SQL Notes

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 is an ANSI (American National Standards Institute) standard

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

RDBMS

- RDBMS stands for Relational Database Management System.
- RDBMS is the basis for SQL, and for all modern database systems like 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 collections of related data entries and it consists of columns and rows.

Example

```
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.

Example

Return data from the Customers table:

```
SELECT CustomerName, City FROM Customers;
```

Syntax

`SELECT` *`column1`* `,` *`column2, ...`* `FROM` *`table_name`* `;`

Here, column1, column2, ... are the *field names* of the table you want to select data from.

The table_name represents the name of the *table* you want to select data from.

Table name: Customers

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 ALL columns

If you want to return all columns, without specifying every column name, you can use the `SELECT *` syntax:

Example

Return all the columns from the Customers table:

```
SELECT * FROM Customers;
```

SQL SELECT DISTINCT Statement

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

Example

```
SELECT DISTINCT Country FROM Customers;
```

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

Syntax

SELECT DISTINCT column1, column2,...

FROM table_name;

Table name: Customers

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 Without DISTINCT

If you omit the `DISTINCT` keyword, the SQL statement returns the "Country" value from all the records of the "Customers" table:

Example

SELECT Country FROM Customers;

Count Distinct

By using the `DISTINCT` keyword in a function called `COUNT`, we can return the number of different countries.

Example

```
SELECT COUNT(DISTINCT Country) FROM Customers;
```

Note: The COUNT(DISTINCT *column_name*) is not supported in Microsoft Access databases.

SQL WHERE Clause

The `WHERE` clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

Example

Select all customers from Mexico:

```
SELECT * FROM CustomersWHERE Country='Mexico';
```

Syntax

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

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

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 CustomersWHERE CustomerID=1;
```

Operators in The WHERE Clause

You can use other operators than the `=` operator to filter the search.

Example

Select all customers with a CustomerID greater than 80:

```
SELECT * FROM Customers
WHERE CustomerID > 80;
```

The following operators can be used 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

SQL ORDER BY Keyword

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

Example

```
SELECT * FROM Products
ORDER BY Price;
```

Syntax

`SELECT` *`column1`* `,` *`column2, ...`*
`FROM` *`table_name`*
`ORDER BY` *`column1, column2, ...`* `ASC|DESC;`

DESC

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

Example

Sort the products from highest to lowest price:

```
SELECT * FROM Products
ORDER BY Price DESC;
```

Order Alphabetically

For string values the `ORDER BY` keyword will order alphabetically:

Example

Sort the products alphabetically by ProductName:

```
SELECT * FROM Products
ORDER BY ProductName;
```

Alphabetically DESC

To sort the table reverse alphabetically, use the `DESC` keyword:

Example

Sort the products by ProductName in reverse order:

```
SELECT * FROM Products
ORDER BY ProductName DESC;
```

ORDER BY Several Columns

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:

Example

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

Using Both ASC and DESC

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;
```

SQL AND Operator

The `WHERE` clause can contain one or many `AND` operators.

The `AND` operator is used to filter records based on more than one condition, like if you want to return all customers from Spain that starts with the letter 'G':

Example

Select all customers from Spain that starts with the letter 'G':

```
SELECT *FROM CustomersWHERE Country = 'Spain' AND CustomerName LIKE 'G%';
```

Syntax

```
SELECT column1, column2, ...FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

AND vs OR

The `AND` operator displays a record if *all* the conditions are TRUE.

The `OR` operator displays a record if *any* of the conditions are TRUE.

Demo Database

Below is a selection from the **Customers** table used in the examples:

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

All Conditions Must Be True

The following SQL statement selects all fields from `Customers` where `Country` is "Germany" AND `City` is "Berlin" AND `PostalCode` is higher than 12000:

Example

```
SELECT * FROM Customers
WHERE Country = 'Germany'AND City = 'Berlin' AND PostalCode > 12000;
```

Combining AND and OR

You can combine the `AND` and `OR` operators.

The following SQL statement selects all customers from Spain that starts with a "G" or an "R".

Make sure you use parenthesis to get the correct result.

Example

Select all Spanish customers that starts with either "G" or "R":

```
SELECT * FROM Customers
WHERE Country = 'Spain' AND (CustomerName LIKE 'G%' OR CustomerName LIKE 'R%');
```

Without parenthesis, the select statement will return all customers from Spain that starts with a "G", *plus* all customers that starts with an "R", regardless of the country value:

Example

Select all customers that either:

are from Spain and starts with either "G", *or*

starts with the letter "R":

```
SELECT * FROM Customers
WHERE Country = 'Spain' AND CustomerName LIKE 'G%' OR CustomerName LIKE 'R%';
```

SQL OR Operator

The `WHERE` clause can contain one or more `OR` operators.

The `OR` operator is used to filter records based on more than one condition, like if you want to return all customers from Germany but also those from Spain:

Example

Select all customers from Germany or Spain:

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

Syntax

`SELECT` `column1` `,` `column2, ...` `FROM` `table_name` `WHERE` `condition1` `OR` `condition2` `OR` `condition3 ...` `;`

OR vs AND

The `OR` operator displays a record if *any* of the conditions are TRUE.

The `AND` operator displays a record if *all* the conditions are TRUE.

Demo Database

Below is a selection from the **Customers** table used in the examples:

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

At Least One Condition Must Be True

The following SQL statement selects all fields from Customers where either `City` is "Berlin", `CustomerName` starts with the letter "G" or `Country` is "Norway":

Example

```
SELECT * FROM CustomersWHERE City = 'Berlin' OR CustomerName LIKE 'G%' OR Country = 'Norway';
```

Combining AND and OR

You can combine the `AND` and `OR` operators.

The following SQL statement selects all customers from Spain that starts with a "G" or an "R".

Make sure you use parenthesis to get the correct result.

Example

Select all Spanish customers that starts with either "G" or "R":

```
SELECT * FROM Customers
WHERE Country = 'Spain' AND (CustomerName LIKE 'G%' OR CustomerName LIKE 'R%');
```

Without parenthesis, the select statement will return all customers from Spain that starts with a "G", *plus* all customers that starts with an "R", regardless of the country value:

Example

Select all customers that either:

are from Spain and starts with either "G", *or*

starts with the letter "R":

```
SELECT * FROM Customers
WHERE Country = 'Spain' AND CustomerName LIKE 'G%' OR CustomerName LIKE 'R%';
```

SQL NOT Operator

The `NOT` operator is used in combination with other operators to give the opposite result, also called the negative result.

In the select statement below we want to return all customers that are NOT from Spain:

Example

Select only the customers that are NOT from Spain:

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

In the example above, the `NOT` operator is used in combination with the `=` operator, but it can be used in combination with other comparison and/or logical operators. See examples below.

Syntax

`SELECT` `column1` `,` `column2, ...` `FROM` `table_name` `WHERE NOT` `condition` `;`

Demo Database

Below is a selection from the **Customers** table used in the examples:

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

NOT LIKE

Example

Select customers that does not start with the letter 'A':

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

NOT BETWEEN

Example

Select customers with a customerID not between 10 and 60:

```
SELECT * FROM Customers
WHERE CustomerID NOT BETWEEN 10 AND 60;
```

NOT IN

Example

Select customers that are not from Paris or London:

```
SELECT * FROM Customers
WHERE City NOT IN ('Paris', 'London');
```

NOT Greater Than

Example

Select customers with a CustomerId not greater than 50:

```
SELECT * FROM Customers
WHERE NOT CustomerID > 50;
```

Note: There is a not-greater-than operator: `!>` that would give you the same result.

NOT Less Than

Example

Select customers with a CustomerID not less than 50:

```
SELECT * FROM Customers
WHERE NOT CustomerId < 50;
```

Note: There is a not-less-than operator: `!<` that would give you the same result.

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` `, ...);`

Demo Database

Below is a selection from the **Customers** table used in the examples:

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 ('Cardi
```

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

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');

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	null	null	Stavanger	null	Norway

Insert Multiple Rows

It is also possible to insert multiple rows in one statement.

To insert multiple rows of data, we use the same `INSERT INTO` statement, but with multiple values:

Example

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)VALUES('Cardi
```

Make sure you separate each set of values with a comma `,`.

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
93	Greasy Burger	Per Olsen	Gateveien 15	Sandnes	4306	Norway
94	Tasty Tee	Finn Egan	Streetroad 19B	Liverpool	L1 0AA	UK

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.

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

How to Test for NULL Values?

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

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

IS NULL Syntax

`SELECT` *`column_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

Below is a selection from the **Customers** table used in the examples:

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	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

	Taquería					
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 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:

Example

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

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

The IS NOT NULL Operator

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

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

Example

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

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 used in the examples:

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 CustomersSET ContactName = 'Alfred Schmidt', City= 'Frankfurt'WHERE CustomerID = 1;

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 will be updated.

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

Example

UPDATE CustomersSET ContactName='Juan'WHERE Country='Mexico';

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 CustomersSET ContactName='Juan';

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

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!

Demo Database

Below is a selection from the **Customers** table used in the examples:

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';
```

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`* `;`

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

Example

```
DELETE FROM Customers;
```

Delete a Table

To delete the table completely, use the `DROP TABLE` statement:

Example

Remove the Customers table:

```
DROP TABLE Customers;
```

SQL TOP, LIMIT, FETCH FIRST or ROWNUM 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.

Example

Select only the first 3 records of the Customers table:

```
SELECT TOP 3 * FROM Customers;
```

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`.

MySQL Syntax:

```
SELECT column_name(s)FROM table_nameWHERE conditionLIMIT number;
```

Demo Database

Below is a selection from the **Customers** table used in the examples:

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

LIMIT

The following SQL statement shows the equivalent example for MySQL:

Example

Select the first 3 records of the Customers table:

```
SELECT * FROM CustomersLIMIT 3;
```

FETCH FIRST

The following SQL statement shows the equivalent example for Oracle:

Example

Select the first 3 records of the Customers table:

```
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):

Example

```
SELECT TOP 50 PERCENT * FROM Customers;
```

The following SQL statement shows the equivalent example for Oracle:

Example

```
SELECT * FROM CustomersFETCH 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):

Example

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

The following SQL statement shows the equivalent example for MySQL:

Example

SELECT * FROM CustomersWHERE Country='Germany'LIMIT 3;

The following SQL statement shows the equivalent example for Oracle:

Example

SELECT * FROM CustomersWHERE Country='Germany'FETCH FIRST 3 ROWS ONLY;

ADD the ORDER BY Keyword

Add the `ORDER BY` keyword when you want to sort the result, and return the first 3 records of the sorted result.

Example

Sort the result reverse alphabetically by CustomerName, and return the first 3 records:

```
SELECT TOP 3 * FROM Customers
ORDER BY CustomerName DESC;
```

The following SQL statement shows the equivalent example for MySQL:

Example

```
SELECT * FROM Customers
ORDER BY CustomerName DESCLIMIT 3;
```

The following SQL statement shows the equivalent example for Oracle:

Example

```
SELECT * FROM Customers
ORDER BY CustomerName DESCFETCH FIRST 3 ROWS ONLY;
```

SQL Aggregate Functions

An aggregate function is a function that performs a calculation on a set of values, and returns a single value.

Aggregate functions are often used with the `GROUP BY` clause of the `SELECT` statement. The `GROUP BY` clause splits the result-set into groups of values and the aggregate function can be used to return a single value for each group.

The most commonly used SQL aggregate functions are:

- `MIN()` - returns the smallest value within the selected column
- `MAX()` - returns the largest value within the selected column

- `COUNT()` - returns the number of rows in a set
- `SUM()` - returns the total sum of a numerical column
- `AVG()` - returns the average value of a numerical column

Aggregate functions ignore null values (except for `COUNT()`).

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 Example

Find the lowest price in the Price column:

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

MAX Example

Find the highest price in the Price column:

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

Syntax

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

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

Demo Database

Below is a selection from the **Products** table used in the examples:

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

Set Column Name (Alias)

When you use `MIN()` or `MAX()`, the returned column will not have a descriptive name. To give the column a descriptive name, use the `AS` keyword:

Example

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

Use MIN() with GROUP BY

Here we use the `MIN()` function and the `GROUP BY` clause, to return the smallest price for each category in the Products table:

Example

```
SELECT MIN(Price) AS SmallestPrice, CategoryID
FROM ProductsGROUP BY CategoryID;
```

SQL COUNT() Function

The `COUNT()` function returns the number of rows that matches a specified criterion.

Example

Find the total number of rows in the `Products` table:

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

Syntax

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

Demo Database

Below is a selection from the **Products** table used in the examples:

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

Specify Column

You can specify a column name instead of the asterix symbol `(*)`.

If you specify a column name instead of `(*)`, NULL values will not be counted.

Example

Find the number of products where the `ProductName` is not null:

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

Add a WHERE Clause

You can add a `WHERE` clause to specify conditions:

Example

Find the number of products where `Price` is higher than 20:

```
SELECT COUNT(ProductID)
FROM ProductsWHERE Price > 20;
```

Ignore Duplicates

You can ignore duplicates by using the `DISTINCT` keyword in the `COUNT()` function.

If `DISTINCT` is specified, rows with the same value for the specified column will be counted as one.

Example

How many *different* prices are there in the `Products` table:

```
SELECT COUNT(DISTINCT Price)
FROM Products;
```

Use an Alias

Give the counted column a name by using the `AS` keyword.

Example

Name the column "Number of records":

```
SELECT COUNT(*) AS [Number of records]
FROM Products;
```

Use COUNT() with GROUP BY

Here we use the `COUNT()` function and the `GROUP BY` clause, to return the number of records for each category in the Products table:

Example

```
SELECT COUNT(*) AS [Number of records], CategoryID
FROM Products
GROUP BY CategoryID;
```