Using where clause and operators with where clause:

The WHERE clause is used to filter records.

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

WHERE Syntax

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

WHERE Clause Example

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

Example

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

Text Fields vs. Numeric Fields

SQL requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes:

Example

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

In, between , like, not in, =, !=, >, <, >=, <=, wildcard operators

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

1. **= Equal Operator :-**

   Example :- SELECT * FROM Products
   WHERE Price = 18;

2. **> Greater Than Operator :-**
   Example :- SELECT * FROM Products
   WHERE Price > 30;

3. **< Less Than Operator :-**
   Example :- SELECT * FROM Products
   WHERE Price < 30;

4. **>= Greater than or equal :-**
   Example :- SELECT * FROM Products
   WHERE Price >= 30;

5. **<= Less than or equal :-**
   Example :- SELECT * FROM Products
   WHERE Price <= 30;

6. **<> Not equal.**
   Note: In some versions of SQL this operator may be written as !=
   Example :- SELECT * FROM Products
   WHERE Price <> 18;

7. **BETWEEN – Between a certain range.**
   Example :- SELECT * FROM Products
   WHERE Price BETWEEN 50 AND 60;

8. **LIKE – Search for a pattern**
   Example :- SELECT * FROM Customers
   WHERE City LIKE 's%';

9. **IN – To specify multiple possible values for a column.**
   Example :- SELECT * FROM Customers
   WHERE City IN ('Paris','London');

SQL Wildcard Characters

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

Wildcard characters are used with the `LIKE` operator. The `LIKE` operator is used in a `WHERE` clause to search for a specified pattern in a column.

Wildcard Characters in SQL Server

Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
^	Represents any character not in the brackets	h[^oa]t finds hit, but not hot and hat
-	Represents 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"

Using the % Wildcard

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

Example

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

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

Example

```
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":

Example

```
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":

Example

```
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":

Example

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

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

Example

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

Using the [!charlist] Wildcard

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

Example

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

Or:

Example

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

SQL ORDER BY, GROUP BY, DISTINCT Keyword

The SQL ORDER BY Keyword

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

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

ORDER BY Syntax

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

ORDER BY Example

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

Example

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

ORDER BY DESC Example

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

Example

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

ORDER BY Several Columns Example1

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;
```

ORDER BY Several Columns Example 2

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

Example

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

SQL GROUP BY Statement

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:

Example

```
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:

Example

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

SQL SELECT DISTINCT Statement

The SQL SELECT DISTINCT Statement

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

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

SELECT DISTINCT Syntax

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

SELECT Example Without DISTINCT

The following SQL statement selects all (including the duplicates) values from the "Country" column in the "Customers" table:

Example

```
SELECT Country FROM Customers;
```

SELECT DISTINCT Examples

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

Example

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

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

Example

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

SQL AND, OR and EXISTS and NOT EXISTS Operators :-

The SQL AND, OR and EXISTS and NOT EXISTS 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 `EXISTS` and `NOT EXISTS` operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

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

AND Example

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

Example

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

OR Syntax

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

OR Example

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

Example

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

EXISTS and NOT EXISTS Syntax

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

EXISTS and NOT EXISTS Example

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

Example

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

Combining AND, OR and EXISTS and NOT EXISTS :-

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

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

Example

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

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

Example

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

SQL Aliases :-

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:

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;
```

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;
```

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

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;
```

MySQL Functions

Aggregate Functions: avg(), max(), min(), sum(), count(), first(),last().

MySQL Numeric Functions

Function	Description
AVG	Returns the average value of an expression
COUNT	Returns the number of records returned by a select query
FIRST()	Returns the first value of selected column.
LAST()	Returns the last value of selected column.
MAX()	Returns the maximum value in a set of values
MIN()	Returns the minimum value in a set of values
SUM()	Calculates the sum of a set of values

MySQL AVG() Function Example :-

Example

Return the average value for the "Price" column in the "Products" table:

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

Definition and Usage

The AVG() function returns the average value of an expression.

Note: NULL values are ignored.

Syntax

AVG(*expression*)

Parameter Values

Parameter	Description
expression	Required. A numeric value (can be a field or a formula)

More Examples

Example

Select the records that have a price above the average price:

```
SELECT * FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```

MySQL COUNT() Function Example :-

Example

Return the number of products in the "Products" table:

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

Definition and Usage

The COUNT() function returns the number of records returned by a select query.

Note: NULL values are not counted.

Syntax

```
COUNT(expression)
```

Parameter Values

Parameter	Description
expression	Required. A field or a string value

SQL FIRST() Function

SQL SELECT FIRST() function returns the first value of selected column.

Syntax:

```
SELECT FIRST(Column_name) FROM table_name;
OR
SELECT FIRST(Column_name) AS First_Name FROM table_name;
```

Example : Query using FIRST() Function
Consider the following table titled as **'Stationary',** which contains the information of products.

Write a query to display a first name from table 'Stationary'.

ID	Name	Quantity	Price
1	Pen	10	200
2	Ink	15	300
3	Notebook	20	400
4	Pencil	30	150

```
SELECT FIRST(Name) AS First_name FROM Stationary;
```

The result is shown in the following table.

First_name
Pen

SQL LAST() Function

SQL SELECT LAST() function returns the last value of selected column.

Syntax:

```
SELECT LAST(Column_name) FROM table_name;
OR
SELECT LAST(Column_name) AS Last_Name FROM table_name;
```

Example : Query using LAST() Function.
Consider the following table titled as **'Stationary',** which contains the information of products.

Write a query to display a last name from table 'Stationary'.

ID	Name	Quantity	Price
1	Pen	10	200
2	Ink	15	300
3	Notebook	20	400
4	Pencil	30	150

```
SELECT Last(Name) AS Last_name FROM Stationary;
```

The result is shown in the following table.

Last_name
Pencil

MySQL MAX() Function

Example

Find the price of the most expensive product in the "Products" table:

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

Definition and Usage

The MAX() function returns the maximum value in a set of values.

Note: See also the MIN() function.

Syntax

```
MAX(expression)
```

Parameter Values

Parameter	Description
expression	Required. A numeric value (can be a field or a formula)

MySQL MIN() Function

Example

Find the price of the cheapest product in the "Products" table:

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

Definition and Usage

The MIN() function returns the minimum value in a set of values.

Note: See also the MAX() function.

Syntax

```
MIN(expression)
```

Parameter Values

Parameter	Description
expression	Required. A numeric value (can be a field or a formula)

MySQL SUM() Function :-

Example

Return the sum of the "Quantity" field in the "OrderDetails" table:

```
SELECT SUM(Quantity) AS TotalItemsOrdered FROM OrderDetails;
```

Definition and Usage

The SUM() function calculates the sum of a set of values.

Note: NULL values are ignored.

Syntax

```
SUM(expression)
```

Parameter Values

Parameter	Description
expression	Required. A field or a formula

Scalar Functions: ucase(), lcase(), round(), mid().

MySQL UCASE() Function :-

Example

Convert the text to upper-case:

```
SELECT UCASE("SQL Tutorial is FUN!");
```

Definition and Usage

The UCASE() function converts a string to upper-case.

Note: This function is equal to the UPPER() function.

Syntax

```
UCASE(text)
```

Parameter Values

Parameter	Description
text	Required. The string to convert

More Examples

Example

Convert the text in "CustomerName" to upper-case:

```
SELECT UCASE(CustomerName) AS UppercaseCustomerName
FROM Customers;
```

MySQL LCASE() Function :-

Example

Convert the text to lower-case:

```
SELECT LCASE("SQL Tutorial is FUN!");
```

Definition and Usage

The LCASE() function converts a string to lower-case.

Note: The LOWER() function is a synonym for the LCASE() function.

Syntax

```
LCASE(text)
```

Parameter Values

Parameter	Description
text	Required. The string to convert

More Examples

Example

Convert the text in "CustomerName" to lower-case:

```
SELECT LCASE(CustomerName) AS LowercaseCustomerName
FROM Customers;
```

MySQL ROUND() Function :-

Example

Round the number to 2 decimal places:

```
SELECT ROUND(135.375, 2);
```

Definition and Usage

The ROUND() function rounds a number to a specified number of decimal places.

Syntax

```
ROUND(number, decimals)
```

Parameter Values

Parameter	Description
number	Required. The number to be rounded
decimals	Optional. The number of decimal places to round *number* to. If omitted, it returns the integer (no decimals)

More Examples

Example

Round the number to 0 decimal places:

```
SELECT ROUND(345.156, 0);
```

Example

Round the Price column (to 1 decimal) in the "Products" table:

```
SELECT ProductName, Price, ROUND(Price, 1) AS RoundedPrice
FROM Products;
```

MySQL MID() Function :-

Example

Extract a substring from a string (start at position 5, extract 3 characters):

```
SELECT MID("SQL Tutorial", 5, 3) AS ExtractString;
```

Definition and Usage

The MID() function extracts a substring from a string (starting at any position).

Note: The MID() and SUBSTR() functions equals the SUBSTRING() function.

Syntax

```
MID(string, start, length)
```

Parameter Values

Parameter	Description
string	Required. The string to extract from
start	Required. The start position. Can be both a positive or negative number. If it is a positive number, this function extracts from the beginning of the string. If it is a negative number, this function extracts from the end of the string
length	Required. The number of characters to extract

More Examples

Example

Extract a substring from the text in a column (start at position 2, extract 5 characters):

```
SELECT MID(CustomerName, 2, 5) AS ExtractString
FROM Customers;
```

Example

Extract a substring from a string (start from the end, at position -5, extract 5 characters):

```
SELECT MID("SQL Tutorial", -5, 5) AS ExtractString;
```