SQL

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

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

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.

```
SELECT * FROM Customers;
```

SQL Statements

- 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 *column1*, *column2* FROM *table_name*;

SELECT * FROM *table_name*;

Example:

SELECT CustomerName, City FROM Customers;

SQL Statements

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 *column1*, *column2* FROM *table_name*;

- Example:
- SELECT Country FROM Customers;

SQL Statements

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.

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

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example:

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

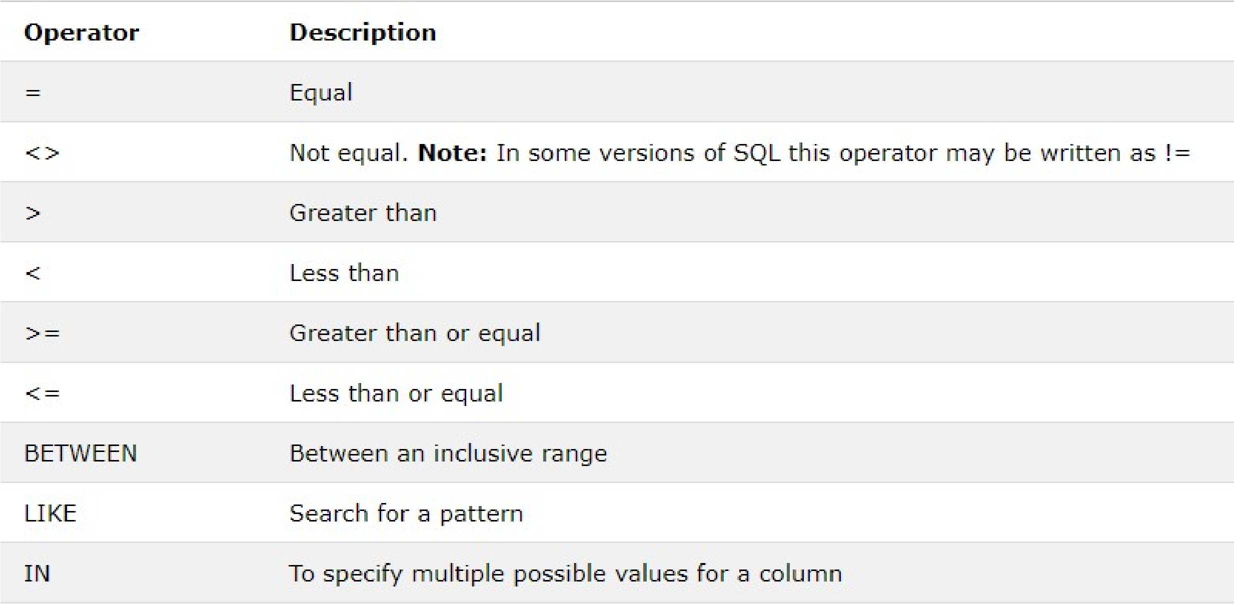
SQL Statements

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:

```
SELECT * FROM Customers
WHERE CustomerID=1;
```



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

SQL Statements

- 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.
- SELECT *column1*, *column2, ...*
  FROM *table_name*
  WHERE *condition1* AND *condition2* AND *condition3 ...*;
- SELECT *column1*, *column2, ...*
  FROM *table_name*
  WHERE *condition1* OR *condition2* OR *condition*
- SELECT *column1*, *column2, ...*
  FROM *table_name*
  WHERE NOT *condition;n3 ...*;

SQL Statements

Example:

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

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

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

You can also combine the AND, OR and NOT operators

```
SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München');
SELECT * FROM Customers
WHERE NOT Country='Germany' AND NOT Country='USA';
```

SQL Statements

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;
```

Example:

```
SELECT * FROM Customers
ORDER BY Country;
```

SQL Statements

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
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. Example:

```
INSERT INTO Customers (CustomerID, CustomerName, ContactName, Address, City,
PostalCode, Country)
VALUES (100, 'Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

SQL Statements

Insert Data Only in Specified Columns

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

```
INSERT INTO table_name (column1, column2, column3)
VALUES (value1, value2, value3);
```

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');
```

SQL Statements

SQL NULL Values

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!

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	Example
SELECT column_names FROM table_name	SELECT LastName, FirstName, Address FROM Persons
WHERE column_name IS NULL;	WHERE Address IS NULL;
IS NOT NULL Syntax	
SELECT column_names FROM table_name	SELECT LastName, FirstName, Address FROM Persons
WHERE column_name IS NOT NULL;	WHERE Address IS NOT NULL;

SQL Statements

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;
```

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!

Example:

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

SQL Statements

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';
```

Update Warning!

```
UPDATE Customers
SET ContactName='Juan';
```

SQL Statements

The SQL DELETE Statement

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

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!

Example:

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*; or DELETE * FROM *table_name*;

SQL Statements

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.

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.

SELECT TOP *number|percent column_name(s)* FROM *table_name*
WHERE *condition*;

Example:

SELECT TOP 3 * FROM Customers;

SELECT TOP 50 PERCENT * FROM Customers;

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';

SQL Statements

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.

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

Example

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

SQL Statements

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.

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

```
SELECT COUNT(ProductID)
FROM Products;
```

```
SELECT AVG(Price)
FROM Products;
```

```
SELECT SUM(Quantity)
FROM Products;
```

SQL Statements

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

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

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

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

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

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

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

SQL Statements

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 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

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

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

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

SQL Statements

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.

```
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);
```

Example

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

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

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

SQL Statements

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.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Example

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

SQL Statements

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.

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

```
SELECT column_name AS alias_name
FROM table_name;
```

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

SQL Statements

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;
```

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;
```

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 AS Address
FROM Customers;
```

SQL Statements

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;
```

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

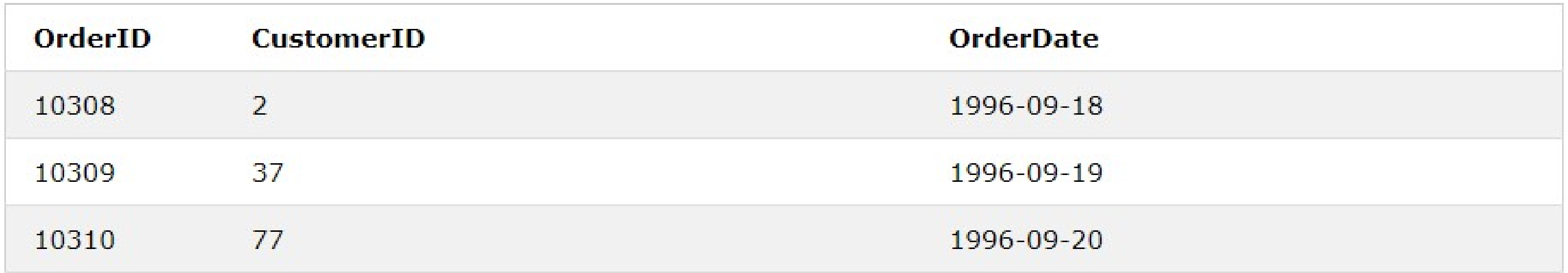
Example

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

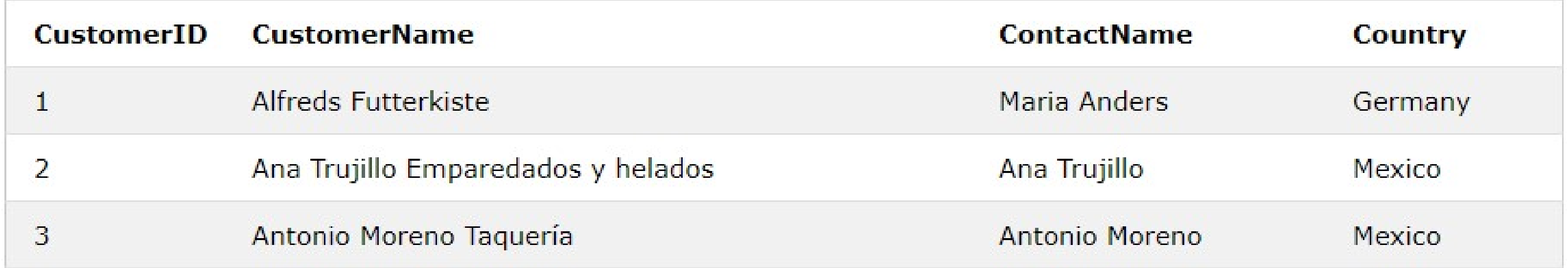
SQL Statements

SQL JOIN

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



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



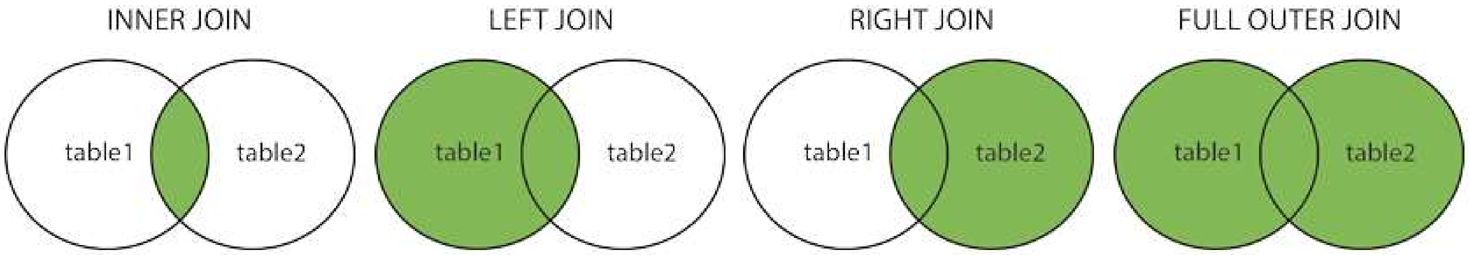
CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

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

SQL Statements

Here are the different types of the JOINs in SQL:

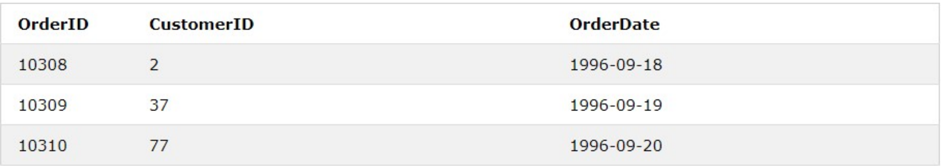
- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table



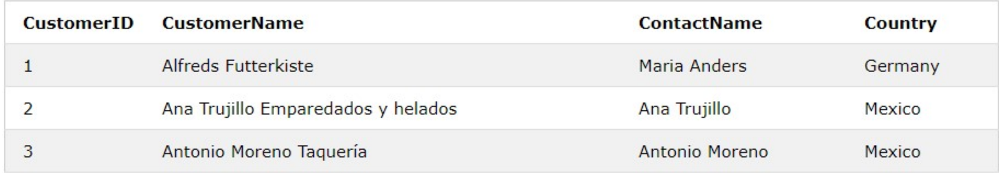
SQL Statements

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

```
SELECT column_name(s)
FROM table1
INNER JOIN table2 ON table1.column_name = table2.column_name;
```



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



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

Example:

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

SQL Statements

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 Statements

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.

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2 ON table1.column_name = table2.column_name;
```

In some databases LEFT JOIN is called LEFT OUTER JOIN.

Example:

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

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

SQL Statements

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.

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2 ON table1.column_name = table2.column_name;
```

In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

Example:

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

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

SQL Statements

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.

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

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2 ON table1.column_name = table2.column_name;
```

Example:

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

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 Statements

- 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
- SELECT *column_name(s)* FROM *table1*
  UNION
  SELECT *column_name(s)* FROM *table2*;

SQL Statements

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

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

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

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

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

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

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

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

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;
```

SQL Statements

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.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

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;
```

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;
```

SQL Statements

The SQL HAVING Clause

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

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

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;
```

SQL Statements

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.

SELECT *column_name(s)*
FROM *table_name*
WHERE EXISTS
(SELECT *column_name* FROM *table_name* WHERE *condition*);

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

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

SQL Statements

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.

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

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);
```

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

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

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

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

SQL Statements

The SQL SELECT INTO Statement

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

Copy all columns into a new table:

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

Examples

The following SQL statement creates a backup copy of Customers:

```
SELECT * INTO CustomersBackup2017 FROM Customers;
```

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;
```

SQL Statements

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.

Copy all columns from one table to another table:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

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

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

SQL Comments

SQL Comments

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

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:

```
--Select all:
SELECT * FROM Customers;
```

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:

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