether the contact person is a "Customer" or a "Supplier".

## The SQL GROUP BY Statement

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

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

### **GROUP BY Syntax**

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

## SQL GROUP BY Examples

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

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

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

## GROUP BY With JOIN Example

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

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

## The SQL GROUP BY Statement

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

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

### **GROUP BY Syntax**

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

## SQL GROUP BY Examples

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

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

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

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

## GROUP BY With JOIN Example

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

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

OR

SELECT COUNT(OrderID) as NumberOfOrders, ShipperId From Orders

Group By ShipperID;

(I wrote the above statement)

## The SQL HAVING Clause

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

### **HAVING Syntax**

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

## SQL HAVING Examples

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

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

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

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

## More HAVING Examples

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

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;

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

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;