MySQL provides a variety of numeric functions that you can use to perform operations on numeric data types. Here is a list of some commonly used numeric functions in MySQL along with brief descriptions of their usage:

1. **ABS(x)**

* Returns the absolute value of x.

**SELECT ABS(-10);** -- Result: 10

The **ABS(x)** function in MySQL is used to return the absolute value of a numeric expression **x**.

The absolute value is the positive distance of a number from zero, meaning it removes the sign of the number and returns the non-negative value.

Here's an example using the Northwind sample database.

Let's assume you have a table called **OrderDetails** in the Northwind database, and you want to retrieve the absolute values of the **Quantity** column:

USE northwind;

-- Example query using ABS(x) on the Quantity column

**SELECT**

**ProductID,**

**Quantity,**

**ABS(Quantity) AS AbsoluteQuantity**

**FROM OrderDetails**

**LIMIT 10;** -- Limiting to 10 rows for demonstration purposes

In this query:

* We're selecting the **ProductID** and **Quantity** columns from the **OrderDetails** table.
* The **ABS(Quantity)** expression is used to calculate the absolute value of the **Quantity** column, and we alias it as **AbsoluteQuantity**.

Here's a hypothetical result for the query:

+-----------+----------+------------------+

| ProductID | Quantity | AbsoluteQuantity |

+-----------+----------+------------------+

| 10248 | -5 | 5 |

| 10248 | 12 | 12 |

| 10248 | 10 | 10 |

| 10249 | 5 | 5 |

| 10249 | 9 | 9 |

| 10250 | 40 | 40 |

| 10250 | 10 | 10 |

| 10250 | 20 | 20 |

| 10251 | 42 | 42 |

| 10251 | 40 | 40 |

+-----------+----------+------------------+

Let's consider another example using the Northwind database. Suppose you want to find the price difference between the original and discounted prices for products in the **Products** table. You can use the **ABS()** function to calculate the absolute value of the price difference. Here's a query for that:

USE northwind;

-- Example query using ABS() to find the absolute price difference

**SELECT ProductName, UnitPrice, Discount,**

**ABS(UnitPrice - (UnitPrice \* (1 - Discount))) AS AbsolutePriceDifference**

**FROM Products**

**ORDER BY AbsolutePriceDifference DESC**

**LIMIT 10;** -- Limiting to 10 rows for demonstration purposes

In this query:

* We're selecting the **ProductName**, **UnitPrice**, and **Discount** columns from the **Products** table.
* The expression **ABS(UnitPrice - (UnitPrice \* (1 - Discount)))** calculates the absolute value of the price difference between the original **UnitPrice** and the discounted price after applying the discount.
* We order the result by the absolute price difference in descending order to highlight the products with the largest absolute differences.
* Finally, we limit the result to 10 rows for demonstration purposes.

Here's a hypothetical result for the query:

+--------------------------+-----------+----------+---------------------------+

| ProductName | UnitPrice | Discount | AbsolutePriceDifference |

+--------------------------+-----------+----------+---------------------------+

| Côte de Blaye | 263.5 | 0.0 | 263.5 |

| Thüringer Rostbratwurst | 123.79 | 0.0 | 123.79 |

| Mishi Kobe Niku | 97 | 0.0 | 97 |

| Sir Rodney's Scones | 10 | 0.25 | 7.5 |

| Carnarvon Tigers | 62.5 | 0.0 | 62.5 |

| Raclette Courdavault | 55 | 0.0 | 55 |

| Rhönbräu Klosterbier | 7.75 | 0.25 | 5.8125 |

| Gnocchi di nonna Alice | 38 | 0.05 | 1.9 |

| Tarte au sucre | 49.3 | 0.05 | 2.465 |

| Rogede sild | 9 | 0.15 | 1.35 |

+--------------------------+-----------+----------+---------------------------+

1. **CEIL(x) or CEILING(x)**

* Returns the smallest integer greater than or equal to x.

**SELECT CEIL(4.25);** -- Result: 5

The **CEIL(x)** or **CEILING(x)** function in MySQL is used to return the smallest integer greater than or equal to a numeric expression **x**.

Both **CEIL(x)** and **CEILING(x)** are synonyms, and you can use either of them interchangeably.

The function is often used when you want to round up a numeric value to the nearest integer.

Let's consider an example using the Northwind database. Suppose you have a table called **Products**, and you want to find the rounded-up prices for products.

Here's a query using the **CEIL()** function:

USE northwind;

-- Example query using CEIL() to find rounded-up prices without discounts

**SELECT ProductName, UnitPrice,**

**CEIL(UnitPrice) AS RoundedUpPrice**

**FROM Products**

**ORDER BY RoundedUpPrice DESC**

**LIMIT 10;** -- Limiting to 10 rows for demonstration purposes

In this corrected query:

* We're selecting the **ProductName** and **UnitPrice** columns from the **Products** table.
* The expression **CEIL(UnitPrice)** calculates the rounded-up price without applying any discount.
* We order the result by the rounded-up price in descending order to highlight the products with the highest rounded-up prices.
* Finally, we limit the result to 10 rows for demonstration purposes.

Hypothetical Result:

+-------------------------+-----------+------------------+

| ProductName | UnitPrice | RoundedUpPrice |

+-------------------------+-----------+------------------+

| Côte de Blaye | 263.5 | 264 |

| Thüringer Rostbratwurst | 123.79 | 124 |

| Mishi Kobe Niku | 97 | 97 |

| Sir Rodney's Scones | 10 | 10 |

| Carnarvon Tigers | 62.5 | 63 |

| Raclette Courdavault | 55 | 55 |

| Rhönbräu Klosterbier | 7.75 | 8 |

| Gnocchi di nonna Alice | 38 | 38 |

| Tarte au sucre | 49.3 | 50 |

| Rogede sild | 9 | 9 |

+-------------------------+-----------+------------------+

In this result, the **RoundedUpPrice** column represents the prices rounded up to the nearest integer based on the **UnitPrice** values in the **Products** table.

Let's consider another example using the Northwind database. Suppose you want to find the number of units of each product that need to be ordered to meet a certain quantity, and you want to round up the result using the **CEIL()** function. Here's a query for that:

USE northwind;

-- Example query using CEIL() to find the rounded-up quantity needed for ordering

**SELECT ProductName, UnitsInStock, ReorderLevel,**

**CEIL(ReorderLevel - UnitsInStock) AS UnitsToOrder**

**FROM Products**

**WHERE ReorderLevel > UnitsInStock**

**ORDER BY UnitsToOrder DESC**

**LIMIT 10;** -- Limiting to 10 rows for demonstration purposes

In this query:

* We're selecting the **ProductName**, **UnitsInStock**, and **ReorderLevel** columns from the **Products** table.
* The expression **CEIL(ReorderLevel - UnitsInStock)** calculates the rounded-up quantity needed for ordering by subtracting the current **UnitsInStock** from the **ReorderLevel**.
* The **WHERE** clause filters out products where the current stock is equal to or exceeds the reorder level, focusing on products that need to be ordered.
* We order the result by the rounded-up quantity in descending order to highlight products with the highest quantities to order.
* Finally, we limit the result to 10 rows for demonstration purposes.

Here's a hypothetical result for the query:

+------------------------+--------------+--------------+--------------+

| ProductName | UnitsInStock | ReorderLevel | UnitsToOrder |

+------------------------+--------------+--------------+--------------+

| Côte de Blaye | 17 | 20 | 3 |

| Raclette Courdavault | 79 | 80 | 1 |

| Gnocchi di nonna Alice | 21 | 30 | 9 |

| Alice Mutton | 0 | 10 | 10 |

| Outback Lager | 15 | 30 | 15 |

| Thüringer Rostbratwurst| 0 | 10 | 10 |

| Gudbrandsdalsost | 26 | 30 | 4 |

| Guaraná Fantástica | 20 | 30 | 10 |

| Manjimup Dried Apples | 20 | 30 | 10 |

| Grandma's Boysenberry | 120 | 0 | 120 |

+------------------------+--------------+--------------+--------------+

1. **FLOOR(x)**

* Returns the largest integer less than or equal to x.

**SELECT FLOOR(4.75);** -- Result: 4

The **FLOOR(x)** function in MySQL is used to return the largest integer less than or equal to a numeric expression **x**.

It rounds a number down to the nearest whole number, or to the nearest integer that is less than or equal to the given value.

Let's consider an example using the Northwind database.

Suppose you have a table called **Products**, and you want to find the rounded-down prices for products.

Here's a query using the **FLOOR()** function:

USE northwind;

-- Corrected example query using FLOOR() to find rounded-down unit prices

**SELECT ProductName, UnitPrice,**

**FLOOR(UnitPrice) AS RoundedDownUnitPrice**

**FROM Products**

**ORDER BY RoundedDownUnitPrice ASC**

**LIMIT 10;** -- Limiting to 10 rows for demonstration purposes

In this corrected query:

* We're selecting the **ProductName** and **UnitPrice** columns from the **Products** table.
* The expression **FLOOR(UnitPrice)** calculates the rounded-down unit price.
* We order the result by the rounded-down unit price in ascending order to highlight the products with the lowest rounded-down unit prices.
* Finally, we limit the result to 10 rows for demonstration purposes.

Here's a hypothetical result for the corrected query:

+-------------------------+-----------+-----------------------+

| ProductName | UnitPrice | RoundedDownUnitPrice |

+-------------------------+-----------+-----------------------+

| Sir Rodney's Scones | 10 | 10 |

| Rogede sild | 9 | 9 |

| Outback Lager | 15 | 15 |

| Gudbrandsdalsost | 36 | 36 |

| Carnarvon Tigers | 62.5 | 62 |

| Alice Mutton | 39 | 39 |

| Tarte au sucre | 49.3 | 49 |

| Raclette Courdavault | 55 | 55 |

| Côte de Blaye | 263.5 | 263 |

| Mishi Kobe Niku | 97 | 97 |

+-------------------------+-----------+-----------------------+

Let's consider another example using the Northwind database. Suppose you want to find the number of units of each product that need to be ordered to meet a certain quantity, and you want to round down the result using the **FLOOR()** function. Here's a query for that:

USE northwind;

-- Example query using FLOOR() to find the rounded-down quantity needed for ordering

**SELECT ProductName, UnitsInStock, ReorderLevel,**

**FLOOR(ReorderLevel - UnitsInStock) AS UnitsToOrder**

**FROM Products**

**WHERE ReorderLevel > UnitsInStock**

**ORDER BY UnitsToOrder ASC**

**LIMIT 10;** -- Limiting to 10 rows for demonstration purposes

In this query:

* We're selecting the **ProductName**, **UnitsInStock**, and **ReorderLevel** columns from the **Products** table.
* The expression **FLOOR(ReorderLevel - UnitsInStock)** calculates the rounded-down quantity needed for ordering by subtracting the current **UnitsInStock** from the **ReorderLevel**.
* The **WHERE** clause filters out products where the current stock is equal to or exceeds the reorder level, focusing on products that need to be ordered.
* We order the result by the rounded-down quantity in ascending order to highlight products with the lowest quantities to order.
* Finally, we limit the result to 10 rows for demonstration purposes.

Here's a hypothetical result for the query:

+------------------------+--------------+--------------+--------------+

| ProductName | UnitsInStock | ReorderLevel | UnitsToOrder |

+------------------------+--------------+--------------+--------------+

| Alice Mutton | 0 | 10 | 10 |

| Grandma's Boysenberry | 120 | 0 | 0 |

| Manjimup Dried Apples | 20 | 30 | 10 |

| Gnocchi di nonna Alice | 21 | 30 | 9 |

| Tarte au sucre | 17 | 30 | 13 |

| Outback Lager | 15 | 30 | 15 |

| Guaraná Fantástica | 20 | 30 | 10 |

| Gudbrandsdalsost | 26 | 30 | 4 |

| Raclette Courdavault | 79 | 80 | 1 |

| Côte de Blaye | 17 | 20 | 3 |

+------------------------+--------------+--------------+--------------+

1. **ROUND(x [, d])**

* Rounds x to the nearest integer or to the specified number of decimal places (d).

1. **MOD(x, y)**

* Returns the remainder of the division of x by y.

SELECT MOD(10, 3); -- Result: 1

1. **POWER(x, y) or POW(x, y)**

* Returns x raised to the power of y.

SELECT POWER(2, 3); -- Result: 8

1. **SQRT(x)**

* Returns the square root of x.

SELECT SQRT(25); -- Result: 5

1. **EXP(x)**

* Returns the exponential value of x (e^x).

SELECT EXP(2); -- Result: 7.389

1. **LOG(x)**

* Returns the natural logarithm of x.

SELECT LOG(10); -- Result: 2.302

1. **LOG10(x)**

* Returns the base-10 logarithm of x.

SELECT LOG10(100); -- Result: 2

1. **RAND()**

* Returns a random floating-point number between 0 and 1.

SELECT RAND(); -- Result: 0.12345 (example)